PFORD **Language:**

Bhagyashree Bohra
Deepti Jindal (project leader)
Shringika Porwal

December 21, 2004

PFORD **Table of Contents:**

PFORD White Paper

1.1 Introduction

PFORD (programming for dummies) is a language targeting both novice and advanced programmers.

The motivation behind PFORD is to provide an easy to learn, easy to understand language which can be used by novices to quickly familiarize themselves with the art of programming and logical thinking. High level languages like C++ and Java often present a formidable challenge to novices who get bogged down by the non-intuitive presentation of the languages. One has to spend substantial time just understanding the syntax before being able to write a single line of code. PFORD has been designed to overcome this learning curve by using an easy to understand and self-explanatory syntax and semantics. The language employs simple representative English words in the syntax and permits specification of statements that are akin to describing the algorithm to solve the problem at hand. This simplicity will enable a novice to solve problems using high level concepts and not worry too much about the programming details.

PFORD also has another neat feature. The compiler can be instructed to generate equivalent code in any of the popular languages like Java and C++ . A user can easily learn PFORD and simply translate to C++ or Java without having to spend time learning the corresponding languages.

1.2 Language

PFORD is a simple, intuitive, easy to learn, user-friendly, and a powerful language, which also has the capability to translate into other high level popular languages.

1.2.1 Ease-of-Use

The goal of PFORD was to build a system that would be simple for the advanced as well as the novice user. Each language has its own advantages and one may be more suitable than the rest for a project. Programming with PFORD, the user needs to write a program only once and then can generate the corresponding Java code by setting a flag at the beginning of the program. It will be possible to test the result over each platform to observe which of them provides the most optimal result. We also wanted to simplify the process of learning programming for beginners.

By making syntax of the commands similar to English language we have eliminated many of the confusing features of Java/ C++ and have made it easier for the user with minimal programming experience to comprehend PFORD.

For example a loop in C/Java
for(int i=0; i<MAX_VALUE; i++)...

PFORD is syntactically much simpler and visually much easier to understand.

loop MAX_VALUE runs {…}

1.2.2 Object-Oriented

PFORD will allow the user the capacity of creating user-defined classes and let the user generate instances of their objects. This feature will provide the capability of building customized Data Types which can be reused, making programming easier.

1.2.3 Powerful

PFORD offers a beginner the power to perform complex tasks by writing fewer lines of code. This language provides compact commands to perform complex mathematical and string manipulation operations, and also allows the creation of user defined objects. Using PFORD, the programmer has the ability to translate code to Java or C++ making it a powerful tool.

1.2.4 Efficient

PFORD was created to save time and resources for experienced programmers and to eliminate the burden of learning complex languages for the novice users. A single language can now be used to generate code in any of the commonly used high level languages.

1.2.5 Customizable

Not only does PFORD provide the ability to translate to Java, but the user can also run their program with this compiler. It can be used to create customized objects, classes and functions.

1.2.6 Robust

PFORD has been developed as a dependable language. It allows compile-time checks to detect the errors that would be found by Java before generating the corresponding code, thus preventing errors while running the programs on their respective platforms. Another feature of PFORD is that memory management is inherently handled by Java thereby eliminating problems related to memory leaks and making PFORD more robust.

1.2.7 Architecture neutral

PFORD code is compiled to Java code, which is interpreted by Java Virtual Machine (JVM). JVM is a machine independent platform, this makes Java portable to virtually any machine. As PFORD is based on Java, this provides PFORD, the quality of being architecture neutral. PFORD Compiled code can be executed on any processor/platform wherever JVM will run.

1.2.8 Data Types

PFORD supports several Data Types such as *integer, (double) float, and string*. These Data Types follow the same fundamental pattern as C++/Java. In addition, user defined objects and arrays are also provided. Arrays are used to represent any of the Data Types listed above. The basic algebra operations (addition, subtraction, multiplication, division) will be supported for integers, doubles as well as arrays of these basic Data Types.

1.2.9 Basic Functions/ Libraries

The three basic libraries in PFORD are math, sorting and string.. Some of the functions supported include sorts on integer and string arrays, string and substring manipulations, and some array functions.

1.2.10 Interpreted

The lexer and parser will be created using ANTLR. ANTLR will generate the required information for PFORD interpreter to execute the user's program. Since PFORD language is not compiled to machine code, it's interpreter will work the same on all the platforms and architectures.

PFORD **Tutorial:**

2.1 Getting Started with Pford

To write a simple program using Pford first open a text editor and save the new file with extension ".pford".
In the following example our file will be called test.pford.
Statements in a Pford program must be written between "startprogram" and "endprogram".
Suppose we want to output the tranditional "hello world". Then in pford it will be written as:

File : **test.pford**

```
startprogram
output("Hello World");
endprogram
```

2.2 Executing a Pford Program

To execute this program on command line type java Main test.pford. and it will print on screen

Hello World

2.3 Translation

To execute translation with the same hello world program at the command line you simply type java Main test.pford <newfilename>. The newfilename must be given the extension either .cpp or .java in order to translate to Java or C++.

For example

java Main testoutput.pford my.java will print out

Hello World

And create a file called my.java. which contains the following

```
class my{
   public static void main(String[] args){
```

```
        System.out.print("Hello World");
    }
}
```

Similarly with C++ the output is generated as follows

java Main testoutput.pford my.cpp

Hello World

The file my.cpp is created

```
void main(){

  cout <<"Hello World";
}
```

2.4 Examples

another example using variable declaration,comments and functions –

File: **test1.pford**

start program
a=10;
b=”Hi”;
c=2.0;

```
func sum_all() //function declaration
{
  w=0.0;
  w = a + c
      return w;
};
p=sum_all(); //function call
```

output p;
endprogram
when run this program prints value of p as “12.0”

A more complex program including loops and ifs

File : **test3.pford**

```
startprogram

/*single line*/
```

```
foo = 3 + 4;

output "Hello world!";

a = 1;
if (1 > 2) then
{
      a= 3;
}
elsedo
{
  b = 3 + 4 * 5 + 6 * 7;
};

output "b should be 65";
output b;

a += 5;


b= 3 * 4 + 5 * 6;


c= b + a;
output "c is 48";
output c;

b = 0;
loop 11 runs
{
b = b+1;

};

loopfrom 12 to 15 interval 1 { b = b + 1;};

output "b shoud be 14";
output b;

func increment(a, c, b)
{
b = a+ c;
return b;
};

d= increment(1,1,4);
output "value of d is 2";
output d;
endprogram
```

PFORD Language Reference Manual:

3.1 Introduction

The motivation behind PFORD is to provide an easy to learn, easy to understand language, which can be used by novices to quickly familiarize themselves with the art of programming and logical thinking. PFORD also has another neat feature. The compiler can be instructed to generate equivalent code in any of the popular languages like Java.

3.2 Lexical Conventions

Token types include identifiers, keywords, constants, strings, operators and separators, where delimiters are used to separate tokens. A token is a sequence of characters that may be matched to any token.

3.2.1 Whitespace

White space is either a blank, tab or new-line. At least one of these is required to separate tokens, otherwise it is ignored.

3.2.2 Comments

PFORD supports C/C++ style comments. A multi-line comment begins with the characters /* and terminates with */ . A single line comment starts with characters // and continues till the end of line. Examples are as below—

```
/* This is an example of
   multi-line
   comment */

// This is an example of single line comment
```

3.2.3 Identifiers

Identifiers basically consist of program variable names for numbers, strings, arrays of functions. An identifier is considered a sequence of letters, digits or underscores (_) that must begin with a letter and is not a keyword. In PFORD lowercase and uppercase letters are distinguished and not considered the same i.e. ‘A’ is not the same as ‘a’.

3.2.4 Keywords

A keyword is an identifier used as reserved word of the language, which cannot be used for naming identifiers such as variables functions etc. Following is the list of keywords for PFORD:

```
startprogram      endprogram
include           OR                  AND              NOT
while             loop                runs             loopfrom
to                interval            skip_iteration   break_loop
if                then                elsedo
output            input               func             mod
```

3.2.5 Spatial Delimiters

( ) Parentheses are used around parameter lists
{ } Braces are used in loops or in around functions
[ ] Brackets are used to in array declarations
; Semicolon must be at the end of each statement
, Comma are used to separate parameters in a parameter list.
. Dot are used to indicate parameters of an object

3.3 Data Types

PFORD supports multiple Data Types. The types can be categorized into primitives and non-primitives.

The primitives include numbers, strings, Boolean, and the non-primitive include arrays and objects.

3.3.1 Numbers

Numbers include integers and floating point numbers. PFORD will be able to internally figure out whether a number is either a floating point or integer.

Digit: 0..9
Digits: digit digit*
optional_fraction: . digits |empty
optional_exponent: (E (+|-|empty)digits) |empty
num: digits optional_fraction optional_exponent

Examples:
a= 1;
b= 1.2;

3.3.2. Strings

Strings are composed of any combination of characters.

a= "Compilers";
b= "abc123";

3.3.3 Boolean

Boolean type is used to store true or false values. False is a value of 0, true is any non zero value

3.3.4 Arrays

Arrays are used to store many items of one type. To make programming simpler, the indices will be numbered from 1 and not 0. Users can declare arrays of any of the Data Types mentioned above and also create array of objects (defined below).
//TODO
Example: number example_array [length];

3.3.5 Objects

PFORD allows users to define their own Data Types. Object can be declared to consist of multiple primitive Data Types. However, the objects can't declare functions in their definition of the object. The declaration of an object is done as follows:

object object_name {

```
        data_type name;
        data_type name1;
};
```

After the object is declared, new instances of the object can be created using other variable declarations:

object_name x,y;

3.4 Variables

3.4.1 Scoping

Variables can be assigned scope as either Global or Local.

3.4.1.1 Global

If a variable is declared outside any function block of code, then it has the Global scope and can be used throughout.

3.4.1.2 Local

A variable has a local scope if it is declared inside a code block such as inside a function. Local variable is accessible and available only inside the block in which it is declared.

3.5 Operators

3.5.1 Unary operators

Unary operators '-' and '+' act as a prefix to an operand. The operand should be a number. While '+' operator do not change the value of operand, '-' operator returns the result of operation as, negation of given operand.

For example
+(42.05) returns 45.05
-(82.889) returns -82.889

Other examples of unary operators are +=, -=, *=. When these operators are used the variable will perform the operation to itself.

For example
a += 5; //this will add a to itself.

3.5.2 Binary operators

3.5.2.1 Multiplicative operators:

Multiplicative operators take two operands of type number and return the output of the operation. They are grouped left to right.
* is for multiplication
/ is for division
% indicates percent
mod indicates modulo

3.5.2.2 Additive operators

Binary operators '+' and '-' indicate addition and subtraction respectively.

'+' can either add two numbers or concatenate a string
'-' subtracts numbers

3.5.2.3 Power operator

Power operator ('^') is used to raise an operand of type number to a certain degree. '^' may be followed by optional '+' or '-' and mandatory number(integer).

3.5.2.4 Relational operators

>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
==	equal to
!=	not equal to

These operations can be applied on operands of type number or regular expressions of type number. The result of this comparison is either 0 (True) or 1(False).

3.5.2.5 Logical operators

Logical operators 'OR', 'AND' and 'NOT' take regular expressions as operands.

OR operator : indicates the 'logical or' of the two regular expressions returns true if any of the regular expression is true.

e.g. expr OR expr

AND operator: indicates the 'logical and' of two regular expressions and returns true if both of them are true.

e.g. expr AND expr

NOT operator: is a unary operator. It takes a regular expression as operand and negates it.

NOT (expr)

3.5.2.6 Assignment Operator(=)

Assignment operator '=' is used to assign a value to an identifier. The value can be of type number or string.

3.6 Operator precedence

The precedence of operators in an expression is shown in descending order in the following explanation. The operators included in the same subsection have equal precedence.

Precedence Level	Operator	Explanation
1	[], (), function call, !	Array indices, expressions in parenthesis, function calls, negation
2	^	power
3	+, -	Unary Operators positive or negative number
4	*, /, % and mod	Multiplication , division, percent and modular division
5	+, -	Addition and subtraction
6	> , >= , <, <= ,	Relational operators
7	==, !=	Equality operators
8	AND, OR	logical operators
9	=	Assignment operator

3.7 Operations on Data Types

The grammar for doing operations on numbers will be
num operator num

For string the main operator that will be supported is concatenation which is
strings + strings

The grammar for relational operators

A relationalop B

3.8 Constructs:

3.8.1 Conditionals

The conditional if statement is used to alter the execution of the program based on whether the specified condition is valid.

The format of the if statement is as follows:

- if (<expression>) then
  {
  program statements;
  …
  };

If the “condition to be tested” is valid (evaluated as true), then the program statements are executed. Or else they are skipped.

- if (<expression>) then
  {
  program statements1;
  …
  }
  elsedo {
  program statements2;
  …
  };

In this case, if the “condition to be tested” is valid (evaluated as true), then the program statements1 block is executed. If invalid (evaluated as false), the program statement2 block is evaluated.

3.8.2 Loops

3.8.2.1 Simple loop

This loop allows us to iterate through a loop a specified number of times.

loop <number> runs {
program statements;
…

```
};
```

This case will cause the program statements specified in the braces to be executed as many times as specified by num.

3.8.2.2. Specific loop

This loop allows us to iterate through a loop beginning from a “minimum” value and going to the “maximum” value at a specified interval.

```
loop from <number> to <number> interval <number>
{
        program statements;
        …
};
```

This will cause the program statements in braces to be executed, starting from the minimum (beginning) value to the maximum (final) value at an interval specified.

3.8.2.3 While loop

The first variation allows the user to iterate through the statements in the braces as long as the specified condition is true.

```
while (<expression>) loop through {
        program statements;
                …
        };
```

3.8.3 Special Statements

3.8.3.1 Skip iteration

This statement, when encountered, in any loop construct as specified above, skips the remaining part of the current iteration

```
skip_iteration;
```

3.8.3.2 Break loop

This statement causes the entire loop to be skipped or “broken out of”. All the remaining iterations, including the current one will be skipped.

```
break_loop;
```

3.9 Input/Output

3.9.1 Output

For printing output to the screen, the following is used:

output “string to be outputted” + <identifier>;

Strings in double quotes are printed exactly as written. The string equivalent of the variables is printed. In the case of numbers and strings, the value of the variable is printed, whereas for an array or hash its contents are listed. If the user would like to print the individual array element then the user must specify the element with its indices.

3.9.2 User Input

input <identifier>

Here the user enters information from screen. Accordingly, the user inputs values on the command line, which are read into the variables one by one. If the number of variables specified and inputted by the user does not match, then an error message is displayed. Also if the Data Types do not match (for e.g.: number given when string expected), an error is displayed.

3.10 Expressions

Expressions are the next unit of analysis after identifying the tokens.
Expressions can be defined as:

expr: expr op expr --> (op is an operators like +,-, *, /)
expr: digit
expr: (expr)

3.11 Statements

Statements form one coherent unit of a program

Statements can be defined as:

stmt: identifer=expr
| if expr then stmt
| loop <number> runs stmt
| loop from <number> to <number> interval <number> stmt

| while expr loop through stmt
| loop through stmt while expr
| output expr
| expr

3.12 Functions

3.12.1 Function Definition

FunctionDecl
“func” FunctionDeclarator Parameter? FunctionBody

FunctionDeclarator
Identifier ‘(‘ (Parameter)? ‘)’

Parameter
Parameter parameter-list

Paramater-list
(‘,’ Parameters | /*nothing*/)*

FunctionBody
‘{‘ (Declarations)? Statement‘}’

User defined functions are declared as follows:
Func function-name(param1, … , paramN)
{
//function body
return retValue;
}

An example using PFORD
func function_add (a, b);
{
	return a+b;
};

When you want to call the function from within the main function the user uses call function.

Using the example function_add from before
a=0;
b= 0;
function_add (a,b);

In pford we can also do much more complex functions such as nested functions.

```
func addmynums(a,b,d)
{
	//e=foo(a);
	c=a+b+d;
	return c;
}
;
func mymy(a,b,d)
{
	func mymymy(m)
	{
		output m;
	};
	mymymy(3);
	output "mymymy";
	a+=1;
	mymymy(a);
}
;

l=3;
m=4;
n=10.5;
r=addmynums(l,m,n);
output r;
mymy(l,m,n);
```

3.12.2 Built-in functions

- sort will return a sorted array either by alphabetical order or by number order
  sort (array);
- sum will return the summation of the whole number array
  sum(array);
- min will return the minimum of the number array
  x=min(array);
- max will return the max of the array
  x= max(array);
- mean will return the average of the number array
  x= mean(array);
- std_dev will return the standard deviation of a number array
  x=std_dev(array);
- contains this will search for the string_match in the ‘string_tar’ (the target string). Options can be specified. For example, I – would stand for match ignoring case, g for global matches.
  contains (string_tar, string_match, options)
- to_capital will return a string array or a string in all caps
  to_capital(stringTemp)

- to_lower will return a string array or a string in all lower case
  to_lower(stringTemp)

PFORD Programming Languages and Translators:

4.1 Scoping

In PFORD the scope is done globally in reference to loops and if statements, but in functions the scope of a variable ends after the function body ends.

In lexical scoping you search in the local function then you search in the function in which that function was defined. Then you search in the function in which that function was defined In dynamic scoping, by contrast, you search in the local function first, then you search in the function that called the local function. From the following program the output was 10 meaning that PFORD uses dynamic scoping as expected per the design of the PFORD. Under lexical scoping the output would have been a 5.

startprogram

```
 /****pford has dynamic scoping***/
func lambda(x)
{
      x=5;
      func gimmex(x)
      {
            lambda(x);
      };

      x=10;
      gimmex(x);
};
endprogram
```

4.2 Binding

In deep binding, lexical variables mentioned in anonymous subroutines are the same ones that were in scope when the subroutine was created. In shallow binding, they are whichever variables with the same names happen to be in scope when the subroutine is called. From the following program the output was 1 meaning that PFORD uses deep binding. Under shallow binding the output would have been 2.

```
startprogram
/***pford has deep binding**/
func a(i,p)
{
```

```
      func b()
      {
            output i;
      };
      if(i==1)then
      {
            a(2,b());
      }
      elsedo
      {
            q();
      };
};
func q()
{
 dummy =1;
};
a(1,q());
endprogram
```

4.3 Order

There are two types of order applicative and normative. In Applicative order evaluation the arguments are evaluated before, and in normal order arguments are evaluated when used. From the following program the output was 8 and 11 meaning that PFORD uses applicative order. Under normal order the output would have been a 9 and 2.

startprogram

```
/*************
*our functions are applicative order
********/

w = 3;
x = 10;
func incw()
{
      w=w+1;
      return w;
};
func incx()
{
      x=x+1;
      return x ;
};
func myfoo(y, z)
{
   output y+y;
   x = 1;
   output z;
};

myfoo(incw(), incx());
```

endprogram

PFORD **Project Plan:**

5.1 Scheduling

To make sure we stayed on task we set up two meetings a week and if any problems arose we met with the TA during her office hours before our next meeting. This set the pace for the project to get complete on time.

5.2 Planning and specification phase

Project planning is the key to project success. Our Initial project meetings were focused on designing the specification for the programming language that we wanted to implement, defining the functionality specifying features of language. These meetings and further references to manuals of popular programming languages helped us define exactly what we wanted our program to achieve and plan for achieving those goals in a time restricted schedule. All these specification and goals were included in Language Reference Manual(LRM) which served as important guideline through out the project.

5.3 Development

During the development phase of our project, we decided to meet twice during the week. Even though at times we worked on different modules, we discussed the problems faced in development and tried to debug the code in pairs during meetings, so that debugging went quickly and every one of us was familiar with all the modules. The benefit of this developments process was there wasn't any gap in communication. During the final phase of development of PFORD we worked together to bring the pieces together.

5.4 Testing

Testing is an essential part in checking the correctness of the desired language. We wrote test programs to check each functionality individually and then did the integrated testing for various features. Then the final version of our language underwent rigorous testing.

5.5 Team Responsibilities

Reading the reports from previous projects we found many complaints about unnecessary time spent in merging code. We felt in order to keep from wasting too much time later on we would meet with at least two people present in all meetings, preferably not the same two all the time. It is faster to program especially early on in the project that way everyone has a common base and knows how everything works. This also makes debugging in the beginning much faster. For instance, when one team member can not figure out the error then another may figure it out. Another advantage of coding together is it helps to keep a steady pace since everyone codes for the number of meeting hours straight.

After that there came points when work needed to be divided up to utilize maximum meeting time. However even when labor was divided up all members were available to help each other out with problems. If we felt the project was falling a week behind schedule due to testing phases, assignments were made for the following week to catch up.

5.5.1 Division of Labor

	Lexer	Parser	AST Parser	Walker	Interpreter	Translation	Built in Functions	Arrays	Datatype stuff	Initial Testing	Final Testing Suite	Presentation	Report
Deepti Jindal (project leader)	X	X	X	X	X	??	X			X	X	X	X
Bhagyashree Bohra	X	X	X	X	X	??	X			X	X	X	X
Shringika Porwal	X	X			X	??	X	X	X	X	X	X	X

5.6 Programming style (coding conventions)

5.6.1 Antlr coding style

- The format for a rule in Antlr is
  Name:
  Statement
  ;
- If name is a token then it should be written in UpperCase Letters.If name is nonterminal then it will be written in lowercase.
- Underscore(_) can be used in names for both for tokens and nonterminals.
- A statement can contain multiple options separated by (|) and the optional statements should be unambiguous. If an option was long then each option was put on its own line.
- All rules must be terminated by ;
- Actions code is written inside {} and follows java coding conventions.

5.6.2 Java coding style

- All class names must begin with prefix "Pford" followed by words whose inititial letter is in uppercase.
- Left brace({) should occupy its own line
- Right Brace(}) occupies a full line.
- Variables are in lowercase. Tried to use Hungarian notation wherever possible.
- No space between function name and its arguments in ().
- Used tabs for indentation at different levels like class definition, method declarion, variable declaration and body in if –elsestatements and in functions.

5.7 Project Timeline

September 28	White Paper
October 21	LRM
October 21	Lexer
October 21	Parser
October 28	AST Parser
November 5	Walker
November 21	Interpreter
November 21	Arrays
November 21	Datatype stuff
November 28	Built in Functions
December 5	Initial Testing
December 10	Final Testing Suite
December 16	Translation
December 20	Presentation
December 20	Report

5.8 Software Development Environment

5.8.1 Operating Systems

Since Pford has been designed in pure java, programming in Pford is possible on all the platforms running JVM(java virtual machine). Most of the programming was done in the CLIC lab on Linux

5.8.2 Java 1.4.0

Most programs are written in java. Lexer, parser and tree walker are written using ANTLR and are translated into java code.

5.8.3 ANTLR 2.7.4

ANTLR, Another Tool for Language Recognition, is a language tool that gives a framework for constructing recognizers, compilers, and translators from grammatical descriptions containing Java, C#, or C++ actions. ANTLR provides excellent support for tree construction, tree walking, and translation. The Pford Language lexical analyzer, parser and tree walker are written using ANTLR.

5.8.3 CVS

CVS(concurrent versions system) is an open source software designed to help implement version control and configuration management of software projects. We used CVS for version control of our project. Our source code was on a Linux server and we used SSH to check in/check out project code from there for working on most updated version of our project.

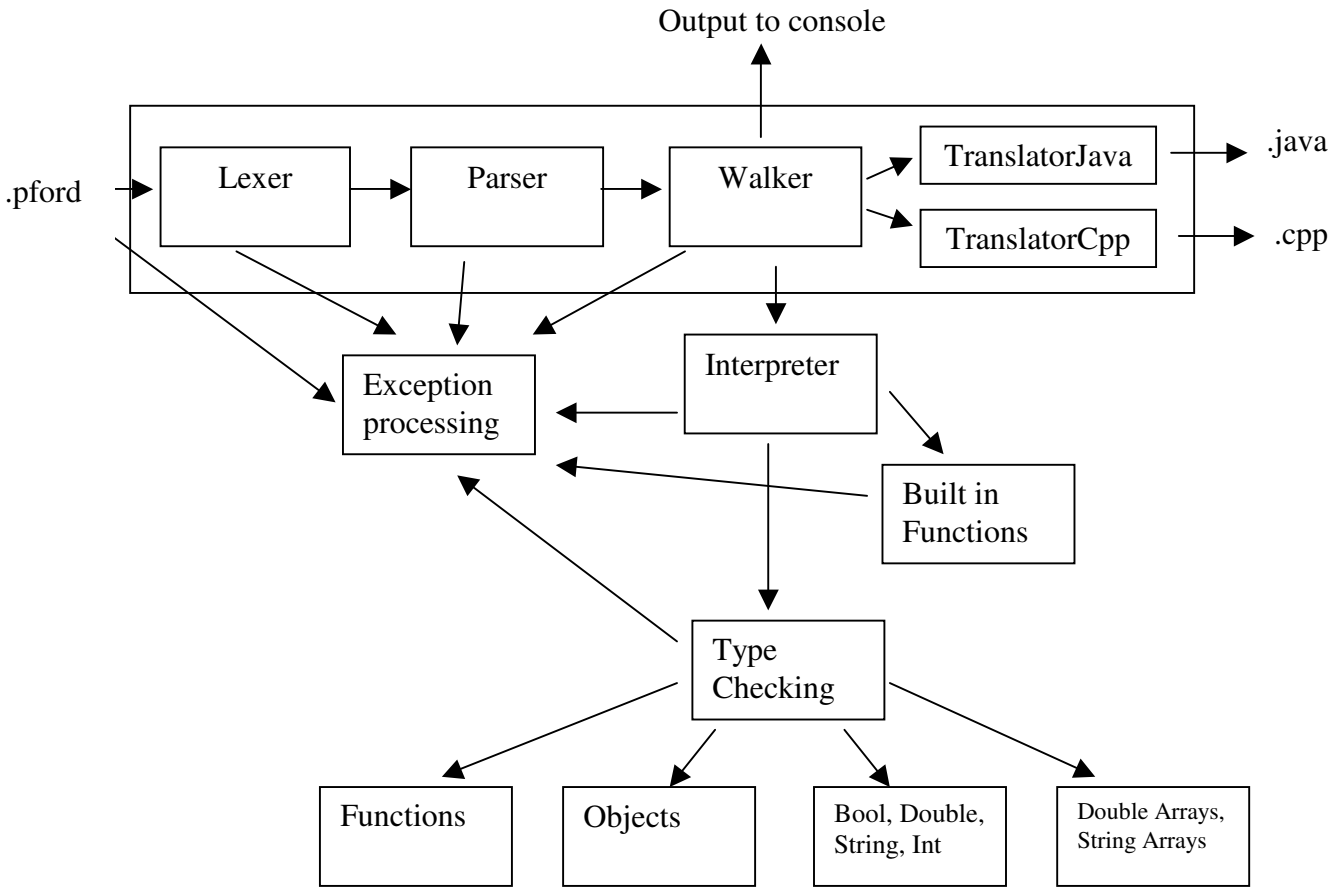
5.9 Project log

September 28	White Paper
October 21	LRM
October 26	Lexer
October 26	Parser
November 5	AST Parser
November 12	Walker
November 12	Arrays
November 12	Datatype stuff
November 28	Interpreter
December 5	Built in Functions
December 8	AST Test Final
December 9	Initial Testing
December 16	Translation
December 18	Final Testing Suite
December 19	Presentation
December 19	Report

PFORD **Architectural Design:**

6.1 Block diagram

Pford source code is feed into the lexer. If the code does not have the extension .pford then an error is passed back. The lexer then produces a collection of tokens according to language PFORD. The parser then analyzes the syntactic structure of the program. If any error has occurred in the lexer or parser stages then an error is passed back. Once everything has been parsed correctly the abstract syntax tree is created and is fed into the tree walker. The walker will then travel the syntax tree and accordingly perform the operations through the Interpreter and output to the console. The Interpreter will take care of all type castings issues, symbol tree creation, built in functions, etc…. If at the command line the user puts two arguments (ie <filename.pford> <newfilename.cpp/newfilename.java> then the code will also go through the translation walker which will create the corresponding Java or C++ files.



6.2 Who did what

	Lexer	Parser	AST Parser	Walker	Interpreter	Translation	Built in Functions	Arrays	Datatype stuff	Initial Testing	Final Testing Suite	Presentation	Report
Deepti Jindal	X	X	X	X	X	X	X			X	X	X	X
Bhagyashree Bohra	X	X	X	X	X		X			X	X		
Shringika Porwal		X			X		X	X	X	X			X

PFORD **Test Plan:**

Testing is a cumbersome procedure because it is essential and must cover every possible combination otherwise results will not be consistent thus making the language weak. Testing must be done at the numerous stages of the compiler development from the very beginning of the implementation.

7.1 Initial Testing

As each component of the lexer and parser were made, small test cases were created to make sure some thing is output correctly. When the correct ASTs and output values from the initial testing passed, then we knew our program "worked". However this was only a general case to get a beta version of our compiler.

7.2 Testing

Once we were near completion of the compiler then the test files were made much more comprehensive to cover all possible test cases. The tests were kept in a database of test files. Test cases were also made to make sure error statements were given at correct locations.

7.3 Regression Testing

If any test failed in the process then all the tests from the database were rerun to make sure that nothing broke in the process of correction. Since the process of testing increased over time and the number of files also increased, we automated the process by putting the files into one file. Once the file ran, the output was cross referenced with the correct test values.

7.4 Sample of our testing database

```
startprogram

sequence=[90,20,30,40];
print("sequence is");
print(sequence);

print("3rd element in sequence is");
print(sequence[2]);
```

```
print("average of sequence is");
avgseq=mean(sequence);
print(avgseq);
print("minimum in sequence is");
minimum=min(sequence);
r=range(sequence);
print(minimum);

print("range is");
print(r);

sizearray=size(sequence);
print("size of array sequence is");
print(sizearray);


print("result after multiplying with 10");
seq10times=sequence*10;
print(seq10times);

output "test assoc.: 2+3=5";
a=2+3;
output a;

output "test assoc.: 2+3*4=14";
a=2+3*4;
output a;

output "test assoc.: 2+3*-4=-10";
a=2+3*-4;
output a;

output "test assoc.: 2*2+3*4=16";
a=2*2+3*4;
output a;

output "test assoc.: 2/2+1=2";
a=2/2+1;
output a;

output "test assoc.:-2/2+1=0";
a=-2/2+1;
output a;

output "test assoc.:2^3=8";
a=2^3;
output a;

output "test assoc.:-2^3=-8";
a=-2^3;
output a;


output "test assoc.:9.0^0.5=3";
a=9.0^0.5;
output a;
```

```
output "test assoc.:9.0^-0.5=0.33";
a=9.0^-0.5;
output a;

output "test assoc.:2.2^3.3=13.76";
a=-2.2^3.3;
output a;

output "test assoc.: 2.0+3.11=5.11";
a=2.0+3.11;
output a;
```

endprogram

7.5 Who did what

	Output	Precedence Rules	Variables	Types	Ifs	Loops	Functions	Built in Functions	Arrays	Objects	Inputs	Translation
Initial testing												
Shringika	X	X	X					X	X			
Bhagyashree	X	X	X	X			X					
Deepti	X	X		X	X	X	X	X		X	X	X
Final testing												
Shringika								X	X		X	
Bhagyashree	X	X	X	X	X	X	X	X				
Deepti				X	X		X			X	X	X

PFORD **Lessons Learned:**

8.1 Deepti

I learned a lot about the design of a language and the complexity that goes into specifying a complete and nonambiguous grammar during the course of this project. After the project was complete we were proud to see so many lines of our own code working as specified. The other thing that I liked about the project was being able to apply the concepts that were taught in class (ie applicative order, dynamic scoping and dynamic binding…) I think the biggest lesson I learned from this project was the importance of organizational skills. I feel since we set up our meetings and deadlines and reevaluated our deadlines on a weekly basis we were able to finish on time. We had our lexer and parser complete by the time LRM was submitted and made sure to have 2 sessions per week after that. The testing process towards the end went very smoothly because we organized ourselves well so early on. One word of advice to future projects, I feel testing would have been easier if from the beginning we had a better idea of how all the pieces comes together.

8.2 Bhagyashree

Be clear about the exact functionality provided by the tools as early on as possible. This will save valuable time. It is also very important to start early and have a well-defined time line. Our group followed this and hence things went pretty smoothly with a few glitches here and there. Also, try to get input from experienced people. This will help in figuring out problems faster. It will be useful to have more extensive test cases planned out in advance. It is also important to learn the working styles of the group members early on. This will result in a smoother functioning. We had two meetings per week and we worked together for 4 hours per meeting. This was very helpful since not only did the work get done on time but also was very helpful in ensuring that everyone was familiar with all portions of the project. This was critical in integration and bug fixing.

8.3 Shringika

The most important lesson that I learned from this project is to start working on the project as early as possible. Keeping a calendar for achieving goals definitely helps and reduces the work to be done at last moment when the project is due. Keeping group meetings every week on a regular basis kept each one of us updated about the progress as well as bugs in our code.
Writing small pieces of code and then exercising that code, makes it possible to fix bugs in code at initial stage. While writing Parser for our language, to include new functionality, we made lot

of changes at a time then after comiling code got lots of errors and it was really difficult to point out where exactly the problem started.
The other very important lesson that I learnt is to work in a team and communication. Communicating with other team members is definitely the key for successful implementation of project. As this project involved various modules and every team member is assigned different modules to work, communicating with group members and asking questions/ discussing integration issues definitely reduces the time to merge all modules.

8.4 Advice for future groups

Fully investigate the feasibility of the supporting tools before deciding what project you would like to do. We decided on doing an easier form of programming which also allows translation to other complex languages. Translation was actually a much more complex task then expected. The other thing is to make sure that you organize your self early and set up deadlines, you never know what can go wrong at the last minute.

PFORD **Language Syntax:**

9.1 Lexical rules

letter - 'a'..'z' | 'A'..'Z' | '_'
digit - '0'..'9'
id - letter (letter |digit)*
number - `(DIGIT)+ ('.' (DIGIT)*)? ('e'(PLUS | MINUS | /*nothing*/) (DIGIT)* )?`
string - `"'! ( '"' '"'! | ~('"'))*  '"'!`

Skipped tokens
linebreak - '\n'
whitespace - space | tab | '\r'| linebreak
comment - ('/*' (~'*/')* '*/') | '//' (~linebreak)* linebreak

9.2 Syntactic rules

program - (statement ';')*
statement - for-stmt | if-stmt | loop-stmt | break-stmt | return-stmt | assign-stmt | input-stmt |
output-stmt | func-decl | func-call | object-def
loop-stmt - 'loopfrom' id 'to' id 'interval' body
| 'loop' '(' id ' | number ')' ? runs body
| 'while' '(' rlogexpr ')' body
if-stmt - 'if' '(' expression ')' 'then' body ('elsedo' body)?
break-stmt - 'break_loop'
cont-stmt - 'skip_iteration'
return-stmt - 'return' expression
include-stmt - 'include' string
assignment - l-value ('=' | '+=' | '-=' | '/=') expression
func-call - id '(' expr-list ')'
expr-list - expression (',' expression)* |
func-def - 'func' id '(' var-list ')' body
var-list - id (',' id)* | ²
body – '{' (statement ';')+ '}'
expression - logic-factor (('or' |'and') logic-factor)*
logic-factor - relat-expr (('==' | '!=')? relat-expr)*
relat-expr - arith-expr (('>=' | '<=' | '>' | '<') arith-expr)?
arith-expr - arith-term (('+'|'-') arith-term)*
arith-term - power (('*'|'/'|'%'|'mod') power)*
power- expr ('^' expr)*

expr - ('+'|'¡') r-value | r-value '0'
r-value - l-value | func-call | number | string |objcall | true | false | array
l-value - id ('[' expression ']')* (dot id)?
objectdef – “object” id body
output – ‘output’ expression
input– ‘input’ id

PFORD **Appendix:**

10.1Source Code

10.1 .1Parser.g

```
/**
 *insert type's descritpion here.
 *Creation Date: (2004-11-08 09:20:10)
 *author: deepti, bhagyashree
 */
class PfordLexer extends Lexer;

options {
    testLiterals = false; // By default, don't check tokens against keywords
    k = 2;                // Need to decide when strings literals end
    charVocabulary = '\3'..'\177';
    exportVocab = PfordAntlr;

}
{
    int nr_error = 0;
    public void reportError( String s ) {
        super.reportError( s );
        nr_error++;
    }
    public void reportError( RecognitionException e ) {
        super.reportError( e );
        nr_error++;
    }

}
SEMI : ';';
PLUS : '+';
POWER  : '^' ;
STAR : '*';
MINUS  : '-' ;
DIV    : '/' ;
ASSIGN : '=' ;
RSQBRACKET: ']' ;
LSQBRACKET: '[' ;
LCURLY: '{' ;
RCURLY: '}' ;
PERCENT : '%';
COMMA : ',';
DOT   : '.';
UPLUSEQ : "+=";
UMINUSEQ : "-=";
UMULTEQ : "*=";
```

```
UDIVEQ : "/=";
REQ : "==";
RGEQ : ">=";
RLEQ : "<=";
RNEQ : "!=";
RGT : '>';
RLT : '<';
INCR: "++";
DECR: "--";

PARENS
options {
    testLiterals = true;
}
    : '(' | ')' ;


protected LETTER : ( 'a'..'z' | 'A'..'Z' ) ;
protected DIGIT : ('0'..'9')  ;

ID
options {
    testLiterals = true;
}
    : LETTER (LETTER | DIGIT | '_')* ;

NUMBER : (DIGIT)+ ('.' (DIGIT)*)? ('e'(PLUS | MINUS | /*nothing*/) (DIGIT)*
)? ;

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

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

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

```
/**
 *insert type's descritpion here.
 *Creation Date: (2004-11-08 09:20:10)
 *author: deepti, bhagyashree
 */
class PfordParser extends Parser;

options {
    buildAST = true;
    exportVocab = PfordAntlr;
    k = 2;           // Need to distinguish between ID by itself and ID
ASSIGN
}

tokens {
    STMT;
    UPLUS;
    UMINUS;
    FUNCCALL;
    VARLIST;
    FUNCVARLIST;
    ARRAY_ROW;
    ARRAY;
    OBJCALL;
}

{
    int nr_error = 0;
    public void reportError( String s ) {
        super.reportError( s );
        nr_error++;
    }
    public void reportError( RecognitionException e ) {
        super.reportError( e );
        nr_error++;
    }
}

program
    :
        "startprogram"! (stmt SEMI!)+ "endprogram"!
        { #program = #([STMT, "PROG"],program); }
    ;

ifStmt
    :
        "if"^ "("! rlogexpr ")"! "then"! body //LCURLY! (stmt SEMI!)+ RCURLY!
        (options {greedy=true;} : "elsedo"! body)?//LCURLY! (stmt SEMI!)+
RCURLY!)?
    ;

outputStmt
    :
        "output"^ rlogexpr
    ;
```

```
inputStmt
    :
        "input"^ ID
    ;

assignStmt
    :
        lvalue (ASSIGN^|UPLUSEQ^|UMINUSEQ^|UMULTEQ^|UDIVEQ^) rlogexpr
    ;

funcvarlist:
        //("number"^ |"string"^ | "bool"^ ) ID (COMMA! ("number"^ |"string"^
| "bool"^ ) ID)*

        ID (COMMA! ID)*
        {#funcvarlist = #([FUNCVARLIST,"FUNCVARLIST"],funcvarlist); }

    |/*nothing*/
        {#funcvarlist = #([FUNCVARLIST,"FUNCVARLIST"], funcvarlist); }
    ;

funcdecl:
        // "func"^ ("number"^ |"string"^ | "bool"^ |"nothing"^) ID "("!
(funcvarlist)? ")"! funcbody
        "func"^ ID "("! funcvarlist ")"! body
    ;

body:
        LCURLY! (stmt SEMI!)+ RCURLY!
        {#body = #([STMT,"BODY"], body); }
    ;

returnStmt:
        "return"^ (rlogexpr)?
    ;

varlist:
        rlogexpr(COMMA! rlogexpr)*
        {#varlist = #([VARLIST,"VARLIST"], varlist); }
    |/*nothing*/  //no parameters for the function call
        {#varlist = #([VARLIST,"VARLIST"], varlist); }
    ;

funccall:
        ID "("! varlist ")"!
        {#funccall = #([FUNCCALL,"FUNCCALL"], funccall); }
    ;

objcall:
        ID DOT! ID
        {#objcall = #([OBJCALL,"OBJCALL"], objcall); }
    ;

loopStmt
    : "loop"^ (ID | NUMBER ) "runs"! body //LCURLY! ( (stmt | loopopttail)
SEMI!)+ RCURLY!
```

```
    | "while"^ "("!  rlogexpr ")"! "loopthrough"! body //LCURLY! ( (stmt |
loopopttail) SEMI!)+ RCURLY!
    | "loopfrom"^ NUMBER "to"! NUMBER "interval"! NUMBER  body//LCURLY! (
(stmt | loopopttail) SEMI!)+ RCURLY!
        //    | "loopthrough"^ LCURLY! ( (stmt | loopopttail) SEMI!)+ RCURLY!
"while"! "("!  rlogexpr ")"!
    ;

loopopttail
    :   "skip_iteration" |"break_loop"
      ;

stmt : //rlogexpr
        outputStmt
    | assignStmt
    | ifStmt
    | loopStmt
    | inputStmt
    | funccall
    | funcdecl
    | objectdef
    | returnStmt
    | loopopttail
        {#stmt = #([STMT,"STMT"], stmt); }
    ;

rlogexpr:
        reqexpr (("AND"^|"OR"^) reqexpr)*;

reqexpr:
        rexpr ((REQ^|RNEQ^) rexpr)*;

rexpr
    :
        ("NOT"^)? expr
    ;

expr
    :
      expr1 ((RGEQ^|RLEQ^|RGT^|RLT^) expr1)*
    ;

expr1 :
        expr2 ( (PLUS^ | MINUS^) expr2 )*
    ;

expr2 :
        expr3 ( (STAR^ | DIV^| PERCENT^|"mod"^) expr3 )*
    ;

expr3 :
        expr4 (POWER^ expr4)*
    ;

expr4:
        PLUS! expr5
        {#expr4 = #([UPLUS,"UPLUS"], expr4); }
```

```
    | MINUS! expr5
        {#expr4 = #([UMINUS,"UMINUS"], expr4); }
    | expr5
    ;

expr5
    :
        lvalue
    | "("! rlogexpr ")"!
    | NUMBER
    | STRING
    | "true"
    |"false"
    | funccall
    | objcall
    | array
    ;

// arrayspec :
//    LSQBRACKET! (ID | NUMBER) RSQBRACKET!
//    ;

// array_row
//         : rlogexpr (COMMA! rlogexpr)*
//             {#array_row = #([ARRAY_ROW,"ARRAY_ROW"], array_row); }
//         ;

//lvalue:
// ID^ (arrayspec)*; //(DOT! ID)?;

lvalue
    :
        ID^ (LSQBRACKET! index RSQBRACKET!)*
    ;
index
    :
        expr1
        {#index = #([VARLIST,"INDEX"],index);}
    ;
array
    :
        LSQBRACKET! expr1 (COMMA! expr1)* RSQBRACKET!
        {#array = #([ARRAY,"ARRAY"], array); }
    ;

objectdef:
        "object"^ ID body
    ;
```

10.1.2 Walker.g

```
/**
 *insert type's descritpion here.
 *Creation Date: (2004-11-08 09:20:10)
 *author: deepti, bhagyashree
 */
{
import java.io.*;
import java.util.*;
}

class PfordTreeWalker extends TreeParser;

options {
 //   buildAST = true;
    importVocab = PfordAntlr;
}

{
    java.util.Hashtable dict = new java.util.Hashtable();
    static PfordDataType null_data=new PfordDataType("<NULL>");
    PfordInterpreter interpret = new PfordInterpreter();

}


//program returns [PfordDataType a]{a =null_data; } : (a=expr)+ ;

expr returns [PfordDataType r]
{

    PfordDataType a, b, c;
    PfordDataType[] p;
    String s = null;
    String[] sa;
    r=null_data;
    Vector v;
}
    :

#(STMT (stmt:. { if ( interpret.canProceed() )  r = expr(#stmt); } )*)

    |#("OR" a=expr right_or:.)
        {
            if ( a instanceof PfordBool )
            r = ( ((PfordBool)a).var ? a : expr(#right_or) );
            else
            r = a.or( expr(#right_or) );
        }

    | #("AND" a=expr right_and:.)
        {
            if ( a instanceof PfordBool )
            r = ( ((PfordBool)a).var ? expr(#right_and) : a );
            else
            r = a.and( expr(#right_and) );
```

```
        }

    |#(UPLUS a=expr)            { r = a; }
    |#(UMINUS a=expr)           { r = a.uminus(); }
    |#("NOT" a=expr)            { r = a.not(); }
    |#(RGEQ a=expr b=expr)    {r=a.gte(b);}
    |#(RLEQ a=expr b=expr)    {r=a.lte(b);}
    |#(RGT a=expr b=expr)     {r=a.gt(b);}
    |#(RLT a=expr b=expr)     {r=a.lt(b);}
    |#(REQ a=expr b=expr)     {r=a.eq(b);}
    |#(RNEQ a=expr b=expr)    {r=a.neq(b);}
    |#(PLUS a=expr b=expr)    {r=a.add(b);}
    |#(MINUS a=expr b=expr)   {r=a.sub(b);}
    |#(STAR a=expr b=expr)    {r=a.times(b);}
    |#(DIV a=expr b=expr)     {r=a.div(b);}
    |#(PERCENT a=expr b=expr) { r = a.percent(b);}
    |#("mod" a=expr b=expr)   {r=a.modulus(b);}
    |#(UPLUSEQ a=expr b=expr) {r=a.add_assign(b);}
    |#(UMINUSEQ a=expr b=expr) {r=a.sub_assign(b);}
    |#(UMULTEQ a=expr b=expr) {r=a.mult_assign(b);}
    |#(UDIVEQ a=expr b=expr)  {r=a.div_assign(b);}
    |#(POWER a=expr b=expr)    {r= a.power(b);}

    |number:NUMBER { r = interpret.getNumber( number.getText()
);}//Integer.parseInt(#NUMBER.getText(), 10); }
    |str:STRING         {r=interpret.getString(str.getText() );}
    |"true"             {r=new      PfordBool(true);}
    |"TRUE"             {r=new      PfordBool(true);}
    |"True"             {r=new      PfordBool(true);}
    |"false"            {r=new      PfordBool(false);}
    |"FALSE"            {r=new      PfordBool(false);}
    |"False"            {r=new      PfordBool(false);}

    | #("output"
            a=expr {
                if((a instanceof PfordString))
                System.out.println(((PfordString)a).var);

                if((a instanceof PfordVariable))
                a.print();

                if((a instanceof PfordInt))
                System.out.println(((PfordInt)a).var);

                if((a instanceof PfordDouble))
                System.out.println(((PfordDouble)a).var);

                if((a instanceof PfordBool))
                System.out.println(((PfordBool)a).var);

            } )

    | #("input" a= expr
            {
                r= interpret.getInput(a);
            } )
```

```
| #(id: ID
        {           r = interpret.getVariable( id.getText() );
        }

        ( p = mexpr                { r = interpret.subMatrix( r,p ); }
        )*
    )

| #(ASSIGN a=expr b = expr)
    {
        r = interpret.assign(a, b);
    }

| #("loopfrom" a=expr b=expr c=predic:expr)
    {
        AST forbody = predic.getNextSibling();
        int initial = 0;
        int finalValue = 0;
        int increment = 0;

        initial = ((PfordInt)a).var;
        finalValue = ((PfordInt)b).var;
        increment = ((PfordInt)c).var;

        for (int i = initial; i<finalValue; i = i + increment)
        {
            r = expr(#forbody);
        }
    }

| #("loop" a=predicate:expr)
    {
        AST loopbody = predicate.getNextSibling();
        int temp = 0;
        if( !(a instanceof PfordInt))
        return a.error("loop: expression should be an int");

        temp = ( (PfordInt)a).var;
        while (temp >0)
        {
            r = expr( #loopbody );
            temp = temp - 1;
        }
    }

| #("while" a=predicateStuff:expr)
    {
        AST whileloopbody = predicateStuff.getNextSibling();
        if( !(a instanceof PfordBool))
        return a.error("whileloop: expression should be a bool");

        while ( ((PfordBool) a).var)
        {
            r = expr( #whileloopbody );
        }
    }
```

```
    |#("if" a=pred:expr {
                AST thenpart = pred.getNextSibling();
                AST elsepart = thenpart.getNextSibling();

                if( !(a instanceof PfordBool))
                return a.error("if: expression should be bool");

                if ( ((PfordBool) a).var) r = expr(#thenpart);
                else if (elsepart != null) r = expr(#elsepart);

            }
        )

    | #("return" ( a=expr        { r = interpret.rvalue( a ); }
            )?
        )                         { interpret.setReturn( null ); }

    | #("break_loop" { interpret.setBreak( s ); })
    | #("skip_iteration"  { interpret.setContinue( s ); })

    | #("func" fname:ID sa=vlist fbody:.)
        { interpret.funcRegister( fname.getText(), sa, #fbody ); }

    | #(FUNCCALL a=expr p=mexpr {
                r = interpret.funcInvoke(this, a,p);
            })

    | #("include" a=expr p=mexpr {
                r = interpret.include(a,p);
            })

    | #("object" oname:ID obody:.)
        { interpret.objRegister( oname.getText(), #obody ); }

    | #(OBJCALL a=expr b=expr {
                r = interpret.objInvoke(this, a);
            })

    | #(ARRAY                     { v = new Vector(); }
            (a=expr                  { v.add( a );}
            )*
        )
        {r = PfordArray.joinVert(interpret.convertExprList( v ) );}


        //  | #(ARRAY_ROW                { v = new Vector(); }
        //              (a=expr                 { v.add( a ); }
        //              )+
        //          )
        //        { r = PfordArray.joinHori( interpret.convertExprList( v )
); }

        //   | #("loopthrough"     predicateDoWhile= a:.)
        //           {
```

```
        //              AST dowhileloopbody =
predicateDoWhile.getNextSibling();
        //              if( !(dowhileloopbody instanceof PfordBool))
        //              return a.error("whileloop: expression should be a
bool");

        //              do
        //              {
        //                  r = expr(a);
        //              }
        //              while ( ((PfordBool) dowhileloopbody).var);
        //          }

    ;

mexpr returns [ PfordDataType[] rv ]
{
    PfordDataType a;
    rv = null;
    Vector v;
}

    : #(VARLIST          { v = new Vector(); }
            ( a=expr       { v.add( a ); }
            )*
        )                   { rv = interpret.convertExprList( v ); }
    | a=expr              { rv = new PfordDataType[1]; rv[0] = a; }
    ;


vlist returns [ String[] sv ]
{
    //PfordDataType a;
    sv = null;
    Vector v;
}

    : #(FUNCVARLIST { v = new Vector(); }
            (s:ID { v.add( s.getText() ); }
            )*
        ) { sv = interpret.convertVarList( v ); }
    ;
```

10.1.3 WalkerJava.g

```
/**
 *insert type's descritpion here.
 *Creation Date: (2004-11-08 09:20:10)
 *author: deepti
 */
{
    import java.io.*;
    import java.util.*;
}

class PfordTreeWalker extends TreeParser;

options {
    //   buildAST = true;
    importVocab = PfordAntlr;
}

{
    java.util.Hashtable dict = new java.util.Hashtable();
    //static PfordDataType null_data=new PfordDataType("<NULL>");
    //PfordInterpreter interpret = new PfordInterpreter();

}


//program returns [PfordDataType a]{a =null_data; } : (a=expr)+ ;

expr returns [float r]
{
    float a,b, c;
    r=0;
}
    :
#(STMT (stmt:. { r = expr(#stmt); } )*)


    |#(UPLUS a=expr)            {System.out.print("+");}
    |#(UMINUS a=expr)           { System.out.print("-");}
    |#("NOT" a=expr)            {System.out.print(" ! ");}
    |#(RGEQ a=expr b=expr)    {System.out.print(" <= ");}
    |#(RLEQ a=expr b=expr)    {System.out.print(" >= ");}
    |#(RGT a=expr b=expr)     {System.out.print(" > ");}
    |#(RLT a=expr b=expr)     {System.out.print(" < ");}
    |#(REQ a=expr b=expr)     {System.out.print(" == ");}
    |#(RNEQ a=expr b=expr)    {System.out.print(" != ");}

        // |#(PERCENT a=expr b=expr) { r = a.percent(b);}
    |#("mod" a=expr b=expr)   {System.out.print(" % ");}
    |#(UPLUSEQ a=expr b=expr) {System.out.print(" += ");}
    |#(UMINUSEQ a=expr b=expr) {System.out.print(" = ");}
    |#(UMULTEQ a=expr b=expr) {System.out.print(" *= ");}
    |#(UDIVEQ a=expr b=expr)  {System.out.print(" /= ");}
        //  |#(POWER a=expr b=expr)    {r= a.power(b);}

    | #(PLUS a=expr b=expr)   { System.out.print(" + "); }
```

```
| #(STAR a=expr b=expr)   { System.out.print(" * "); }
| #(MINUS a=expr b=expr { System.out.print(" - ");})
| #(DIV a=expr b=expr {System.out.print(" / ");})
    //  |str:STRING       {System.out.print(str.getText() );}
|"true"             {System.out.print("true");}
|"TRUE"             {System.out.print("true");}
|"True"             {System.out.print("true");}
|"false"            {System.out.print("false");}
|"FALSE"            {System.out.print("false");}
|"False"            {System.out.print("false");}

| i:NUMBER
    {
        System.out.print((float)Integer.parseInt(i.getText()));
        System.out.print (" ");
    }

| #("output"
        (  s:STRING {
                System.out.println("");
                System.out.print("cout <<"+ '"');
                System.out.print(#s.getText());
                System.out.print('"' +";");
                System.out.println("");
            }
        |
            a=expr {
                System.out.println("");
                System.out.print("cout <<");
                System.out.print(a);
                System.out.print(";");
                System.out.println("");
            } ) )

|#(id: ID {

            System.out.print(id.getText());
        })

| #(ASSIGN ID a=expr
        {
            System.out.print(#ID.getText() + "=");
        } )

|#("if" a=pred:expr {
            AST thenpart = pred.getNextSibling();
            AST elsepart = thenpart.getNextSibling();
            System.out.print("if ( " + a + ") {");
            //if (a != 0)
            r = expr(thenpart);
            System.out.print("}");

            if (elsepart != null){
                System.out.print("else {");
                r = expr(elsepart);
                System.out.print("}");
            }
```

```
                else r = 0;
            }
        )

    | #("break_loop" { System.out.println("break;");})
    | #("skip_iteration"  { System.out.print("continue;");})

    |#("OR" a=expr b=expr)
        {
            System.out.print(" || ");

            //if ( a instanceof PfordBool )
            //r = ( ((PfordBool)a).var ? a : expr(#right_or) );
            //else
            //r = a.or( expr(#right_or) );
        }

    | #("AND" a=expr b=expr)
        {
            System.out.print(" && ");

            // if ( a instanceof PfordBool )
            //             r = ( ((PfordBool)a).var ? expr(#right_and) : a );
            //             else
            //             r = a.and( expr(#right_and) );
        }

    | #("loopfrom" a=expr b=expr c=predic:expr)
        {
            AST forbody = predic.getNextSibling();
            int initial = 0;
            int finalValue = 0;
            int increment = 0;

            //initial = ((PfordInt)a).var;
            //finalValue = ((PfordInt)b).var;
            //increment = ((PfordInt)c).var;

            System.out.println("");
            System.out.print("for (int i = ");
            System.out.println(initial + " ; i < " + finalValue + "; i = i +"
+ increment + "){");
            r = expr(#forbody);
            System.out.println("}");
        }

    | #("loop" a=predicate:expr)
        {
            AST loopbody = predicate.getNextSibling();
            float temp = 0;
            // if( !(a instanceof PfordInt))
            //             return a.error("loop: expression should be an
int");

            temp = a;
            System.out.println("");
```

```
        System.out.print("for (int i = 0");
        System.out.println(" ; i < " + temp + "; i++){");
        r = expr(#loopbody);
        System.out.println("}");
    }

| #("while" a=predicateStuff:expr)
    {
        AST whileloopbody = predicateStuff.getNextSibling();

        System.out.println("");
        System.out.print("while (");
        System.out.println("){");

        r = expr(#whileloopbody);
        System.out.println("}");

    }
;
```

10.1.4 WalkerCpp.g

```
/**
 *insert type's descritpion here.
 *Creation Date: (2004-11-08 09:20:10)
 *author: deepti
 */
{
    import java.io.*;
    import java.util.*;
}

class PfordTreeWalker extends TreeParser;

options {
    //   buildAST = true;
    importVocab = PfordAntlr;
}

{
    java.util.Hashtable dict = new java.util.Hashtable();
    //static PfordDataType null_data=new PfordDataType("<NULL>");
    //PfordInterpreter interpret = new PfordInterpreter();

}


//program returns [PfordDataType a]{a =null_data; } : (a=expr)+ ;

expr returns [float r]
{
    float a,b, c;
    r=0;
}
    :
#(STMT (stmt:. { r = expr(#stmt); } )*)


    |#(UPLUS a=expr)            {System.out.print("+");}
    |#(UMINUS a=expr)           { System.out.print("-");}
    |#("NOT" a=expr)            {System.out.print(" ! ");}
    |#(RGEQ a=expr b=expr)    {System.out.print(" <= ");}
    |#(RLEQ a=expr b=expr)    {System.out.print(" >= ");}
    |#(RGT a=expr b=expr)     {System.out.print(" > ");}
    |#(RLT a=expr b=expr)     {System.out.print(" < ");}
    |#(REQ a=expr b=expr)     {System.out.print(" == ");}
    |#(RNEQ a=expr b=expr)    {System.out.print(" != ");}

        // |#(PERCENT a=expr b=expr) { r = a.percent(b);}
    |#("mod" a=expr b=expr)   {System.out.print(" % ");}
    |#(UPLUSEQ a=expr b=expr) {System.out.print(" += ");}
    |#(UMINUSEQ a=expr b=expr) {System.out.print(" = ");}
    |#(UMULTEQ a=expr b=expr) {System.out.print(" *= ");}
    |#(UDIVEQ a=expr b=expr)  {System.out.print(" /= ");}
        //  |#(POWER a=expr b=expr)    {r= a.power(b);}
```

```
| #(PLUS a=expr b=expr)   { System.out.print(" + "); }
| #(STAR a=expr b=expr)   { System.out.print(" * "); }
| #(MINUS a=expr b=expr { System.out.print(" - ");})
| #(DIV a=expr b=expr {System.out.print(" / ");})
    //  |str:STRING       {System.out.print(str.getText() );}
|"true"             {System.out.print("true");}
|"TRUE"             {System.out.print("true");}
|"True"             {System.out.print("true");}
|"false"            {System.out.print("false");}
|"FALSE"            {System.out.print("false");}
|"False"            {System.out.print("false");}

| i:NUMBER
    {
        System.out.print((float)Integer.parseInt(i.getText()));
        System.out.print (" ");
    }

| #("output"
        (  s:STRING {
                System.out.println("");
                System.out.print("System.out.println("+ '"');
                System.out.print(#s.getText());
                System.out.print('"' +");");
                System.out.println("");
            }
        |
            a=expr {
                System.out.println("");
                System.out.print("System.out.println(");
                System.out.print(a);
                System.out.print(");");
                System.out.println("");
            } ) )

|#(id: ID {

            System.out.print(id.getText());
        })

| #(ASSIGN ID a=expr
        {
            System.out.print(#ID.getText() + "=");
        } )

|#("if" a=pred:expr {
            AST thenpart = pred.getNextSibling();
            AST elsepart = thenpart.getNextSibling();
            System.out.print("if ( " + a + ") {");
            //if (a != 0)
            r = expr(thenpart);
            System.out.print("}");

            if (elsepart != null){
                System.out.print("else {");
                r = expr(elsepart);
                System.out.print("}");
```

```
                }

                else r = 0;
            }
        )

    | #("break_loop" { System.out.println("break;");})
    | #("skip_iteration"  { System.out.print("continue;");})

    |#("OR" a=expr b=expr)
        {
            System.out.print(" || ");

            //if ( a instanceof PfordBool )
            //r = ( ((PfordBool)a).var ? a : expr(#right_or) );
            //else
            //r = a.or( expr(#right_or) );
        }

    | #("AND" a=expr b=expr)
        {
            System.out.print(" && ");

            // if ( a instanceof PfordBool )
            //             r = ( ((PfordBool)a).var ? expr(#right_and) : a );
            //             else
            //             r = a.and( expr(#right_and) );
        }

    | #("loopfrom" a=expr b=expr c=predic:expr)
        {
            AST forbody = predic.getNextSibling();
            int initial = 0;
            int finalValue = 0;
            int increment = 0;

            //initial = ((PfordInt)a).var;
            //finalValue = ((PfordInt)b).var;
            //increment = ((PfordInt)c).var;

            System.out.println("");
            System.out.print("for (int i = ");
            System.out.println(initial + " ; i < " + finalValue + "; i = i +"
+ increment + "){");
            r = expr(#forbody);
            System.out.println("}");
        }

    | #("loop" a=predicate:expr)
        {
            AST loopbody = predicate.getNextSibling();
            float temp = 0;
            // if( !(a instanceof PfordInt))
            //             return a.error("loop: expression should be an
int");

            temp = a;
```

```
        System.out.println("");
        System.out.print("for (int i = 0");
        System.out.println(" ; i < " + temp + "; i++){");
        r = expr(#loopbody);
        System.out.println("}");
    }

| #("while" a=predicateStuff:expr)
    {
        AST whileloopbody = predicateStuff.getNextSibling();

        System.out.println("");
        System.out.print("while (");
        System.out.println("){");

        r = expr(#whileloopbody);
        System.out.println("}");

    }
;
```

10.1.5 PfordTreeWalkerCpp.java

```
/**
 * Insert type's description here
 * Creation Date :(2004-11-11 15:36:15)
 *author: Deepti Jindal
 */

// $ANTLR 2.7.3: "Parser.g" -> "PfordTreeWalkerCpp.java"$

import antlr.TreeParser;
import antlr.Token;
import antlr.collections.AST;
import antlr.RecognitionException;
import antlr.ANTLRException;
import antlr.NoViableAltException;
import antlr.MismatchedTokenException;
import antlr.SemanticException;
import antlr.collections.impl.BitSet;
import antlr.ASTPair;
import antlr.collections.impl.ASTArray;

import java.io.*;
import java.util.*;


public class PfordTreeWalkerCpp extends antlr.TreeParser       implements
PfordTreeWalkerTokenTypes {

    java.util.Hashtable dict = new java.util.Hashtable();
    //static PfordDataType null_data=new PfordDataType("<NULL>");
    //PfordInterpreter interpret = new PfordInterpreter();

    public PfordTreeWalkerCpp() {
        tokenNames = _tokenNames;
    }

    public final float  expr(PrintStream ps, AST _t) throws
RecognitionException {
        float r;

        AST expr_AST_in = (_t == ASTNULL) ? null : (AST)_t;
        AST stmt = null;
        AST i = null;
        AST s = null;
        AST id = null;
        AST pred = null;
        AST predic = null;
        AST predicate = null;
        AST predicateStuff = null;

        float a,b, c;
        r=0;


        try {      // for error handling
```

```
            if (_t==null) _t=ASTNULL;
            switch ( _t.getType()) {
                case STMT: {
                    AST __t129 = _t;
                    AST tmp89_AST_in = (AST)_t;
                    match(_t,STMT);
                    _t = _t.getFirstChild(); {
                        _loop131:
                            do {
                                if (_t==null) _t=ASTNULL;
                                if (((_t.getType() >= SEMI && _t.getType() <=
LITERAL_False))) {
                                    stmt = (AST)_t;
                                    if ( _t==null ) throw new
MismatchedTokenException();
                                    _t = _t.getNextSibling();
                                    r = expr(ps,stmt);
                                }
                                else {
                                    break _loop131;
                                }

                            } while (true);
                    }
                    _t = __t129;
                    _t = _t.getNextSibling();
                    break;
                }
                case UPLUS: {
                    AST __t132 = _t;
                    AST tmp90_AST_in = (AST)_t;
                    match(_t,UPLUS);
                    _t = _t.getFirstChild();
                    a=expr(ps,_t);
                    _t = _retTree;
                    _t = __t132;
                    _t = _t.getNextSibling();
                    ps.print("+");
                    break;
                }
                case UMINUS: {
                    AST __t133 = _t;
                    AST tmp91_AST_in = (AST)_t;
                    match(_t,UMINUS);
                    _t = _t.getFirstChild();
                    a=expr(ps,_t);
                    _t = _retTree;
                    _t = __t133;
                    _t = _t.getNextSibling();
                    ps.print("-");
                    break;
                }
                case LITERAL_NOT: {
                    AST __t134 = _t;
                    AST tmp92_AST_in = (AST)_t;
                    match(_t,LITERAL_NOT);
                    _t = _t.getFirstChild();
```

```
    a=expr(ps,_t);
    _t = _retTree;
    _t = __t134;
    _t = _t.getNextSibling();
    ps.print(" ! ");
    break;
}
case RGEQ: {
    AST __t135 = _t;
    AST tmp93_AST_in = (AST)_t;
    match(_t,RGEQ);
    _t = _t.getFirstChild();
    a=expr(ps,_t);
    _t = _retTree;
    b=expr(ps,_t);
    _t = _retTree;
    _t = __t135;
    _t = _t.getNextSibling();
    ps.print(" <= ");
    break;
}
case RLEQ: {
    AST __t136 = _t;
    AST tmp94_AST_in = (AST)_t;
    match(_t,RLEQ);
    _t = _t.getFirstChild();
    a=expr(ps,_t);
    _t = _retTree;
    b=expr(ps,_t);
    _t = _retTree;
    _t = __t136;
    _t = _t.getNextSibling();
    ps.print(" >= ");
    break;
}
case RGT: {
    AST __t137 = _t;
    AST tmp95_AST_in = (AST)_t;
    match(_t,RGT);
    _t = _t.getFirstChild();
    a=expr(ps,_t);
    _t = _retTree;
    b=expr(ps,_t);
    _t = _retTree;
    _t = __t137;
    _t = _t.getNextSibling();
    ps.print(" > ");
    break;
}
case RLT: {
    AST __t138 = _t;
    AST tmp96_AST_in = (AST)_t;
    match(_t,RLT);
    _t = _t.getFirstChild();
    a=expr(ps,_t);
    _t = _retTree;
    b=expr(ps,_t);
```

```
    _t = _retTree;
    _t = __t138;
    _t = _t.getNextSibling();
    ps.print(" < ");
    break;
}
case REQ: {
    AST __t139 = _t;
    AST tmp97_AST_in = (AST)_t;
    match(_t,REQ);
    _t = _t.getFirstChild();
    a=expr(ps,_t);
    _t = _retTree;
    b=expr(ps,_t);
    _t = _retTree;
    _t = __t139;
    _t = _t.getNextSibling();
    ps.print(" == ");
    break;
}
case RNEQ: {
    AST __t140 = _t;
    AST tmp98_AST_in = (AST)_t;
    match(_t,RNEQ);
    _t = _t.getFirstChild();
    a=expr(ps,_t);
    _t = _retTree;
    b=expr(ps,_t);
    _t = _retTree;
    _t = __t140;
    _t = _t.getNextSibling();
    ps.print(" != ");
    break;
}
case LITERAL_mod: {
    AST __t141 = _t;
    AST tmp99_AST_in = (AST)_t;
    match(_t,LITERAL_mod);
    _t = _t.getFirstChild();
    a=expr(ps,_t);
    _t = _retTree;
    b=expr(ps,_t);
    _t = _retTree;
    _t = __t141;
    _t = _t.getNextSibling();
    ps.print(" % ");
    break;
}
case UPLUSEQ: {
    AST __t142 = _t;
    AST tmp100_AST_in = (AST)_t;
    match(_t,UPLUSEQ);
    _t = _t.getFirstChild();
    a=expr(ps,_t);
    _t = _retTree;
    b=expr(ps,_t);
    _t = _retTree;
```

```
    _t = __t142;
    _t = _t.getNextSibling();
    ps.print(" += ");
    break;
}
case UMINUSEQ: {
    AST __t143 = _t;
    AST tmp101_AST_in = (AST)_t;
    match(_t,UMINUSEQ);
    _t = _t.getFirstChild();
    a=expr(ps,_t);
    _t = _retTree;
    b=expr(ps,_t);
    _t = _retTree;
    _t = __t143;
    _t = _t.getNextSibling();
    ps.print(" = ");
    break;
}
case UMULTEQ: {
    AST __t144 = _t;
    AST tmp102_AST_in = (AST)_t;
    match(_t,UMULTEQ);
    _t = _t.getFirstChild();
    a=expr(ps,_t);
    _t = _retTree;
    b=expr(ps,_t);
    _t = _retTree;
    _t = __t144;
    _t = _t.getNextSibling();
    ps.print(" *= ");
    break;
}
case UDIVEQ: {
    AST __t145 = _t;
    AST tmp103_AST_in = (AST)_t;
    match(_t,UDIVEQ);
    _t = _t.getFirstChild();
    a=expr(ps,_t);
    _t = _retTree;
    b=expr(ps,_t);
    _t = _retTree;
    _t = __t145;
    _t = _t.getNextSibling();
    ps.print(" /= ");
    break;
}
case PLUS: {
    AST __t146 = _t;
    AST tmp104_AST_in = (AST)_t;
    match(_t,PLUS);
    _t = _t.getFirstChild();
    a=expr(ps,_t);
    _t = _retTree;
    b=expr(ps,_t);
    _t = _retTree;
    _t = __t146;
```

```
    _t = _t.getNextSibling();
    ps.print(" + ");
    break;
}
case STAR: {
    AST __t147 = _t;
    AST tmp105_AST_in = (AST)_t;
    match(_t,STAR);
    _t = _t.getFirstChild();
    a=expr(ps,_t);
    _t = _retTree;
    b=expr(ps,_t);
    _t = _retTree;
    _t = __t147;
    _t = _t.getNextSibling();
    ps.print(" * ");
    break;
}
case MINUS: {
    AST __t148 = _t;
    AST tmp106_AST_in = (AST)_t;
    match(_t,MINUS);
    _t = _t.getFirstChild();
    a=expr(ps,_t);
    _t = _retTree;
    b=expr(ps,_t);
    _t = _retTree;
    ps.print(" - ");
    _t = __t148;
    _t = _t.getNextSibling();
    break;
}
case DIV: {
    AST __t149 = _t;
    AST tmp107_AST_in = (AST)_t;
    match(_t,DIV);
    _t = _t.getFirstChild();
    a=expr(ps,_t);
    _t = _retTree;
    b=expr(ps,_t);
    _t = _retTree;
    ps.print(" / ");
    _t = __t149;
    _t = _t.getNextSibling();
    break;
}
case LITERAL_true: {
    AST tmp108_AST_in = (AST)_t;
    match(_t,LITERAL_true);
    _t = _t.getNextSibling();
    ps.print("true");
    break;
}
case LITERAL_TRUE: {
    AST tmp109_AST_in = (AST)_t;
    match(_t,LITERAL_TRUE);
    _t = _t.getNextSibling();
```

```
    ps.print("true");
    break;
}
case LITERAL_True: {
    AST tmp110_AST_in = (AST)_t;
    match(_t,LITERAL_True);
    _t = _t.getNextSibling();
    ps.print("true");
    break;
}
case LITERAL_false: {
    AST tmp111_AST_in = (AST)_t;
    match(_t,LITERAL_false);
    _t = _t.getNextSibling();
    ps.print("false");
    break;
}
case LITERAL_FALSE: {
    AST tmp112_AST_in = (AST)_t;
    match(_t,LITERAL_FALSE);
    _t = _t.getNextSibling();
    ps.print("false");
    break;
}
case LITERAL_False: {
    AST tmp113_AST_in = (AST)_t;
    match(_t,LITERAL_False);
    _t = _t.getNextSibling();
    ps.print("false");
    break;
}
case NUMBER: {
    i = (AST)_t;
    match(_t,NUMBER);
    _t = _t.getNextSibling();

    ps.print((float)Integer.parseInt(i.getText()));
    ps.print(" ");

    break;
}
case LITERAL_output: {
    AST __t150 = _t;
    AST tmp114_AST_in = (AST)_t;
    match(_t,LITERAL_output);
    _t = _t.getFirstChild(); {
        if (_t==null) _t=ASTNULL;
        switch ( _t.getType()) {
            case STRING: {
                s = (AST)_t;
                match(_t,STRING);
                _t = _t.getNextSibling();

                ps.println("");
                ps.print("cout <<"+ '"');
                ps.print(s.getText());
                ps.print('"' +";");
```

```
                ps.println("");

                break;
            }
            case PLUS:
            case STAR:
            case MINUS:
            case DIV:
            case ASSIGN:
            case UPLUSEQ:
            case UMINUSEQ:
            case UMULTEQ:
            case UDIVEQ:
            case REQ:
            case RGEQ:
            case RLEQ:
            case RNEQ:
            case RGT:
            case RLT:
            case ID:
            case NUMBER:
            case STMT:
            case UPLUS:
            case UMINUS:
            case LITERAL_if:
            case LITERAL_output:
            case LITERAL_loop:
            case LITERAL_while:
            case LITERAL_loopfrom:
            case LITERAL_skip_iteration:
            case LITERAL_break_loop:
            case LITERAL_AND:
            case LITERAL_OR:
            case LITERAL_NOT:
            case LITERAL_mod:
            case LITERAL_true:
            case LITERAL_false:
            case LITERAL_TRUE:
            case LITERAL_True:
            case LITERAL_FALSE:
            case LITERAL_False: {
                a=expr(ps,_t);
                _t = _retTree;

                ps.println("");
                ps.print("cout <<");
                ps.print(a);
                ps.print(";");
                ps.println("");

                break;
            }
            default: {
                throw new NoViableAltException(_t);
            }
        }
    }
```

```
    _t = __t150;
    _t = _t.getNextSibling();
    break;
}
case ID: {
    AST __t152 = _t;
    id = _t==ASTNULL ? null :(AST)_t;
    match(_t,ID);
    _t = _t.getFirstChild();


    ps.print(id.getText());

    _t = __t152;
    _t = _t.getNextSibling();
    break;
}
case ASSIGN: {
    AST __t153 = _t;
    AST tmp115_AST_in = (AST)_t;
    match(_t,ASSIGN);
    _t = _t.getFirstChild();
    AST tmp116_AST_in = (AST)_t;
    match(_t,ID);
    _t = _t.getNextSibling();
    a=expr(ps,_t);
    _t = _retTree;

    ps.print(tmp116_AST_in.getText() + "=");

    _t = __t153;
    _t = _t.getNextSibling();
    break;
}
case LITERAL_if: {
    AST __t154 = _t;
    AST tmp117_AST_in = (AST)_t;
    match(_t,LITERAL_if);
    _t = _t.getFirstChild();
    pred = _t==ASTNULL ? null : (AST)_t;
    a=expr(ps,_t);
    _t = _retTree;

    AST thenpart = pred.getNextSibling();
    AST elsepart = thenpart.getNextSibling();
    ps.print("if ( " + a + ") {");
    //if (a != 0)
    r = expr(ps,thenpart);
    ps.print("}");

    if (elsepart != null){
        ps.print("else {");
        r = expr(ps,elsepart);
        ps.print("}");
    }

    else r = 0;
```

```
        _t = __t154;
        _t = _t.getNextSibling();
        break;
    }
    case LITERAL_break_loop: {
        AST __t155 = _t;
        AST tmp118_AST_in = (AST)_t;
        match(_t,LITERAL_break_loop);
        _t = _t.getFirstChild();
        ps.println("break;");
        _t = __t155;
        _t = _t.getNextSibling();
        break;
    }
    case LITERAL_skip_iteration: {
        AST __t156 = _t;
        AST tmp119_AST_in = (AST)_t;
        match(_t,LITERAL_skip_iteration);
        _t = _t.getFirstChild();
        ps.print("continue;");
        _t = __t156;
        _t = _t.getNextSibling();
        break;
    }
    case LITERAL_OR: {
        AST __t157 = _t;
        AST tmp120_AST_in = (AST)_t;
        match(_t,LITERAL_OR);
        _t = _t.getFirstChild();
        a=expr(ps,_t);
        _t = _retTree;
        b=expr(ps,_t);
        _t = _retTree;
        _t = __t157;
        _t = _t.getNextSibling();

        ps.print(" || ");

        //if ( a instanceof PfordBool )
        //r = ( ((PfordBool)a).var ? a : expr(ps,#right_or) );
        //else
        //r = a.or( expr(ps,#right_or) );

        break;
    }
    case LITERAL_AND: {
        AST __t158 = _t;
        AST tmp121_AST_in = (AST)_t;
        match(_t,LITERAL_AND);
        _t = _t.getFirstChild();
        a=expr(ps,_t);
        _t = _retTree;
        b=expr(ps,_t);
        _t = _retTree;
        _t = __t158;
        _t = _t.getNextSibling();
```

```
                    ps.print(" && ");

                    // if ( a instanceof PfordBool )
                    //             r = ( ((PfordBool)a).var ?
expr(ps,#right_and) : a );
                    //             else
                    //             r = a.and( expr(ps,#right_and) );

                    break;
                }
                case LITERAL_loopfrom: {
                    AST __t159 = _t;
                    AST tmp122_AST_in = (AST)_t;
                    match(_t,LITERAL_loopfrom);
                    _t = _t.getFirstChild();
                    a=expr(ps,_t);
                    _t = _retTree;
                    b=expr(ps,_t);
                    _t = _retTree;
                    predic = _t==ASTNULL ? null : (AST)_t;
                    c=expr(ps,_t);
                    _t = _retTree;
                    _t = __t159;
                    _t = _t.getNextSibling();

                    AST forbody = predic.getNextSibling();
                    int initial = 0;
                    int finalValue = 0;
                    int increment = 0;

                    //initial = ((PfordInt)a).var;
                    //finalValue = ((PfordInt)b).var;
                    //increment = ((PfordInt)c).var;

                    ps.println("");
                    ps.print("for (int i = ");
                    ps.println(initial + " ; i < " + finalValue + "; i = i +"
+ increment + "){");
                    r = expr(ps,forbody);
                    ps.println("}");

                    break;
                }
                case LITERAL_loop: {
                    AST __t160 = _t;
                    AST tmp123_AST_in = (AST)_t;
                    match(_t,LITERAL_loop);
                    _t = _t.getFirstChild();
                    predicate = _t==ASTNULL ? null : (AST)_t;
                    a=expr(ps,_t);
                    _t = _retTree;
                    _t = __t160;
                    _t = _t.getNextSibling();

                    AST loopbody = predicate.getNextSibling();
                    float temp = 0;
```

```
                    // if( !(a instanceof PfordInt))
                    //             return a.error("loop: expression should be
an int");

                    temp = a;
                    ps.println("");
                    ps.print("for (int i = 0");
                    ps.println(" ; i < " + temp + "; i++){");
                    r = expr(ps,loopbody);
                    ps.println("}");

                    break;
                }
                case LITERAL_while: {
                    AST __t161 = _t;
                    AST tmp124_AST_in = (AST)_t;
                    match(_t,LITERAL_while);
                    _t = _t.getFirstChild();
                    predicateStuff = _t==ASTNULL ? null : (AST)_t;
                    a=expr(ps,_t);
                    _t = _retTree;
                    _t = __t161;
                    _t = _t.getNextSibling();

                    AST whileloopbody = predicateStuff.getNextSibling();

                    ps.println("");
                    ps.print("while (");
                    ps.println("){");

                    r = expr(ps,whileloopbody);
                    ps.println("}");


                    break;
                }
                default: {
                    throw new NoViableAltException(_t);
                }
            }
        }
        catch (RecognitionException ex) {
            reportError(ex);
            if (_t!=null) {_t = _t.getNextSibling();}
        }
        _retTree = _t;
        return r;
    }


    public static final String[] _tokenNames = {
        "<0>",
        "EOF",
        "<2>",
        "NULL_TREE_LOOKAHEAD",
        "SEMI",
        "PLUS",
```

```
"POWER",
"STAR",
"MINUS",
"DIV",
"ASSIGN",
"RSQBRACKET",
"LSQBRACKET",
"LCURLY",
"RCURLY",
"PERCENT",
"COMMA",
"DOT",
"UPLUSEQ",
"UMINUSEQ",
"UMULTEQ",
"UDIVEQ",
"REQ",
"RGEQ",
"RLEQ",
"RNEQ",
"RGT",
"RLT",
"INCR",
"DECR",
"PARENS",
"LETTER",
"DIGIT",
"ID",
"NUMBER",
"STRING",
"WS",
"COMMENT",
"STMT",
"UPLUS",
"UMINUS",
"FUNCCALL",
"VARLIST",
"FUNCVARLIST",
"ARRAY_ROW",
"ARRAY",
"OBJCALL",
"\"startprogram\"",
"\"endprogram\"",
"\"if\"",
"\"(\"",
"\")\"",
"\"then\"",
"\"elsedo\"",
"\"output\"",
"\"input\"",
"\"func\"",
"\"return\"",
"\"loop\"",
"\"runs\"",
"\"while\"",
"\"loopthrough\"",
"\"loopfrom\"",
```

```
        "\"to\"",
        "\"interval\"",
        "\"skip_iteration\"",
        "\"break_loop\"",
        "\"AND\"",
        "\"OR\"",
        "GT",
        "LT",
        "\"NOT\"",
        "\"mod\"",
        "\"true\"",
        "\"false\"",
        "\"object\"",
        "\"TRUE\"",
        "\"True\"",
        "\"FALSE\"",
        "\"False\""
    };

}
```

10.1.6 PfordTreeWalkerJava.java

```
/**
 * Insert type's description here
 * Creation Date :(2004-11-11 15:36:15)
 *author: Deepti Jindal
 */

// $ANTLR 2.7.3: "Parser.g" -> "PfordTreeWalkerJava.java"$

import antlr.TreeParser;
import antlr.Token;
import antlr.collections.AST;
import antlr.RecognitionException;
import antlr.ANTLRException;
import antlr.NoViableAltException;
import antlr.MismatchedTokenException;
import antlr.SemanticException;
import antlr.collections.impl.BitSet;
import antlr.ASTPair;
import antlr.collections.impl.ASTArray;

import java.io.*;
import java.util.*;


public class PfordTreeWalkerJava extends antlr.TreeParser       implements
PfordTreeWalkerTokenTypes {

    java.util.Hashtable dict = new java.util.Hashtable();
    //static PfordDataType null_data=new PfordDataType("<NULL>");
    //PfordInterpreter interpret = new PfordInterpreter();

    public PfordTreeWalkerJava() {
        tokenNames = _tokenNames;
    }

    public final float  expr(PrintStream ps, AST _t) throws
RecognitionException {
        float r;

        AST expr_AST_in = (_t == ASTNULL) ? null : (AST)_t;
        AST stmt = null;
        AST i = null;
        AST s = null;
        AST id = null;
        AST pred = null;
        AST predic = null;
        AST predicate = null;
        AST predicateStuff = null;

        float a,b, c;
        r=0;


        try {      // for error handling
```

```
            if (_t==null) _t=ASTNULL;
            switch ( _t.getType()) {
                case STMT: {
                    AST __t129 = _t;
                    AST tmp89_AST_in = (AST)_t;
                    match(_t,STMT);
                    _t = _t.getFirstChild(); {
                        _loop131:
                            do {
                                if (_t==null) _t=ASTNULL;
                                if (((_t.getType() >= SEMI && _t.getType() <=
LITERAL_False))) {
                                    stmt = (AST)_t;
                                    if ( _t==null ) throw new
MismatchedTokenException();
                                    _t = _t.getNextSibling();
                                    r = expr(ps,stmt);
                                }
                                else {
                                    break _loop131;
                                }

                            } while (true);
                    }
                    _t = __t129;
                    _t = _t.getNextSibling();
                    break;
                }
                case UPLUS: {
                    AST __t132 = _t;
                    AST tmp90_AST_in = (AST)_t;
                    match(_t,UPLUS);
                    _t = _t.getFirstChild();
                    a=expr(ps,_t);
                    _t = _retTree;
                    _t = __t132;
                    _t = _t.getNextSibling();
                    ps.print("+");
                    break;
                }
                case UMINUS: {
                    AST __t133 = _t;
                    AST tmp91_AST_in = (AST)_t;
                    match(_t,UMINUS);
                    _t = _t.getFirstChild();
                    a=expr(ps,_t);
                    _t = _retTree;
                    _t = __t133;
                    _t = _t.getNextSibling();
                    ps.print("-");
                    break;
                }
                case LITERAL_NOT: {
                    AST __t134 = _t;
                    AST tmp92_AST_in = (AST)_t;
                    match(_t,LITERAL_NOT);
                    _t = _t.getFirstChild();
```

```
    a=expr(ps,_t);
    _t = _retTree;
    _t = __t134;
    _t = _t.getNextSibling();
    ps.print(" ! ");
    break;
}
case RGEQ: {
    AST __t135 = _t;
    AST tmp93_AST_in = (AST)_t;
    match(_t,RGEQ);
    _t = _t.getFirstChild();
    a=expr(ps,_t);
    _t = _retTree;
    b=expr(ps,_t);
    _t = _retTree;
    _t = __t135;
    _t = _t.getNextSibling();
    ps.print(" <= ");
    break;
}
case RLEQ: {
    AST __t136 = _t;
    AST tmp94_AST_in = (AST)_t;
    match(_t,RLEQ);
    _t = _t.getFirstChild();
    a=expr(ps,_t);
    _t = _retTree;
    b=expr(ps,_t);
    _t = _retTree;
    _t = __t136;
    _t = _t.getNextSibling();
    ps.print(" >= ");
    break;
}
case RGT: {
    AST __t137 = _t;
    AST tmp95_AST_in = (AST)_t;
    match(_t,RGT);
    _t = _t.getFirstChild();
    a=expr(ps,_t);
    _t = _retTree;
    b=expr(ps,_t);
    _t = _retTree;
    _t = __t137;
    _t = _t.getNextSibling();
    ps.print(" > ");
    break;
}
case RLT: {
    AST __t138 = _t;
    AST tmp96_AST_in = (AST)_t;
    match(_t,RLT);
    _t = _t.getFirstChild();
    a=expr(ps,_t);
    _t = _retTree;
    b=expr(ps,_t);
```

```
        _t = _retTree;
        _t = __t138;
        _t = _t.getNextSibling();
        ps.print(" < ");
        break;
    }
    case REQ: {
        AST __t139 = _t;
        AST tmp97_AST_in = (AST)_t;
        match(_t,REQ);
        _t = _t.getFirstChild();
        a=expr(ps,_t);
        _t = _retTree;
        b=expr(ps,_t);
        _t = _retTree;
        _t = __t139;
        _t = _t.getNextSibling();
        ps.print(" == ");
        break;
    }
    case RNEQ: {
        AST __t140 = _t;
        AST tmp98_AST_in = (AST)_t;
        match(_t,RNEQ);
        _t = _t.getFirstChild();
        a=expr(ps,_t);
        _t = _retTree;
        b=expr(ps,_t);
        _t = _retTree;
        _t = __t140;
        _t = _t.getNextSibling();
        ps.print(" != ");
        break;
    }
    case LITERAL_mod: {
        AST __t141 = _t;
        AST tmp99_AST_in = (AST)_t;
        match(_t,LITERAL_mod);
        _t = _t.getFirstChild();
        a=expr(ps,_t);
        _t = _retTree;
        b=expr(ps,_t);
        _t = _retTree;
        _t = __t141;
        _t = _t.getNextSibling();
        ps.print(" % ");
        break;
    }
    case UPLUSEQ: {
        AST __t142 = _t;
        AST tmp100_AST_in = (AST)_t;
        match(_t,UPLUSEQ);
        _t = _t.getFirstChild();
        a=expr(ps,_t);
        _t = _retTree;
        b=expr(ps,_t);
        _t = _retTree;
```

```
    _t = __t142;
    _t = _t.getNextSibling();
    ps.print(" += ");
    break;
}
case UMINUSEQ: {
    AST __t143 = _t;
    AST tmp101_AST_in = (AST)_t;
    match(_t,UMINUSEQ);
    _t = _t.getFirstChild();
    a=expr(ps,_t);
    _t = _retTree;
    b=expr(ps,_t);
    _t = _retTree;
    _t = __t143;
    _t = _t.getNextSibling();
    ps.print(" = ");
    break;
}
case UMULTEQ: {
    AST __t144 = _t;
    AST tmp102_AST_in = (AST)_t;
    match(_t,UMULTEQ);
    _t = _t.getFirstChild();
    a=expr(ps,_t);
    _t = _retTree;
    b=expr(ps,_t);
    _t = _retTree;
    _t = __t144;
    _t = _t.getNextSibling();
    ps.print(" *= ");
    break;
}
case UDIVEQ: {
    AST __t145 = _t;
    AST tmp103_AST_in = (AST)_t;
    match(_t,UDIVEQ);
    _t = _t.getFirstChild();
    a=expr(ps,_t);
    _t = _retTree;
    b=expr(ps,_t);
    _t = _retTree;
    _t = __t145;
    _t = _t.getNextSibling();
    ps.print(" /= ");
    break;
}
case PLUS: {
    AST __t146 = _t;
    AST tmp104_AST_in = (AST)_t;
    match(_t,PLUS);
    _t = _t.getFirstChild();
    a=expr(ps,_t);
    _t = _retTree;
    b=expr(ps,_t);
    _t = _retTree;
    _t = __t146;
```

```
    _t = _t.getNextSibling();
    ps.print(" + ");
    break;
}
case STAR: {
    AST __t147 = _t;
    AST tmp105_AST_in = (AST)_t;
    match(_t,STAR);
    _t = _t.getFirstChild();
    a=expr(ps,_t);
    _t = _retTree;
    b=expr(ps,_t);
    _t = _retTree;
    _t = __t147;
    _t = _t.getNextSibling();
    ps.print(" * ");
    break;
}
case MINUS: {
    AST __t148 = _t;
    AST tmp106_AST_in = (AST)_t;
    match(_t,MINUS);
    _t = _t.getFirstChild();
    a=expr(ps,_t);
    _t = _retTree;
    b=expr(ps,_t);
    _t = _retTree;
    ps.print(" - ");
    _t = __t148;
    _t = _t.getNextSibling();
    break;
}
case DIV: {
    AST __t149 = _t;
    AST tmp107_AST_in = (AST)_t;
    match(_t,DIV);
    _t = _t.getFirstChild();
    a=expr(ps,_t);
    _t = _retTree;
    b=expr(ps,_t);
    _t = _retTree;
    ps.print(" / ");
    _t = __t149;
    _t = _t.getNextSibling();
    break;
}
case LITERAL_true: {
    AST tmp108_AST_in = (AST)_t;
    match(_t,LITERAL_true);
    _t = _t.getNextSibling();
    ps.print("true");
    break;
}
case LITERAL_TRUE: {
    AST tmp109_AST_in = (AST)_t;
    match(_t,LITERAL_TRUE);
    _t = _t.getNextSibling();
```

```
    ps.print("true");
    break;
}
case LITERAL_True: {
    AST tmp110_AST_in = (AST)_t;
    match(_t,LITERAL_True);
    _t = _t.getNextSibling();
    ps.print("true");
    break;
}
case LITERAL_false: {
    AST tmp111_AST_in = (AST)_t;
    match(_t,LITERAL_false);
    _t = _t.getNextSibling();
    ps.print("false");
    break;
}
case LITERAL_FALSE: {
    AST tmp112_AST_in = (AST)_t;
    match(_t,LITERAL_FALSE);
    _t = _t.getNextSibling();
    ps.print("false");
    break;
}
case LITERAL_False: {
    AST tmp113_AST_in = (AST)_t;
    match(_t,LITERAL_False);
    _t = _t.getNextSibling();
    ps.print("false");
    break;
}
case NUMBER: {
    i = (AST)_t;
    match(_t,NUMBER);
    _t = _t.getNextSibling();

    ps.print((float)Integer.parseInt(i.getText()));
    ps.print(" ");

    break;
}
case LITERAL_output: {
    AST __t150 = _t;
    AST tmp114_AST_in = (AST)_t;
    match(_t,LITERAL_output);
    _t = _t.getFirstChild(); {
        if (_t==null) _t=ASTNULL;
        switch ( _t.getType()) {
            case STRING: {
                s = (AST)_t;
                match(_t,STRING);
                _t = _t.getNextSibling();

                ps.println("");
                ps.print("System.out.print("+ '"');
                ps.print(s.getText());
                ps.print('"' +");");
```

```
                ps.println("");

                break;
            }
            case PLUS:
            case STAR:
            case MINUS:
            case DIV:
            case ASSIGN:
            case UPLUSEQ:
            case UMINUSEQ:
            case UMULTEQ:
            case UDIVEQ:
            case REQ:
            case RGEQ:
            case RLEQ:
            case RNEQ:
            case RGT:
            case RLT:
            case ID:
            case NUMBER:
            case STMT:
            case UPLUS:
            case UMINUS:
            case LITERAL_if:
            case LITERAL_output:
            case LITERAL_loop:
            case LITERAL_while:
            case LITERAL_loopfrom:
            case LITERAL_skip_iteration:
            case LITERAL_break_loop:
            case LITERAL_AND:
            case LITERAL_OR:
            case LITERAL_NOT:
            case LITERAL_mod:
            case LITERAL_true:
            case LITERAL_false:
            case LITERAL_TRUE:
            case LITERAL_True:
            case LITERAL_FALSE:
            case LITERAL_False: {
                a=expr(ps,_t);
                _t = _retTree;

                ps.println("");
                ps.print("System.out.print(");
                ps.print(a);
                ps.print(");");
                ps.println("");

                break;
            }
            default: {
                throw new NoViableAltException(_t);
            }
        }
    }
```

```
    _t = __t150;
    _t = _t.getNextSibling();
    break;
}
case ID: {
    AST __t152 = _t;
    id = _t==ASTNULL ? null :(AST)_t;
    match(_t,ID);
    _t = _t.getFirstChild();


    ps.print(id.getText());

    _t = __t152;
    _t = _t.getNextSibling();
    break;
}
case ASSIGN: {
    AST __t153 = _t;
    AST tmp115_AST_in = (AST)_t;
    match(_t,ASSIGN);
    _t = _t.getFirstChild();
    AST tmp116_AST_in = (AST)_t;
    match(_t,ID);
    _t = _t.getNextSibling();
    a=expr(ps,_t);
    _t = _retTree;

    ps.print(tmp116_AST_in.getText() + "=");

    _t = __t153;
    _t = _t.getNextSibling();
    break;
}
case LITERAL_if: {
    AST __t154 = _t;
    AST tmp117_AST_in = (AST)_t;
    match(_t,LITERAL_if);
    _t = _t.getFirstChild();
    pred = _t==ASTNULL ? null : (AST)_t;
    a=expr(ps,_t);
    _t = _retTree;

    AST thenpart = pred.getNextSibling();
    AST elsepart = thenpart.getNextSibling();
    ps.print("if ( " + a + ") {");
    //if (a != 0)
    r = expr(ps,thenpart);
    ps.print("}");

    if (elsepart != null){
        ps.print("else {");
        r = expr(ps,elsepart);
        ps.print("}");
    }

    else r = 0;
```

```
        _t = __t154;
        _t = _t.getNextSibling();
        break;
    }
    case LITERAL_break_loop: {
        AST __t155 = _t;
        AST tmp118_AST_in = (AST)_t;
        match(_t,LITERAL_break_loop);
        _t = _t.getFirstChild();
        ps.println("break;");
        _t = __t155;
        _t = _t.getNextSibling();
        break;
    }
    case LITERAL_skip_iteration: {
        AST __t156 = _t;
        AST tmp119_AST_in = (AST)_t;
        match(_t,LITERAL_skip_iteration);
        _t = _t.getFirstChild();
        ps.print("continue;");
        _t = __t156;
        _t = _t.getNextSibling();
        break;
    }
    case LITERAL_OR: {
        AST __t157 = _t;
        AST tmp120_AST_in = (AST)_t;
        match(_t,LITERAL_OR);
        _t = _t.getFirstChild();
        a=expr(ps,_t);
        _t = _retTree;
        b=expr(ps,_t);
        _t = _retTree;
        _t = __t157;
        _t = _t.getNextSibling();

        ps.print(" || ");

        //if ( a instanceof PfordBool )
        //r = ( ((PfordBool)a).var ? a : expr(ps,#right_or) );
        //else
        //r = a.or( expr(ps,#right_or) );

        break;
    }
    case LITERAL_AND: {
        AST __t158 = _t;
        AST tmp121_AST_in = (AST)_t;
        match(_t,LITERAL_AND);
        _t = _t.getFirstChild();
        a=expr(ps,_t);
        _t = _retTree;
        b=expr(ps,_t);
        _t = _retTree;
        _t = __t158;
        _t = _t.getNextSibling();
```

```
                    ps.print(" && ");

                    // if ( a instanceof PfordBool )
                    //             r = ( ((PfordBool)a).var ?
expr(ps,#right_and) : a );
                    //             else
                    //             r = a.and( expr(ps,#right_and) );

                    break;
                }
                case LITERAL_loopfrom: {
                    AST __t159 = _t;
                    AST tmp122_AST_in = (AST)_t;
                    match(_t,LITERAL_loopfrom);
                    _t = _t.getFirstChild();
                    a=expr(ps,_t);
                    _t = _retTree;
                    b=expr(ps,_t);
                    _t = _retTree;
                    predic = _t==ASTNULL ? null : (AST)_t;
                    c=expr(ps,_t);
                    _t = _retTree;
                    _t = __t159;
                    _t = _t.getNextSibling();

                    AST forbody = predic.getNextSibling();
                    int initial = 0;
                    int finalValue = 0;
                    int increment = 0;

                    //initial = ((PfordInt)a).var;
                    //finalValue = ((PfordInt)b).var;
                    //increment = ((PfordInt)c).var;

                    ps.println("");
                    ps.print("for (int i = ");
                    ps.println(initial + " ; i < " + finalValue + "; i = i +"
+ increment + "){");
                    r = expr(ps,forbody);
                    ps.println("}");

                    break;
                }
                case LITERAL_loop: {
                    AST __t160 = _t;
                    AST tmp123_AST_in = (AST)_t;
                    match(_t,LITERAL_loop);
                    _t = _t.getFirstChild();
                    predicate = _t==ASTNULL ? null : (AST)_t;
                    a=expr(ps,_t);
                    _t = _retTree;
                    _t = __t160;
                    _t = _t.getNextSibling();

                    AST loopbody = predicate.getNextSibling();
                    float temp = 0;
```

```
                    // if( !(a instanceof PfordInt))
                    //             return a.error("loop: expression should be
an int");

                    temp = a;
                    ps.println("");
                    ps.print("for (int i = 0");
                    ps.println(" ; i < " + temp + "; i++){");
                    r = expr(ps,loopbody);
                    ps.println("}");

                    break;
                }
                case LITERAL_while: {
                    AST __t161 = _t;
                    AST tmp124_AST_in = (AST)_t;
                    match(_t,LITERAL_while);
                    _t = _t.getFirstChild();
                    predicateStuff = _t==ASTNULL ? null : (AST)_t;
                    a=expr(ps,_t);
                    _t = _retTree;
                    _t = __t161;
                    _t = _t.getNextSibling();

                    AST whileloopbody = predicateStuff.getNextSibling();

                    ps.println("");
                    ps.print("while (");
                    ps.println("){");

                    r = expr(ps,whileloopbody);
                    ps.println("}");


                    break;
                }
                default: {
                    throw new NoViableAltException(_t);
                }
            }
        }
        catch (RecognitionException ex) {
            reportError(ex);
            if (_t!=null) {_t = _t.getNextSibling();}
        }
        _retTree = _t;
        return r;
    }


    public static final String[] _tokenNames = {
        "<0>",
        "EOF",
        "<2>",
        "NULL_TREE_LOOKAHEAD",
        "SEMI",
        "PLUS",
```

```
"POWER",
"STAR",
"MINUS",
"DIV",
"ASSIGN",
"RSQBRACKET",
"LSQBRACKET",
"LCURLY",
"RCURLY",
"PERCENT",
"COMMA",
"DOT",
"UPLUSEQ",
"UMINUSEQ",
"UMULTEQ",
"UDIVEQ",
"REQ",
"RGEQ",
"RLEQ",
"RNEQ",
"RGT",
"RLT",
"INCR",
"DECR",
"PARENS",
"LETTER",
"DIGIT",
"ID",
"NUMBER",
"STRING",
"WS",
"COMMENT",
"STMT",
"UPLUS",
"UMINUS",
"FUNCCALL",
"VARLIST",
"FUNCVARLIST",
"ARRAY_ROW",
"ARRAY",
"OBJCALL",
"\"startprogram\"",
"\"endprogram\"",
"\"if\"",
"\"(\"",
"\")\"",
"\"then\"",
"\"elsedo\"",
"\"output\"",
"\"input\"",
"\"func\"",
"\"return\"",
"\"loop\"",
"\"runs\"",
"\"while\"",
"\"loopthrough\"",
"\"loopfrom\"",
```

```
        "\"to\"",
        "\"interval\"",
        "\"skip_iteration\"",
        "\"break_loop\"",
        "\"AND\"",
        "\"OR\"",
        "GT",
        "LT",
        "\"NOT\"",
        "\"mod\"",
        "\"true\"",
        "\"false\"",
        "\"object\"",
        "\"TRUE\"",
        "\"True\"",
        "\"FALSE\"",
        "\"False\""
    };

}
```

10.1.7 PfordInterpreter.java

```
/**
   *
   *Author: Deepti,Bhagyashri,Shringika
**/
import java.util.*;
import java.io.*;
import java.io.IOException;
import antlr.CommonAST;
import antlr.collections.AST;
import antlr.RecognitionException;
import antlr.TokenStreamException;
import antlr.TokenStreamIOException;

class PfordInterpreter {
    Symbol symt;
    final static int fc_none = 0;
    final static int fc_break = 1;
    final static int fc_continue = 2;
    final static int fc_return = 3;
    private int control = fc_none;
    private String label;
    private Random random = new Random();


    public PfordInterpreter()
    {
        symt = new Symbol( null);
        registerInternal();
      registerExternal();
    }


    public void registerInternal() {
      PfordBuiltInFunction.register( symt );
    }

     public void registerExternal() {
      PfordIncludeFunction.register( symt );
    }


    public PfordDataType[] convertExprList( Vector v ) {

      PfordDataType[] x = new PfordDataType[v.size()];
      for ( int i=0; i<x.length; i++ )
          x[i] = (PfordDataType) v.elementAt( i );
      return x;
    }

    public static String[] convertVarList( Vector v ) {

      String[] sv = new String[ v.size() ];
      for ( int i=0; i<sv.length; i++ )
          sv[i] = (String) v.elementAt( i );
```

```
  return sv;
}

public static PfordDataType getString( String s )
{
  return new PfordString(s);
}

public static PfordDataType getNumber( String s )
{
  if ( s.indexOf( '.' ) >= 0
       || s.indexOf( 'e' ) >= 0 || s.indexOf( 'E' ) >= 0 )
      return new PfordDouble( Double.parseDouble( s ) );

  return new PfordInt( Integer.parseInt( s ) );
}

public PfordDataType getVariable( String s )
{
  // default static scoping
  PfordDataType x = symt.getValue( s, true, 0 );
  if ( null == x )
      return new PfordVariable( s );
  return x;
}

public PfordDataType getInput(PfordDataType a )
{

  PfordDataType  d = new PfordDataType();
  try{
      //((PfordInt)d).var = System.in.read();
      BufferedReader bk = new BufferedReader (
                                    new InputStreamReader(System.in));
      String sb;
      sb = bk.readLine();
      System.out.println(sb);
      bk.close();
      ((PfordString)d).var = sb;
  }catch( IOException e ) {
      System.err.println( "Error: I/O: " + e );
  }
  return assign(a,d);
}

public PfordDataType rvalue( PfordDataType a )
{
  if ( null == a.name )
      return a;
  if (a instanceof PfordArray)
      return ((PfordArray)a).deepCopy();

  return a.copy();
}

/*public PfordDataType deepRvalue( PfordDataType a )
  {
```

```
      if ( null == a.name )
      return a;
      if ( a instanceof PfordArray )
      return ((PfordArray)a).deepCopy();
      return a.copy();
      }
    */

    public PfordDataType assign( PfordDataType a, PfordDataType b ) {
      if ( null != a.name )
          {
            PfordDataType x = b;

            //PfordDataType x = deepRvalue( b );
            x.setName( a.name );
            symt.setValue( x.name, x, true, 0 );  // scope?
            return x;
          }

      if ( a instanceof PfordArray )
          {
            if ( b instanceof PfordArray )
            ((PfordArray)a).vect.assign( ((PfordArray)b).vect );

            return a;
          }
      return a.error( b, "=" );
    }


public PfordDataType subMatrix( PfordDataType x,
                        PfordDataType[] vexpr )
    {
         if ( x instanceof PfordArray )
           {
           if ( 1 == vexpr.length && (vexpr[0] instanceof PfordInt
                    || vexpr[0] instanceof PfordDouble ))
           {
                return new PfordDouble(

((PfordArray)x).vect.elementAt(PfordInt.intValue( vexpr[0])) );
           }
     }

        return x.error( "vector" );
    }

    public PfordDataType funcInvoke(PfordTreeWalker walker, PfordDataType
func,
                            PfordDataType[] params ) throws
antlr.RecognitionException
    {

      // Is this function an internal function?
      // Names of formal args are not necessary for internal functions.
      if ( ((PfordFunction)func).isInternal() ){
```

```
            return execInternal( ((PfordFunction)func).getInternalId(),params
);
        }

        // func must be an existing function
        if ( !( func instanceof PfordFunction ) )
            return func.error( "not a function" );

        // otherwise check numbers of actual and formal arguments
        String[] args = ((PfordFunction)func).getArgs();
        if ( args.length != params.length )
            return func.error( "unmatched length of parameters" );

        // create a new symbol table
        symt = new Symbol( ((PfordFunction)func).getParentSymbolTable() );
        // assign actual parameters to formal arguments
        for ( int i=0; i<args.length; i++ )
            {
              PfordDataType d = rvalue( params[i] );
              d.setName( args[i] );
              symt.setValue( args[i], d, false, 0 );
            }

        // call the function body
        PfordDataType r = walker.expr( ((PfordFunction)func).getBody() );

        // no break or continue can go through the function
        if ( control == fc_break || control == fc_continue )
            throw new PfordException( "nowhere to break or continue" );

        // if a return was called
        if ( control == fc_return )
            tryResetFlowControl( ((PfordFunction)func).name );

        // remove this symbol table and return
        symt = symt.Parent();
        return r;
    }


    public void funcRegister(String name, String[] args, AST body ) throws
antlr.RecognitionException
    {
      symt.put( name, new PfordFunction( name, args, body, symt) );
    }

    public PfordDataType objInvoke(PfordTreeWalker walker, PfordDataType obj)
throws antlr.RecognitionException
    {
      // func must be an existing function
      if ( !( obj instanceof PfordObject) )
          return obj.error( "not an object" );

      // create a new symbol table
      symt = new Symbol( ((PfordObject)obj).getParentSymbolTable());

      // call the function body
```

```
      PfordDataType r = walker.expr( ((PfordObject)obj).getBody() );

      // no break or continue can go through the function
      if ( control == fc_break || control == fc_continue )
          throw new PfordException( "nowhere to break or continue" );

      // remove this symbol table and return
      symt = symt.Parent();
      return r;
    }

   public void objRegister(String name, AST body ) throws
antlr.RecognitionException
    {
      symt.put( name, new PfordObject( name, body, symt) );
    }

    public void setBreak( String label )
    {
      this.label = label;
      control = fc_break;
    }

    public void setContinue( String label )
    {
      this.label = label;
      control = fc_continue;
    }

    public void setReturn( String label )
    {
      this.label = label;
      control = fc_return;
    }

    public void tryResetFlowControl( String loop_label )
    {
      if ( null == label || label.equals( loop_label ) )
          control = fc_none;
    }

    public void loopNext( String loop_label )
    {
      if ( control == fc_continue )
          tryResetFlowControl( loop_label );
    }

    public void loopEnd( String loop_label )
    {
      if ( control == fc_break )
          tryResetFlowControl( loop_label );
    }

    public boolean canProceed()
    {
      return control == fc_none;
    }
```

```
public PfordInt[] forInit( PfordDataType[] mexpr )
{
  // very much like function call
  for ( int i=0; i<mexpr.length; i++ )
      if ( ! ( mexpr[i] instanceof PfordRange )  )
        {
            mexpr[i].error( "for: [range expected]" );
            return null;
        }
  // create a new symbol table
  symt = new Symbol( symt);
  PfordInt[] x = new PfordInt[mexpr.length];
  for ( int i=0; i<mexpr.length; i++ )
      {
        x[i] = new PfordInt( ((PfordRange)mexpr[i]).range.first() );
        x[i].setName( mexpr[i].name );
        symt.setValue( mexpr[i].name, x[i], false, 0 );
      }
  symt.setReadOnly();
  return x;
}

public boolean forCanProceed( PfordDataType[] mexpr, PfordInt[] values )
{
  if ( control != fc_none )
      return false;
  for ( int i=mexpr.length-1; ; i-- )
      {
        if ( ((PfordRange)mexpr[i]).range.contain( values[i].var ) )
            return true;
        if ( 0 == i )
            return false;
        values[i].var = ((PfordRange)mexpr[i]).range.first();
        values[i-1].var =
            ((PfordRange)mexpr[i-1]).range.next( values[i-1].var );
      }
}

public void forNext( PfordDataType[] mexpr, PfordInt[] values )
{
  values[ values.length-1 ].var++;
  if ( control == fc_continue )
      tryResetFlowControl( mexpr[0].name );
}

public void forEnd( PfordDataType[] mexpr )
{
  if ( control == fc_break )
      tryResetFlowControl( mexpr[0].name );
  // remove this symbol table
  symt = symt.Parent();
}

public PfordDataType execInternal( int id, PfordDataType[] params ) {
  return PfordBuiltInFunction.run( symt, id, params );
}
```

```
    public PfordDataType include(PfordDataType func, PfordDataType[] params)
    {
      if ( ((PfordFunction)func).isInternal() )
          return execExternal( ((PfordFunction)func).getInternalId(),params
);
          return func.error( "not a function" );

    }

    public PfordDataType execExternal( int id, PfordDataType[] params ) {
      return PfordIncludeFunction.run( symt, id, params );
    }


}
```

10.1.8 PfordMain.java

```
/**
 * Insert type's description here
 * Creation Date :(2004-11-11 15:36:15)
 *author: Deepti Jindal
 */
import java.io.* ;
import antlr.CommonAST;
import antlr.collections.AST;
import antlr.debug.misc.ASTFrame;
class Main {
    public static void main(String[] args) {
        try {

            int numParam = args.length;
            String filename = null;
            String newfilename = null;

            if (numParam == 2){
                filename = args[args.length-2];
                newfilename = args[args.length-1];
            }
            else if (numParam == 1) {
                filename = args[args.length-1];
            }

            //String filename = args[args.length-1];
            //System.out.println(args[args.length-1]);

            if (filename.endsWith(".pford")){

                PfordLexer lexer = new PfordLexer((InputStream) new
FileInputStream(filename));
                PfordParser parser = new PfordParser(lexer);


                System.out.println("Invoking Parser:");
                //parser.expr();
                parser.program();

                if ( lexer.nr_error > 0 || parser.nr_error > 0 ) {
                    System.err.println( "Parsing errors found. Stop." );
                    return;
                }


                CommonAST t = (CommonAST) parser.getAST();
                System.out.println(t.toStringTree());

                // Open a window in which the AST is displayed graphically
                //ASTFrame frame = new ASTFrame("AST from the Pford parser",
t);
                //frame.setVisible(true);

                PfordTreeWalker walker = new PfordTreeWalker();
```

```
                System.out.println("Invoking Walker:");


                PfordDataType r = walker.expr(t);
                //System.out.println("Value is "+r);

                r.print();

                FileOutputStream fos = new FileOutputStream(newfilename);
                PrintStream ps = new PrintStream(fos);
                //System.setErr(ps);

                if (numParam ==2){
                    if(newfilename.endsWith(".java")) {
                        String nameOfClass = newfilename.substring(0,
newfilename.indexOf(".java"));
                        PfordTreeWalkerJava walker2 = new
PfordTreeWalkerJava();
                        ps.println("class "+  nameOfClass +"{");//start class
                        ps.println("public static void main(String[]
args){");//start main

                        float r2 = walker2.expr(ps, t);

                        ps.println("}");//end main
                        ps.println("}");//end class
                    }

                    else if (newfilename.endsWith(".cpp")) {
                        PfordTreeWalkerCpp walker2 = new
PfordTreeWalkerCpp();

                        ps.println("void main(){");//start main
                        float r2 = walker2.expr(ps, t);
                        ps.println("}");//end main
                    }


                }
            }
            else{
                System.err.println("Invalid file format must end with .pford
extension");
            }
        } catch (Exception e) {
            System.err.println("exception: "+e);
        }

    }
}
```

10.1.9 PfordArray.java

```
/**
 * Insert type's description here
 * Creation Date :(2004-11-11 15:36:15)
 *author: Shringika Porwal
 */
import java.io.PrintWriter;
import java.util.*;

class PfordArray extends PfordDataType {
    PfVector vect;

    PfordArray(PfVector vect) {
        this.vect=vect;
    }

    public PfordDataType copy() {
        return new PfordArray(vect);
    }

    public PfordDataType deepCopy() {
        return new PfordArray(vect.copy());
    }

    public PfordDataType eq(PfordDataType b) {
        if (b instanceof PfordArray) {
            return new PfordBool(vect.eq(((PfordArray)b).vect));
        }
        return b.eq(this);
    }

    private static final int getDim(PfordDataType x) {
        if (x instanceof PfordDouble || x instanceof PfordInt)
            return 1;
        if ( x instanceof PfordArray)
            return ((PfordArray)x).vect.size();
        x.error("not an aray or no elements");
        return 0;
    }
    //verticaljoin of arrays

    public static PfordDataType joinVert(PfordDataType []x) {
        if (x.length==0)
            throw new IllegalArgumentException("no data in array");

        int vectorSize=x.length;

        PfordArray y = new PfordArray(new PfVector(vectorSize));
        int coly=0;

        for ( int i=0;i<vectorSize;i++)
            y.vect.set( coly++, PfordDouble.doubleValue( x[i] ));

        return y;
    }
```

```
    public PfordDataType mul( PfordDataType b ) {

        vect.selfmul( PfordDouble.doubleValue( b ) );
        return this;
    }

    public PfordDataType neq( PfordDataType b ) {
        if ( b instanceof PfordArray)

            return new PfordBool(vect.neq(((PfordArray)b).vect));
        return b.neq(this);
    }

    public PfordDataType times(PfordDataType b) {

        return new PfordArray(vect.times(PfordDouble.doubleValue(b)));
    }

    public String typename() {
        return "vector";
    }

    public PfordDataType uminus() {
        return new PfordArray(vect.uminus());//negation of array
    }

    public void print(PrintWriter w) {
        if (name !=null)
            w.println(name + "=");
        vect.print(w,8,4);//print method frm Pfvector that takes
o/p,numberformat and width as args
    }

    public PfordDataType div(PfordDataType b) {
        return new PfordArray(vect.div(PfordDouble.doubleValue(b)));
    }

    public PfordDataType sortArray() {
        return new PfordArray(vect.arraySelectionSort());//sort array and
return sorted one
    }

}
```

10.1.10 PfordBool.java

```
/**
 * insert the Type's description here.
 *Creation Date: (2004-11-05 10:11:12)
 * @author : Shringika, Deepti
 */
import java.io.PrintWriter;
class PfordBool extends PfordDataType {
    boolean var;

    PfordBool( boolean var) {
        this.var=var;
    }

    public PfordDataType and(PfordDataType b) {
        if (b instanceof PfordBool)
            return new PfordBool(var && ((PfordBool)b).var);
        return error(b,"and");
    }

    public PfordDataType copy() {
        return new PfordBool(var);
    }

    public PfordDataType eq( PfordDataType b) {
        if (b instanceof PfordBool)
            return new PfordBool(( var && ((PfordBool)b).var )
            || ( !var && !((PfordBool)b).var ));

        return error(b,"==");
    }
    public PfordDataType ne( PfordDataType b) {
        if (b instanceof PfordBool)
            return new PfordBool(( var && !((PfordBool)b).var )
            || ( !var && ((PfordBool)b).var ));

        return error(b,"!=");
    }

    public PfordDataType not() {
        return new PfordBool( !var);
    }

    public PfordDataType or( PfordDataType b) {
        if (b instanceof PfordBool)
            return new PfordBool( var || ((PfordBool)b).var );
        return error(b,"or");
    }

    public void print( PrintWriter w) {
        if (name != null)
            w.print( name + "=");
        w.println(var ? "true" : "false");
    }
}
```

10.1.11 PfordBuiltInFunction.java

```
/**
 * insert the Type's description here.
 *Creation Date: (2004-11-05 10:11:12)
 * @author : Shringika
 */
import java.util.*;
import java.io.*;  //for using IOException

/** functions for string and array operations like min,max
,sum,std.deviation,mean,length,convert to lowercase uppercase
 * 2004/12/13
 */

class PfordBuiltInFunction {
    static int Intval =0;
    static double doubleval=0;
    static boolean boolval=false;
    static String stringval="";
    static PfordFunctionsAll functionsAll=new PfordFunctionsAll();


    final static int f_splitString = 0; //?
    final static int f_join = 1;
    final static int f_contains = 2; //for array
    final static int f_toLower = 3;
    final static int f_toUpper = 4;
    final static int f_stringLength = 5;
    //arrayfunctions
    final static int f_sort = 6;
    final static int f_min = 7;
    final static int f_max = 8;
    final static int f_stdDev = 9;
    final static int f_sum = 10;
    final static int f_mean = 11;
    final static int f_arrayLength = 12;
    final static int f_factorial = 13;
    final static int f_replace = 14;//in string
    final static int f_print = 15;
    final static int f_range = 16;
    final static int f_size = 17;




    public static void register(Symbol st ) {

        //st.setValue("splitString", new PfordFunction(
null,f_splitString),false,0); //0
        st.put( "splitString", new PfordFunction( null,f_splitString ) ); //0
        st.put( "join", new PfordFunction( null,f_join ) ); //1
        st.put( "contains", new PfordFunction( null,f_contains ) ); //2
        st.put( "toLower", new PfordFunction(null, f_toLower ) ); //3
```

```
        st.put( "toUpper", new PfordFunction( null,f_toUpper ) ); //4
        st.put( "stringLength", new PfordFunction(null, f_stringLength) );
//5
        st.put( "sort", new PfordFunction( null, f_sort) ); //6
        st.put( "min", new PfordFunction( null, f_min) ); //7
        st.put( "max", new PfordFunction( null, f_max) ); //8
        st.put( "stdDev", new PfordFunction( null, f_stdDev) ); //9
        st.put( "sum", new PfordFunction( null, f_sum) ); //10
        st.put( "mean", new PfordFunction( null, f_mean) ); //11
        st.put( "arrayLength", new PfordFunction( null, f_arrayLength) );
//12
        st.put( "factorial", new PfordFunction( null, f_factorial) ); //13
        st.put( "replace", new PfordFunction( null, f_replace) ); //14
        st.put( "print", new PfordFunction( null, f_print) ); //15
        st.put( "range", new PfordFunction( null, f_range) ); //16
        st.put( "size", new PfordFunction( null, f_size) ); //17
    }

    private static boolean isString(PfordDataType b) {
        if (! (b instanceof PfordString))
            return false;
        return true;
    }
    public static PfordDataType run(Symbol st,int id,PfordDataType[] params)
{

        switch (id) {

            case f_join://1

                if (params.length !=2)
                    throw new PfordException("join of Strings takes 2
parameter;");
                if (!(params[0] instanceof PfordString))
                    throw new PfordException("join of  string takes string
argument;");
                if (!(params[1] instanceof PfordString))
                    throw new PfordException("join of  string takes string
argument;");
                PfordString strnew =
functionsAll.join(((PfordString)params[0]),((PfordString)params[1]));
                return strnew;

            case f_toLower://3

                if (params.length !=1)
                    throw new PfordException("toLower accepts one
parameter;");
                return functionsAll.toLower(((PfordString)params[0]));

            case f_toUpper://4

                if (params.length !=1)
                    throw new PfordException("toLower accepts one
parameter;");
                return functionsAll.toUpper(((PfordString)params[0]));
```

```
            case f_stringLength://5

                if (params.length !=1)
                    throw new PfordException("stringLength() accpets one
parameter;");
                Intval  = functionsAll.stringLength((PfordString)params[0]);
                return new PfordInt(Intval);

            case f_sort://6

                if (params.length !=1)

                    throw new PfordException("sort() accpets one
parameter(s);");

                if ((params[0] instanceof PfordArray))

                    return new
PfordArray(functionsAll.sort(((PfordArray)params[0]).vect));

                if ((params[0] instanceof PfordStringArray))

                    return new
PfordStringArray(functionsAll.sortStrings(((PfordStringArray)params[0]).vect)
);

            case f_contains://2

                if (params.length !=2)
                    throw new PfordException("contains() accepts two
parameters;");


boolval=functionsAll.contains(((PfordArray)params[0]).vect,PfordDouble.double
Value(params[1]));
                return new PfordBool(boolval);

            case f_min://7

                if (params.length !=1)
                    throw new PfordException("Min accepts/takes  one
parameter;");

                doubleval  = functionsAll.min(((PfordArray)params[0]).vect);
                return new PfordDouble(doubleval);

            case f_max://8

                if (params.length !=1)
                    throw new PfordException("max accepts/takes  one
parameter;");
                doubleval  = functionsAll.max(((PfordArray)params[0]).vect);
                return new PfordDouble(doubleval);

            case f_stdDev://9

                if (params.length !=1)
```

```
                    throw new PfordException(" std. dev. accpets one
parameter;");
                doubleval  =
functionsAll.stdDev(((PfordArray)params[0]).vect);
                return new PfordDouble(doubleval);

            case f_sum://10

                if (params.length !=1)
                    throw new PfordException("sum of arrays takes one
parameter;");
                doubleval  = functionsAll.sum(((PfordArray)params[0]).vect);
                return new PfordDouble(doubleval);

            case f_mean://11

                if (params.length !=1)
                    throw new PfordException("mean() accpets one
parameter;");
                doubleval  = functionsAll.mean(((PfordArray)params[0]).vect);
                return new PfordDouble(doubleval);

            case f_arrayLength://11

                if (params.length !=1)
                    throw new PfordException("arrayLenth() accpets one
parameter;");
                Intval  =
functionsAll.arrayLength(((PfordArray)params[0]).vect);
                return new PfordInt(Intval);

            case f_print://15

                for (int i=0;i<params.length;i++)
                    params[i].print();
                return null;

            case f_range://16
                if (params.length !=1)
                    throw new PfordException("range() accepts 1
parameter(s);");
                doubleval=functionsAll.range(((PfordArray)params[0]).vect);
                return new PfordDouble(doubleval);


            case f_size: //17
                if ( params.length != 1 )
                    throw new PfordException( "size() accepts 1 parameter(s)"
);
                Intval =functionsAll.size(((PfordArray)params[0]).vect);
                return new PfordInt(Intval);



            default:
                throw new PfordException("built in function not defined for
this language");
```

```
        }//end of switch


    }//end of run
}//end of class def
```

10.1.12 PfordDataType.java

```
/**
 * insert the Type's description here.
 *Creation Date: (2004-11-05 10:11:12)
 * @author : Shringika
 */

import java.io.PrintWriter;
import java.sql.*;
import java.util.*;
import java.io.IOException;

public class PfordDataType

{

    String name;   //used in  hash table

    public PfordDataType() {
        name=null;

    }

    public PfordDataType(String name) {
        this.name=name;

    }

    public String typename() {
        return "unknown";
    }

    public PfordDataType copy() {
        return new PfordDataType();
    }

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

    public PfordDataType error(String msg) {
        throw new PfordException("illegal operation:" + msg
        + "(<" + typename() + ">"
        +(name !=null ? name: "<?>")
        + ")" );

    }

    public PfordDataType error(PfordDataType b, String msg ) {
        if (null == b)
        { return error(msg); }

        throw new PfordException(
        "illegal operation: " + msg
        + "(<" + typename() + ">"
```

```
    + (name !=null ? name : "<?>")
    + "and"
    + "<" + typename() + ">"
    + (name !=null ? name : "<?>")
    + ")" );
}

//comparison operators

public PfordDataType eq(PfordDataType b) {
    return error(b, "==");
}

public PfordDataType neq(PfordDataType b) {
    return error(b, "!=");
}

public PfordDataType gt(PfordDataType b) {
    return error( b, ">");
}

public PfordDataType gte(PfordDataType b) {
    return error(b, ">=");
}

public PfordDataType lt(PfordDataType b) {
    return error(b, "<");
}

public PfordDataType lte(PfordDataType b) {
    return error(b, "<=");
}

//arithmetic operators
public PfordDataType assign(PfordDataType b) {
    return error(b, "=");
}


public PfordDataType uminus() {
    return error("-");
}

public PfordDataType power(PfordDataType b) {
    return error(b, "^");
}

public PfordDataType add_assign(PfordDataType b) {
    return error(b, "+=");
}

public PfordDataType sub_assign(PfordDataType b) {
    return error(b, "-=");
}
public PfordDataType mult_assign(PfordDataType b) {
    return error(b, "*=");
}
```

```
    public PfordDataType div_assign(PfordDataType b) {
        return error(b, "/=");
    }

    public PfordDataType add(PfordDataType b) {
        return error("+");
    }
    public PfordDataType sub(PfordDataType b) {
        return error(b, "-");
    }
    public PfordDataType times(PfordDataType b) {
        return error(b, "*");
    }

    public PfordDataType div(PfordDataType b) {
        return error(b, "/");
    }

    public PfordDataType modulus(PfordDataType b) {
        return error(b, "mod");
    }
    public PfordDataType rem(PfordDataType b) {
        return error(b, "%=");
    }

    public PfordDataType percent(PfordDataType b) {
        return error(b, "%");
    }

    //logical operators
    public PfordDataType not() {
        return error("not");
    }

    public PfordDataType and(PfordDataType b) {
        return error(b, "and");
    }

    public PfordDataType or(PfordDataType b) {
        return error(b, "or");
    }


    public void print(PrintWriter w) {
        if (name !=null)
            w.print(name + "=");
        w.println(",undefined>");
    }
    public void print() {
        print(new PrintWriter(System.out,true));
        //println will flush the o/p buffer
    }


}
```

10.1.13 PfordDouble.java

```
/**
 * Insert the type's description here.
 *creation Date : (2004-11-10 16:20:28)
 *author : shringika, bhagyashri, deepti
 */
import java.lang.Math;
import java.io.PrintWriter;

class PfordDouble extends PfordDataType {
    double var;
    public PfordDouble(double x) {
        var=x;
    }

    public PfordDataType add(PfordDataType b) {
        return new PfordDouble(var+doubleValue(b));
    }

    public PfordDataType copy() {
        return new PfordDouble(var);
    }

    public static double doubleValue(PfordDataType b) {
        if (b instanceof PfordDouble)
            return ((PfordDouble)b).var;
        if (b instanceof PfordInt)
            return (double)((PfordInt)b).var;
        b.error("cast to double");
        return 0;
    }

    public PfordDataType uminus() {
        return new PfordDouble(-var);
    }

    public PfordDataType eq(PfordDataType b) {
        return new PfordBool(var == doubleValue(b));
    }

    public PfordDataType neq(PfordDataType b) {
        return new PfordBool(var != doubleValue(b));
    }

    public PfordDataType gte(PfordDataType b) {
        return new PfordBool(var >= doubleValue(b));
    }

    public PfordDataType gt(PfordDataType b) {
        return new PfordBool(var > doubleValue(b));
    }

    public PfordDataType lte(PfordDataType b) {
        return new PfordBool(var <= doubleValue(b));
```

```
}

public PfordDataType div(PfordDataType b) {
    return new PfordDouble(var / doubleValue(b));
}

public PfordDataType lt(PfordDataType b) {
    return new PfordBool(var < doubleValue(b));
}

public PfordDataType sub(PfordDataType b) {
    return new PfordDouble(var - doubleValue(b));
}

public PfordDataType times(PfordDataType b) {
    if (b instanceof PfordArray)
        return b.times(this);

    return new PfordDouble(var * doubleValue(b));
}

public PfordDataType percent(PfordDataType b) {
    return new PfordDouble(var/doubleValue(b)*100);
}

public PfordDataType modulus(PfordDataType b) {
    return new PfordDouble(var % doubleValue(b));
}

public PfordDataType power(PfordDataType b) {
    if (b instanceof PfordDouble)
        return new PfordDouble(Math.pow(var,doubleValue(b)));
    return error(b,"incorrect datat types");
}

public PfordDataType rem(PfordDataType b) {
    var %= doubleValue(b);
    return this;
}
public PfordDataType sub_assign(PfordDataType b) {
    var -= doubleValue(b);
    return this;
}

public PfordDataType mult_assign(PfordDataType b) {
    var *= doubleValue(b);
    return this;
}
public PfordDataType div_assign(PfordDataType b) {
    var /= doubleValue(b);
    return this;
}


public void print(PrintWriter w) {
    if (name !=null)
        w.print(name + "=");
```

```
        w.println(Double.toString(var));
    }
    public String typename() {
        return "double";
    }


}
```

10.1.14 PfordException.java

```
/**
 * Insert the type's description here.
 * Creation date: (2004-11-9 05:30:00)
 * @author: Shringika Porwal
 */
class PfordException extends RuntimeException {
    PfordException( String msg ) {
        System.err.println( "Error: "+ msg );
    }
}import java.io.PrintWriter;
import antlr.collections.AST;

class PfordFunction extends PfordDataType {

    String[] args;
    AST body;
    Symbol pst; // the symbol table
    int id;
```

10.1.15 PfordFunction.java

```
/**
   *Insert type desctiption here
      * Creation  date: (2004-11-28 14:30:00)
      * @author : shringika, deepti bhagyashri
**/
    public PfordFunction( String name, String[] args, AST body, Symbol pst) {
        super( name );
        this.args = args;
        this.body = body;
        this.pst = pst;
    }

    public PfordFunction( String name, int id ) {
        super( name );

        this.args = null;
        this.id = id;
        pst = null;
        body = null;
    }

    public final boolean isInternal() {
        return body == null;
    }

    public final int getInternalId() {
        return id;
    }

    public String typename() {
        return "function";
    }
```

```
    public PfordDataType copy() {
        return new PfordFunction( name, args, body, pst );
    }

    public void print( PrintWriter w ) {
        if ( body == null ) {
            w.println( name + " = <internal-function> #" + id );

        }
        else {
            if ( name != null )
                w.print( name + " = " );
            w.print( "<function>(" );
            for ( int i=0; ; i++ ) {
                w.print( args[i] );
                if ( i >= args.length - 1 )
                    break;
                w.print( "," );
            }
            w.println( ")" );
        }
    }

    public String[] getArgs() {
        return args;
    }

    public Symbol getParentSymbolTable() {
        return pst;
    }

    public AST getBody() {
        return body;
    }
}
```

10.1.16 PfordFunctionsAll.java

```
/**
 * Insert the type's description here.
 * Creation date: (2004-11-9 05:30:00)
 * @author: Shringika Porwal
 */
import java.util.*;
import java.io.*;

public class PfordFunctionsAll implements Cloneable {
    // constructor for this class
    public PfordFunctionsAll()
    { super();
    }


    public boolean contains(PfVector vect, double number) {
        return vect.contains(number);
    }

    public int size(PfVector vect) {
        return vect.size();
    }


    public int factorial(int i) {
        int factorial = 1;
        for (int j = 1; j < i + 1; j++)
            factorial *= j;
        return factorial;
    }

    public double sum(PfVector vect) {
        return vect.sumArray();
    }

    public double mean(PfVector vect) {
        double sum = 0;
        int i=1;
        for (i = 0; i < vect.size(); i++) {
            sum = sum+ vect.elementAt(i);
        }
        return sum/i;
    }

    public double min(PfVector vect) {
        double minimum=0;
        minimum= vect.min();
        return minimum;
    }

    public double max(PfVector vect) {
        return vect.max();
    }
```

```
    public PfVector sort(PfVector vect) {
        PfVector sortedVector = new PfVector(vect.size());
        PfVector tempVector = new PfVector(vect);
        int position = 0;
        int size = vect.size();
        int innerSize = size;
        for (int i = 0; i < size; i ++) {
            double mimValue = tempVector.elementAt(0);
            for (int j=0; j < innerSize; j ++) {
                double currentElement = vect.elementAt(j);
                if(currentElement < mimValue ) {
                    mimValue = currentElement;
                    position = j;
                }
            }
            sortedVector.set(i, mimValue);
            tempVector.removeElementAt(position);
            innerSize--;
        }
        return sortedVector;
    }

    public PfStringArray sortStrings(PfStringArray vect) {

        return vect.stringArraySort();
    }

    public double stdDev(PfVector vect) {

        return vect.stddev();
    }

    public int arrayLength(PfVector vect) {
        return vect.Arraylength();
    }

    public double range(PfVector vect) {

        double range = vect.max() - vect.min();
        return range;
    }

    public int stringLength( PfordString str)
    {return str.stringLength();}

    public PfordString toLower( PfordString str)
    {return str.toLower();}

    public PfordString toUpper( PfordString str)
    {return str.toUpper();}

    public PfordString join(PfordString str1,PfordString str2)
{//PfordString(((PfordString)b).var)
        return ((PfordString)(str1.join(str2)));
    }
}
```

10.1.17 PfordIncludeFunction.java

```
/**
 * Insert type's description here
 * Creation Date :(2004-11-11 15:36:15)
 *author: Shringika Porwal, Deepti Jindal
 */

import java.util.*;
import java.io.*;  //for using IOException

/** functions for string and array operations like min,max
,sum,std.deviation,mean,length,convert to lowercase uppercase
 * 2004/12/13
 */

class PfordIncludeFunction {
    static int Intval =0;
    static double doubleval=0;
    static boolean boolval=false;
    static String stringval="";
    static PfordFunctionsAll functionsAll=new PfordFunctionsAll();

    //arrayfunctions
    final static int f_sort = 6;
    final static int f_min = 7;
    final static int f_max = 8;
    final static int f_stdDev = 9;
    final static int f_sum = 10;
    final static int f_mean = 11;
    final static int f_arrayLength = 12;
    final static int f_factorial = 13;
    final static int f_print = 15;
    final static int f_range = 16;

    public static void register(Symbol st ) {
        st.put( "mean", new PfordFunction( null, f_mean) ); //11
        st.put( "arrayLength", new PfordFunction( null, f_arrayLength) );
//12
        st.put( "factorial", new PfordFunction( null, f_factorial) ); //13
        st.put( "print", new PfordFunction( null, f_print) ); //15
        st.put( "range", new PfordFunction( null, f_range) ); //16
    }

    private static boolean isString(PfordDataType b) {
        if (! (b instanceof PfordString))
            return false;
        return true;
    }
    public static PfordDataType run(Symbol st,int id,PfordDataType[] params)
{

        switch (id) {
            case f_sort://6
```

```
                if (params.length !=1)

                    throw new PfordException("sort() accpets one
parameter(s);");

                if ((params[0] instanceof PfordArray))

                    return new
PfordArray(functionsAll.sort(((PfordArray)params[0]).vect));

                if ((params[0] instanceof PfordStringArray))

                    return new
PfordStringArray(functionsAll.sortStrings(((PfordStringArray)params[0]).vect)
);


            case f_min://7

                if (params.length !=1)
                    throw new PfordException("Min accepts/takes  one
parameter;");

                doubleval  = functionsAll.min(((PfordArray)params[0]).vect);
                return new PfordDouble(doubleval);

            case f_max://8

                if (params.length !=1)
                    throw new PfordException("max accepts/takes  one
parameter;");
                doubleval  = functionsAll.max(((PfordArray)params[0]).vect);
                return new PfordDouble(doubleval);

            case f_stdDev://9

                if (params.length !=1)
                    throw new PfordException(" std. dev. accpets one
parameter;");
                doubleval  =
functionsAll.stdDev(((PfordArray)params[0]).vect);
                return new PfordDouble(doubleval);

            case f_sum://10

                if (params.length !=1)
                    throw new PfordException("sum of arrays takes one
parameter;");
                doubleval  = functionsAll.sum(((PfordArray)params[0]).vect);
                return new PfordDouble(doubleval);

            case f_mean://11

                if (params.length !=1)
                    throw new PfordException("mean() accpets one
parameter;");
                doubleval  = functionsAll.mean(((PfordArray)params[0]).vect);
```

```
                return new PfordDouble(doubleval);

            case f_arrayLength://12

                if (params.length !=1)
                    throw new PfordException("arrayLenth() accpets one
parameter;");
                Intval  =
functionsAll.arrayLength(((PfordArray)params[0]).vect);
                return new PfordInt(Intval);

            case f_factorial://13

                if (params.length !=1)
                    throw new PfordException("factorial accpets one
parameter;");
                Intval  =
functionsAll.factorial(PfordInt.intValue(params[0]));
                return new PfordInt(Intval);

            case f_print://15

                for (int i=0;i<params.length;i++)
                    params[i].print();
                return null;

            case f_range://16
                if (params.length !=1)
                    throw new PfordException("range() accepts 1
parameter(s);");
                doubleval=functionsAll.range(((PfordArray)params[0]).vect);
                return new PfordDouble(doubleval);

            default:
                throw new PfordException("built in function not defined for
this language");

        }//end of switch


    }//end of run
}//end of class def
```

10.1.18 PfordInt.java

```
/**
 * Insert type's description here
 * Creation Date :(2004-11-11 15:36:15)
 *author: Shringika Porwal, Deepti Jindal
 */

import java.io.PrintWriter;

public class PfordInt extends PfordDataType

{
    int var;
    public PfordInt( int x ) {
        var = x;
    }

    public PfordDataType copy() {
        return new PfordInt(var);
    }

    public String typename() {
        return "int";
    }

    public static int intValue( PfordDataType b ) {
        if (b instanceof PfordDouble)
            return (int)((PfordDouble)b).var;
        if ( b instanceof PfordInt )
            return ((PfordInt)b).var;
        b.error( "cast to int" );
        return 0;
    }

    public PfordDataType uminus() {
        return new PfordInt(-var);
    }

    public PfordDataType add_assign( PfordDataType b ) {
        var += intValue( b );   //method for +=operator
        return this;
    }

    public PfordDataType sub_assign( PfordDataType b ) {
        var -= intValue( b );  //method for -=operator
        return this;
    }
    public PfordDataType mult_assign( PfordDataType b ) {
        var *= intValue( b );   //method for *=operator
        return this;
    }
    public PfordDataType div_assign( PfordDataType b ) {
        var /= intValue( b );   //method for /=operator
        return this;
    }
```

```
public PfordDataType power(PfordDataType b) {

    if (b instanceof PfordInt)
    {int temp = 1;
     for ( int i = 0; i < Math.abs(intValue(b)); i++) {
         if(intValue(b) >0){
             temp = temp * var;
         }
         else
             b.error("Use double version for a negative power");
     }
     var = temp;

    }
    return this;
}

public PfordDataType add(PfordDataType b)

{
    if (b instanceof PfordInt)
        return new PfordInt(var + intValue(b));
    return new PfordDouble(var + PfordDouble.doubleValue(b));
}

public PfordDataType eq(PfordDataType b) {
    if (b instanceof PfordInt)
        return new PfordBool(var == intValue(b));
    return b.eq(this);
}

public PfordDataType neq(PfordDataType b) {
    if (b instanceof PfordInt)
        return new PfordBool(var != intValue(b));
    return b.neq(this);
}

public PfordDataType gte(PfordDataType b) {
    if (b instanceof PfordInt)
        return new PfordBool(var >= intValue(b));
    return b.lte(this);
}

public PfordDataType gt(PfordDataType b) {
    if (b instanceof PfordInt)

        return new PfordBool(var > intValue(b));
    return b.lt(this);
}

public PfordDataType lte(PfordDataType b) {
    if (b instanceof PfordInt)
        return new PfordBool(var <= intValue(b));
    return b.gte(this);
}
```

```
    public PfordDataType lt(PfordDataType b) {
        if (b instanceof PfordInt)
            return new PfordBool(var < intValue(b));
        return b.gt(this);

    }

    public PfordDataType sub(PfordDataType b) {
        if (b instanceof PfordInt)
            return new PfordInt(var - intValue(b));
        return new PfordDouble(var - PfordDouble.doubleValue(b));
    }

    public PfordDataType div(PfordDataType b) {
        if (b instanceof PfordInt)
            return new PfordInt(var / intValue(b));
        return new PfordDouble(var / PfordDouble.doubleValue(b));
    }
    public PfordDataType times(PfordDataType b) {
        if (b instanceof PfordArray)
            return b.times(this);
        if (b instanceof PfordInt)
            return new PfordInt(var * intValue(b));
        return new PfordDouble(var * PfordDouble.doubleValue(b));
    }


    public PfordDataType percent(PfordDataType b) {
        return new PfordInt(var/intValue(b)*100);
    }

    public PfordDataType modulus(PfordDataType b) {
        if (b instanceof PfordInt)
            return new PfordInt(var % intValue(b));
        return new PfordDouble(var % PfordDouble.doubleValue(b));
    }

    public PfordDataType rem(PfordDataType b) {
        var %=intValue(b);
        return this;
    }

    public void print(PrintWriter w) {
        if (name !=null)
            w.print(name + "=");
        w.println(Integer.toString(var));
    }


}
```

10.1.19 PfordStringArray.java

```
/**
 * Insert type's description here
 * Creation Date :(2004-11-11 15:36:15)
 *author: Shringika Porwal
 */
import java.io.PrintWriter;
import java.util.*;

class PfordStringArray extends PfordDataType {
    PfStringArray vect;

    PfordStringArray(PfStringArray vect) {
        this.vect=vect;
    }

    public PfordDataType copy() {
        return new PfordStringArray(vect);
    }

    public PfordDataType deepCopy() {
        return new PfordStringArray(vect.copy());
    }

    public PfordDataType eq(PfordDataType b) {
        if (b instanceof PfordStringArray) {
            return new PfordBool(vect.eq(((PfordStringArray)b).vect));
        }
        return b.eq(this);
    }

    private static final int getDim(PfordDataType x) {
        if (x instanceof PfordDouble || x instanceof PfordInt)
            return 1;
        if ( x instanceof PfordStringArray)
            return ((PfordStringArray)x).vect.size();
        x.error("not an aray or no elements");
        return 0;
    }
    //verticaljoin of arrays

    public static PfordDataType joinVert(PfordDataType []x) {
        if (x.length==0)
            throw new IllegalArgumentException("no data in array");
        int vectorSize=x.length;
        PfordStringArray y=new PfordStringArray(new
PfStringArray(vectorSize));
        int coly=0;
        for ( int i=0;i<vectorSize;i++)
            y.vect.set( coly++, PfordString.stringVal( x[i] )
            );

        return y;
    }
```

```
    public PfordDataType neq( PfordDataType b ) {
        if ( b instanceof PfordStringArray)

            return new PfordBool(vect.neq(((PfordStringArray)b).vect));
        return b.neq(this);
    }

    public String typename()
    {return "string vector";
    }


    public void print(PrintWriter w) {
        if (name !=null)
            w.println(name + "=");
        vect.print(w,8,4);//print method frm PfStringArray that takes
o/p,numberformat and width as args
    }


    public PfordDataType sortArray() {
        return new PfordStringArray(vect.stringArraySort());//sort array and
return sorted one
    }



}
```

10.1.20 PfordString.java

```
/**
 * Insert type's description here
 * Creation Date :(2004-11-11 15:36:15)
 *author: Shringika Porwal
 */

import java.lang.String;
import java.io.PrintWriter;

class PfordString extends PfordDataType {
    String var;

    public PfordString(String var) {
        this.var = var;
    }
    //modified on 12/13/2004

    public static String stringVal(PfordDataType b) {
        if ( b instanceof PfordString)
        {   return ((PfordString)b).var;
        }
        return ("not a string");
    }

    public PfordDataType add_assign(PfordDataType b) {
        if (b instanceof PfordString) {
            return new PfordString(var + ((PfordString)b).var);
        }
        return error(b,"+=");
    }

    public PfordDataType copy() {
        return new PfordString(var);
    }

    public PfordDataType add(PfordDataType b) {
        if ( b instanceof PfordString )
            return new PfordString(var + ((PfordString)b).var);
        return error(b,"+");

    }

    public void print( PrintWriter w) {
        if (name != null)
            w.print(name + "=");
        w.print(var);
        w.println();

    }

    public String typename() {
        return "string";
    }
```

```
    public PfordDataType join(PfordString b) {
        if ( b instanceof PfordString )
            return new PfordString(var + ((PfordString)b).var);
        return error(b,"not joining a string");

    }

    public PfordString toLower() {
        return new PfordString(var.toLowerCase());
    }

    public PfordString toUpper() {
        return new PfordString(var.toUpperCase());
    }


    public int stringLength() {
        return var.length();

    }

    public String[] splitString(PfordDataType b) {
        if ( b instanceof PfordString )
            return var.split(((PfordString)b).var);
        error(b,"trying to split string with non strings");
        return null;
    }


    public boolean contains(PfordDataType b) {
        if ( b instanceof PfordString )
            return var.equals(((PfordString)b).var);
        error(b,"trying to find a non string");
        return false;
    }
}
```

10.1.21 PfordVariable.java

```
/**
 *insert type's descritpion here.
 *Creation Date: (2004-11-08 09:20:10)
 *author: deepti
 */

import java.io.PrintWriter;

class PfordVariable extends PfordDataType {
    public PfordVariable( String name ) {
        super( name );
    }
    public String typename() {
        return "undefined variable";
    }
    public PfordDataType copy() {
        throw new PfordException( "Variable " + name + " not defined" );
    }
    public void print( PrintWriter w ) {
        w.println( name + " = <undefined>" );
    }
}//modified on 12/10/2004 for sort of arrays
```

10.1.22 PfStringArray.java

```
/**
 *insert type's descritpion here.
 *Creation Date: (2004-11-08 09:20:10)
 *author: shringika
 */

import java.io.*;
import java.text.*;
import java.util.*;
//import java.lang.Math;

public class PfStringArray implements Cloneable {


    /* internal data structure
     * a: array to store array entries
     * n: number of columns/elements
     */
    String [] a;
    //= { "Meal", "aan", "Raoul", "sico" };


    int n;

    public PfStringArray( int n ) {
        this.n = n;
        a = new String[n];
    }


    public PfStringArray( int n, String[] array) {
        this.n = n;

        a = array;
    }


    public PfStringArray( PfStringArray vect ) {
        n = vect.n;
        a = vect.a;
    }

    public PfStringArray assign( PfStringArray b ) {
        if ( n != b.n )
            throw new IllegalArgumentException( dimErrMsg( b )
            );
        for ( int i=0; i<n; i++ )
            set( i, b.get( i ) );//  Replace the element at ith position in a
with the specified element in b.
        return this;
    }

    public Object clone() {
        return copy();
    }
```

```
public boolean contains( String x ) {
    for ( int i=0; i<n; i++ )
        if ( get(i) == x )
            return true;
    return false;
}

public final PfStringArray copy() {
    PfStringArray x = new PfStringArray(n);
    for ( int i=0; i<n; i++ ) {
        x.a[i] = a[i];
    }
    return x;
}

// Returns a string that indicates an error for array dimensions
private final String dimErrMsg() {
    return "Invalid dimensions: " + n;
}
private final String dimErrMsg( PfStringArray b ) {
    return "Dimensions do not match: " + n + " <=> "
    + b.n;
}

public String elementAt(int i) {

    return a[i];
}


public boolean neq( PfStringArray b ) {
    if ( n != b.n )
        throw new IllegalArgumentException( dimErrMsg( b )
        );
    for ( int i=0; i<n; i++ )
        if ( get(i) != b.get(i) )
            return true;
    return false;
}

public boolean eq( PfStringArray b ) {
    if ( n != b.n )
        throw new IllegalArgumentException( dimErrMsg( b )
        );

    for ( int i=0; i<n; i++ )
        if ( !(get(i) == b.get(i)) )
            return false;
    return true;
}
public String firstElement() {
    return a[0];
}

public final String get( int i) {
    return a[i];
}
```

```
public boolean isEmpty() {
    if (n == 0)
        return true;
    return false;
}

public String lastElement() {
    return a[n-1];
}

public final void removeElementAt( int i ) {
    for (int j = i; j < n - 1; j ++)
        a[j] = a[j+1];
    n = n-1;
}

public final void set( int i, String x ) {
    a[i] = x;
}

public final int size() {
    return n;
}

private static void print(Object[] oa) {
    for (int i=0; i<oa.length; i++) {
        System.out.print(oa[i] + " ");
    }
    System.out.println("");
}

public void print( PrintWriter output, int width, int
precision
) {

    DecimalFormat fmt = new DecimalFormat();
    fmt.setDecimalFormatSymbols( new
    DecimalFormatSymbols(Locale.US) );
    fmt.setMinimumIntegerDigits( 1 );
    fmt.setMaximumFractionDigits( precision );
    fmt.setMinimumFractionDigits( precision );
    fmt.setGroupingUsed( false );
    print( output, fmt, width );
}

public void print( PrintWriter output, NumberFormat format ,
int width) {
    for ( int i=0; i<n; i++ ) {
        String str = format.format( get(i) );
        int space = width - str.length();
        output.print( ' ' );
        space--;
        for ( ; space>0; space-- )
            output.print( ' ' );
        output.print( str );
    }
```

```
        output.println();
    }

    public void stringSort() throws PfordException {

        try {
            Arrays.sort(a);
            // return a;

            //for (int i=0;i<a.length;i++)
            //System.out.println(a[i]);
        }
        catch( Exception e)

        {
            throw new PfordException("no elements to sort in string array");
        }
    } // end of method stringSort

    public PfStringArray stringArraySort() {
        PfStringArray x = new PfStringArray( n );
        x.stringSort();
        return x;
    }

}
```

10.1.23 PfVector.java

```
/** insert type description here
 *
 * Creation Date: (2004-11-20 17:20:10)
 * a@author: shringika,deepti
 */

import java.io.PrintWriter;
import java.io.RandomAccessFile;
import java.io.IOException;
import java.text.NumberFormat;
import java.text.DecimalFormat;
import java.text.DecimalFormatSymbols;
import java.util.*;
import java.lang.Math;

public class PfVector implements Cloneable {

    double [] a;

    int m,q, n, r, ms, ns;

    public PfVector( int n ) {
        this.n = n;
        a = new double[n];
    }

    public PfVector( PfVector vect ) {
        n = vect.n;
        a = vect.a;
    }

    public PfVector( int m, int q ) {
        this.m = m;
        this.q = q;
        r = 0;
        ms = 1;     // the matrix is saved column-by-column
        ns = m;
        a = new double[m*q];
    }

    public PfVector( int m, int q, double s ) {
        this.m = m;
        this.n = q;
        r = 0;
        ms = 1;
        ns = m;
        a = new double[m*q];
        for ( int i=0; i<a.length; i++ )
            a[i] = s;
    }


    public PfVector assign( PfVector b ) {
        if ( n != b.n )
```

```
            throw new IllegalArgumentException( dimErrMsg( b )
            );
        for ( int i=0; i<n; i++ )
            set( i, b.get( i ) );//  Replace the element at ith position in a
with the specified element in b.
        return this;
    }

    public Object clone() {
        return copy();
    }

    public boolean contains( double x ) {
        for ( int i=0; i<n; i++ )
            if ( get(i) == x )
                return true;
        return false;
    }

    public final PfVector copy() {
        PfVector x = new PfVector(n );
        for ( int i=0; i<n; i++ ) {
            x.a[i] = a[i];
        }
        return x;
    }

    // this method returns a string if there is an error in demensions for
array

    private final String dimErrMsg() {
        return "dimensions not proper: " + n;
    }

    private final String dimErrMsg( PfVector b ) {
        return "Dimensions do not match: " + n + " <=> "
        + b.n;
    }

    public double elementAt(int i) {

        return a[i];
    }

    public boolean eq( PfVector b ) {
        if ( n != b.n )
            throw new IllegalArgumentException( dimErrMsg( b )
            );

        for ( int i=0; i<n; i++ )
            if ( !(get(i) == b.get(i)) )
                return false;
        return true;
    }

    public double firstElement() {
        return a[0];
```

```
    }

    public final double get( int i) {
        return a[i];
    }

    public boolean isEmpty() {
        if (n == 0)
            return true;
        return false;
    }

    public double lastElement() {
        return a[n-1];
    }

    public double max() {
        double d = Double.MIN_VALUE;
        for ( int i=0; i<n; i++ )
            if ( get(i) > d )
                d = get(i);
        return d;
    }


    public double min() {
        double d = Double.MAX_VALUE; //const holdingthe largest finite value
of type double
        for ( int i=0; i<n; i++ )
        {  if ( get(i) < d )
               d = get(i);
        }
        return d;
    }

    public boolean neq( PfVector b ) {
        if ( n != b.n )
            throw new IllegalArgumentException( dimErrMsg( b )
            );
        for ( int i=0; i<n; i++ )
            if ( get(i) != b.get(i) )
                return true;
        return false;
    }

    public final void removeElementAt( int i ) {
        for (int j = i; j < n - 1; j ++)
            a[j] = a[j+1];
        n = n-1;
    }


    public PfVector selfadd( PfVector b ) {
        if ( n != b.n )
            throw new IllegalArgumentException( dimErrMsg( b )
            );
        for ( int i=0; i<n; i++ )
```

```
        set( i , get(i) + b.get(i) );
    return this;
}



public PfVector selfmul( double f ) {
    for ( int i=0; i<n; i++ )
        setMul( i, f );
    return this;
}

public PfVector selfneg() {
    for ( int i=0; i<n; i++ )
        set( i, -get(i) );
    return this;
}

public PfVector selfsub( PfVector b ) {
    if ( n != b.n )
        throw new IllegalArgumentException( dimErrMsg( b )
        );
    for ( int i=0; i<n; i++ )
        setSub( i , b.get(i) );
    return this;
}

public final void set( int i, double x ) {
    a[i] = x;
}


public final void setAdd( int i, double x ) {
    a[i] += x;
}

public final void setMul( int i, double x ) {
    a[i] *= x;
}
public final void setSub( int i,double x ) {
    a[i] -= x;
}
public final int size() {
    return n;
}

public PfVector times( double s ) {

    PfVector x = new PfVector( n );
    for ( int i=0; i<n; i++ )
        x.set( i, get(i) * s );
    return x;
}

public PfVector uminus() {

    PfVector x = new PfVector( n );
```

```
    for ( int i=0; i<n; i++ )
        x.set( i, -get(i) );
    return x;
}

public void print( PrintWriter output, int width, int
precision
) {

    DecimalFormat fmt = new DecimalFormat();
    fmt.setDecimalFormatSymbols( new
    DecimalFormatSymbols(Locale.US) );
    fmt.setMinimumIntegerDigits( 1 );
    fmt.setMaximumFractionDigits( precision );
    fmt.setMinimumFractionDigits( precision );
    fmt.setGroupingUsed( false );
    print( output, fmt, width );
}

public void print( PrintWriter output, NumberFormat format ,
int width) {
    for ( int i=0; i<n; i++ ) {
        String str = format.format( get(i) );
        int space = width - str.length();
        output.print( ' ' );
        space--;
        for ( ; space>0; space-- )
            output.print( ' ' );
        output.print( str );
    }

    output.println();
}

public PfVector div( double f ) {
    PfVector x = new PfVector( n );
    for ( int i=0; i<n; i++ )
        x.set( i, get(i) / f );
    return x;
}

public void swap(int i, int j) {
    double temp = a[i];
    a[i] = a[j];
    a[j] = temp;
}

public void selectionSort() {
    int i = 0, j = 0, minLocation = 0;

    //  repeatedly locate correct value for position i
    for (i = 0 ; i < n - 1 ; i++) {
        minLocation = i;
        // check elements from i to end for smallest
        for (j = i + 1 ; j < n; j++) {
            if (a[j] < a[minLocation]) {
                // record index of smallest value
```

```
                minLocation = j;
            } // end of if
        } // end of "for" loop for  j
        // swap smallest of remaining values into i<sup>th</sup> position
        swap(minLocation, i);
    } // end of "for" loop for i

} // end of method selectionSort

public PfVector arraySelectionSort() {
    PfVector x = new PfVector( n );
    x.selectionSort();
    return x;
}

public PfVector arrayQuickSort() {

    PfVector x = new PfVector( n );
    //Arrays.sort(x);
    x.sort(0, a.length - 1);
    return x;
}

private void sort(int low, int high) {
    if (low >= high) return;
    int p = partition(low, high);
    sort(low, p);
    sort(p + 1, high);
}

private int partition(int low, int high) {
    double pivot = a[low]; //1st element
    int i = low - 1;
    int j = high + 1;
    while (i < j) {
        i++; while (a[i] < pivot) i++;
        j--; while (a[j] > pivot) j--;
        if (i < j) swap(i, j);
    }
    return j;
}

public double mean() {
    double mean = 0.0;
    for ( int i=0; i<n; i++ )
        mean = mean + get(i);
    return (double) mean/n;
}

public double sumArray() {
    double d = 0.0;
    for ( int i=0; i<n; i++ )
        d = d + get(i);
    return d;
}

public double stddev() {
```

```
        PfVector x = new PfVector( n );

        double xbar = mean();

        double xxbar = 0.0;
        for ( int i=0; i<n; i++ ) {
            xxbar += (get(i) - xbar)*(get(i) - xbar) ;
        }
        double variance=xxbar/(n-1);
        return Math.sqrt(variance);
    }

    public int Arraylength() {
        return n;
    }
}
```

10.1.24 Range.java

```
/**
 *insert type's descritpion here.
 *Creation Date: (2004-11-08 09:20:10)
 *author: deepti
 */
public class Range implements Cloneable {
    private int number;
    private int base;
    private int stride;

    public Range( int first, int last ) {
        this.base = first;
        number = last - base + 1;
        if ( number < 0 ) {
            number = -number;
            stride = -1;
        }
        else
            stride = 1;
    }

    public Range( int first, int size, int interval ) {
        this.base = first;
        this.number = size;
        this.stride = interval;
    }

    public final int first() {
        return base;
    }


    public final int next( int n ) {
        return n + stride;
    }

    public final boolean contain( int n ) {
        n -= base;
        if ( stride >= 0 )
            return n >= 0 && n < number * stride && 0 == n % stride;
        return n <= 0 && n > number * stride && 0 == (-n) % (-stride);
    }

}
```

10.1.25 Symbol.java

```
/**
 *insert type's descritpion here.
 *Creation Date: (2004-11-08 09:20:10)
 *author: deepti
 */
import java.util.*;
import java.io.PrintWriter;

class Symbol extends HashMap {
    Symbol v_parent;
    boolean read_only;
    public Symbol( Symbol parent) {
        v_parent = parent;
        read_only = false;
    }
    public void setReadOnly() {
        read_only = true;
    }

    //return the parent for the symbol
    public final Symbol Parent() {
        return v_parent;
    }

    //find variable name in the hash table
    public final boolean containsVar( String name ) {
        return containsKey( name );
    }

    //look for the global or local reference of the variable
    private final Symbol gloabalOrLocal( int level, boolean is_static ) {
        Symbol st = this;
        if ( level < 0 ) {
            // global variable
            while ( null != st.v_parent )
                st = st.Parent();
        }
        else {
            // local variable
            for ( int i=level; i>0; i-- )

            {
                while ( st.read_only ) {
                    st = st.Parent();
                    //assert st != null;
                }
                if ( null != st.Parent() )
                    st = st.Parent();
                else
                    break;
            }
        }
        return st;
```

```
    }

    //return the value of the variable
    public final PfordDataType getValue( String name, boolean is_static, int
level ) {
        Symbol st = gloabalOrLocal( level, is_static );
        Object x = st.get( name );
        while ( null == x && null != st.Parent() ) {
            st = st.Parent();
            x = st.get( name );
        }
        return (PfordDataType) x;
    }
    public final void setValue( String name, PfordDataType data, boolean
is_static, int level ) {
        Symbol st = gloabalOrLocal( level, is_static );
        while ( st.read_only ) {
            st = st.Parent();
            //assert st != null;
        }
        st.put( name, data );
    }
}
```

10.2.1 testapplic.pford

```
/*********************
@author: Bhagyashree
************/
startprogram

/*************
*our functions are applicative order
********/

w = 3;
x = 10;
func incw()
{
      w=w+1;
      return w;
};
func incx()
{
      x=x+1;
      return x ;
}
;
func myfoo(y, z)
{
   output y+y;
   x = 1;
   output z;
}
;

myfoo(incw(), incx());

endprogram
```

10.2.2 testarray.pford

```
/*********************
@author: Shringika
************/

startprogram
sequence=[90,20,30,40];
output("sequence is");
output(sequence);
output("3rd element in sequence is");
output(sequence[2]);
output("average of sequence is");
avgseq=mean(sequence);
output(avgseq);
output("minimum in sequence is");
minimum=min(sequence);
r=range(sequence);
output(minimum);
output("range is");
output(r);
sizearray=size(sequence);
output("size of array sequence is");
output(sizearray);


output ("result after multiplying with 10");
seq10times=sequence*10;
output(seq10times);

endprogram
```

10.2.3 testassoc.pford

```
/*********************
@author: Bhagyashree
************/
startprogram
output "test assoc.: 2+3=5";
a=2+3;
output a;

output "test assoc.: 2+3*4=14";
a=2+3*4;
output a;

output "test assoc.: 2+3*-4=-10";
a=2+3*-4;
output a;

output "test assoc.: 2*2+3*4=16";
a=2*2+3*4;
output a;

output "test assoc.: 2/2+1=2";
a=2/2+1;
output a;

output "test assoc.:-2/2+1=0";
a=-2/2+1;
output a;

output "test assoc.:2^3=8";
a=2^3;
output a;

output "test assoc.:-2^3=-8";
a=-2^3;
output a;


output "test assoc.:9.0^0.5=3";
a=9.0^0.5;
output a;

output "test assoc.:9.0^-0.5=0.33";
a=9.0^-0.5;
output a;

output "test assoc.:2.2^3.3=13.76";
a=-2.2^3.3;
output a;

output "test assoc.: 2.0+3.11=5.11";
a=2.0+3.11;
output a;

output "test assoc.: 2.1+3.5*4.5=17.85";
a=2.1+3.5*4.5;
```

```
output a;

output "test assoc.: 2+3.333*-4.0=-11.32";
a=2+3.33*-4.0;
output a;

output "test assoc.: (2*(2+3))+-8)*(4+5.5/1.1-(9-6))=12";
a=(2*(2+3)+-8)*(4+5.5/1.1-(9-6));
output a;

output "test assoc.: 2.44/-2+1=-0.22";
a=2.44/-2+1;
output a;

output "test assoc.:(((2+3)-(4+5)))=-4";
a=(((2+3)-(4+5)));
output a;

endprogram
```

10.2.4 testbool.pford

```
/*********************
@author: Bhagyashree
************/
startprogram

a = true;
b = false;
c = a OR b;
output c;

output (a OR b);
output true AND false;
output true AND true;
output false AND false;
output false AND true;

c= a AND b;
output c;

x=4;

output (x==4);
output (x>4);

endprogram
```

10.2.5 testbuiltin.pford

```
/*********************
@author: Bhagyashree
************/
startprogram

a = "Bhagyashree:";
b= "Deepti";
c="Shringika";
m="shree";

/***Builtin function "join"
*which joins 2 strings
****/

d= join(a,c);
output d;

output join(d,b);

/***builtin function to change to
*upper case adn lower case
***********/

output toUpper(d);

b = toUpper(b);
output b;

output toLower(b);
output toUpper(toLower(toUpper("Bhagyashree")));

/***builtin function for string length
***********/
output stringLength("Bhagyashree");
output stringLength("Bhagyashree"+b);

/***builtin function to test if string
contains given string
***********/
a="Bhagyashree";
if(contains(a,m))then
{
      output "you got it";
};

endprogram
```

10.2.6 testcomment.pford

```
/**********************
@author: Bhagyashree
************/
startprogram

/********
multiline
********/

output "Multiline comment worked";

//singleline//singleline
output "Single line comment worked";

endprogram
```

10.2.7 testif.pford

```
/*********************
@author: Bhagyashree
************/
startprogram

if (true) then
{
      output "inside first if";
};

if (false) then
{
      output " hi";
}
elsedo
{
      output "am in else now";
}
;

a = 2;
if(a>1) then
{
      output "a is 2 now";
      output "a is 2 now";
      output "a is 2 now";

}
elsedo
{
      output"oops";
      output"oops";
      output"oops";

}
;

b=a+1;

if(b<1) then
{
      if(b<10) then
      {
            output "b is 3 now";
      };

}
elsedo
{
      b=6;
      if(b==6) then
      {
            output"b has internal value";
            output b;
```

```
	}
	elsedo
	{
		output "b has value";
		output b;
	};
}
;
endprogram
```

10.2.8 testinput.pford

```
/*********************
@author: Deepti
************/
startprogram

a= 5;
input a;
output a;

endprogram
```

10.2.9 testloop.pford

```
/*********************
@author: Bhagyashree
************/
startprogram

loop 5 runs
{
      output "what's up??";
}
;

a = 1;
output "intial value of a";
output a;

loop 10 runs
{
      a = a+1;
}
;

output"value of a after looping 10 times is";
output a;

b=0;
output "intial value of b";
output b;
loop 10 runs
{
      loop 5 runs
      {

            b = b+1;
            //break_loop;
      };
}
;
output"value of b after looping 5 times nested in looping 10 times is:";
output b;


//output"value of c after looping 5 times nested in looping 10 times with
break at 16 is:";
//output c;
/*b=3;
output "value of b before while is:";
output b;

while(b==3) loopthrough
{
      b=b-1;
      output b;
}
;
```

```
output"value of b after while-loopthrough is:";
output b;
*/


d = 1;
output "intial value of d";
output d;

loopfrom 1 to 10 interval 2
{
      d = d*2;
}
;
output"value of d after loopfrom 1 to 10 interval 2 is:";
output d;

e=1;
output "value of e before loopfrom nested is:";
output e;

loopfrom 1 to 10 interval 2
{
      e = e*2;
      loopfrom 1 to 2 interval 1
      {
            e=e/2;
            loop 1 runs
            {
                  e=e*2;
            };
      };
}
;
output"value of e after loopfrom 1 to 10 interval 2 with nested loopfrom 1 to
2, inrval 1 with loop 1 runs :";
output d;

endprogram
```

10.2.10 testobject.pford

```
startprogram
/**************************************
*@Author: Deepti
**************************************/
object a{
i= 3;
j= 10;
};

output a.i;


object a{
i= 3;
j= 10;
output "deepti";
};

output a.i;

object a{
i= 3;
j= "elice";
};

output a.j;

endprogram
```

10.2.11 testorder.pford

```
/*********************
@author: Bhagyashree
************/
startprogram
w=3;
x=10;
func incw()
{
      w=w+1;
      return w;
}
;
func incx()
{
      x=x+1;
      return x;
};

func myfoo(y,z)
{
      output y+y;
      x=1;
      output z;
}
;

myfoo(incw(),incx());
endprogram
```

10.2.12 testoutput.pford

```
/*********************
@author: Bhagyashree
************/
startprogram

/********
multiline
********/

output "Hello World";
a=2;
b=3;
output a+b;
//singleline

endprogram
```

10.2.13 testscope.pford

```
/*********************
@author: Bhagyashree
************/
startprogram
a=1;
b=2;

if(true)then
{
      a=2;
      output a;
};
output a;

if(true)then
{
      output b;
};

func foo(x)
{
      a=10;
      q=34;
      output a;
}
;

func bar()
{
      output a;
};

foo(b);
output a;
output q;
a=10;
bar();

/****pford has dynamic scoping***/
func lambda(x)
{
      x=5;
      func gimmex(x)
      {
            lambda(x);
      };

      x=10;
      gimmex(x);
};

/***pford has deep binding**/
func a(i,p)
{
```

```
	func b()
	{
		output i;
	};
	if(i==1)then
	{
		a(2,b());
	}
	elsedo
	{
		q();
	};
};
func q()
{
 dummy =1;
};
a(1,q());


endprogram
```

10.2.14 teststring.pford

```
/*********************
@author: Bhagyashree
************/

startprogram

a = "Bhagyashree ";
output a;
b= "Deepti ";
output a+b;
c= a+b+"Shringika ";
output c;
output "Bhagyashree "+"Bohra ";
output "Bhagyashree"+b;

loop 5 runs
{
      a = a+b+c;
}
;
output "result of looping a+b+c 5 times:";
output a;

endprogram
```

10.2.15 testuserfunc.pford

```
/*********************
@author: Bhagyashree
************/


startprogram

a = 1;
output "oustide a";
output a;

func foo(a)
{
      //a=2;
      output "foo a ";
      output a;
      return a;
}
;

foo(a+10);
foo(3);
foo("Bhagyashree");
foo(a==2);

func addmynums(a,b,d)
{
      //e=foo(a);
      c=a+b+d;
      return c;
}
;
func mymy(a,b,d)
{
      func mymymy(m)
      {
            output m;
      };
      mymymy(3);
      output "mymymy";
      a+=1;
      mymymy(a);
}
;

l=3;
m=4;
n=10.5;
r=addmynums(l,m,n);
output r;
mymy(l,m,n);
/*************
*our functions are applicative order
********/
```

```
w = 3;
x = 10;
func incw()
{
      w=w+1;
      return w;
};
func incx()
{
      x=x+1;
      return x ;
}
;
func myfoo(y, z)
{
   output y+y;
   x = 1;
   output z;
}
;

myfoo(incw(), incx());

a = 2.0 ^ -3.0;
output a;

b = 2 ^ -3;
output b;
endprogram
```

10.2.16 testfunc2.pford

```
startprogram
/**************************************
*@Author: Shringika, Deepti,Bhagyashree
*Test file for pford functions
*version 1.1
*Date: 12/04/2004, 12.28pm
**************************************/
x = 10;
y = 12;
z= 10.0;
w= 12.5;


func func1()
{
      output("lazy,doing nothing");
};


func adderInt(a, b)
{
      b = a;
      c = b;

      return b+c;
};


func adderDouble(a,b)
{
      b = a;
      d = b;

      return b+d;
};


func mystring(s)
{
      return s;
};

func nestedfunc()
{
      a=0;
      b =2;
      c = 3;

      a = adderInt(b,c);
      d="PFORD language";
      output (mystring(d));

};
```

```
output("calling func1");
func1();

output("calling adderInt");
output(adderInt(x,y));


output("calling adderDouble");
output(adderDouble(z,w));

output("calling nestedfunc");
nestedfunc();

endprogram
```

10.2.17 testfinal.pford

```
startprogram

/**************************************
*@Author: Deepti
**************************************/
/*
test different things in one file
*/

/*single line*/


foo = 3 + 4;

output "Hello world!";

a = 1;
if (1 > 2) then
{
      a= 3;
}
elsedo
{
  b = 3 + 4 * 5 + 6 * 7;
};

output "b should be 65";
output b;

a += 5;


b= 3 * 4 + 5 * 6;


c= b + a;
output "c is 48";
output c;

b = 0;
loop 11 runs
{
b = b+1;

};

loopfrom 12 to 15 interval 1 { b = b + 1;};

output "b shoud be 14";
output b;

func increment(a, c, b)
{
b = a+ c;
return b;
```

```
};

d= increment(1,1,4);
output "value of d is 2";
output d;

endprogram
```

10.2.18 testbool2.pford

```
startprogram
/***************************************
*@AuthOR: Shringika, Deepti,Bhagyashree
*Test file fOR pfORd boolean
*version 1.1
*Date: 12/04/2004, 12.58pm
***************************************/

func NOTtrue()
{
      output(NOT true);
};

func NOTfalse()
{
      output(NOT false);
};

func NOTtruevar()
{
        a = true;
        output(NOT a);
};

func NOTfalsevar()
{
        a = false;
        output(NOT a);
};

func trueANDtrue()
{
      output(true AND true);
};

func trueANDfalse()
{
      output(true AND false);
};

func falseANDtrue()
{
      output(false AND true);
};

func falseANDfalse()
{
      output(false AND false) ;
};

func trueORtrue()
{
      output(true OR true);
};
```

```
func trueORfalse()
{
      output(true OR false);
};

func falseORtrue()
{
      output(false OR true);
};

func falseORfalse()
{
      output(false OR false);
};

func mixedtrue()
{
      output(true OR true AND false);
};

func mixedfalse()
{
      output((false OR true) AND false);
};

func mixedfalse1()
{
      output(true AND false OR true AND false);
};

func mixedtrue1()
{
      output(true AND NOT false AND ( NOT true OR true OR false ));
};

func eqtffalse()
{
      output(true==false);
};

func eqfftrue()
{
      output(false==false);
};

func eqtttrue()
{
      output(true==true);
};

func eqftfalse()
{
      output(false == true);
};

func neqtftrue()
```

```
{
      output(true!=false);
};

func neqfffalse()
{
      output(false!=false);
};

func neqttfalse()
{
      output(true!=true);
};

func neqfttrue()
{
      output(false != true);
};

func vartrue()
{
        a = true;
        b = NOT a;
        c = a AND b;
        d = a OR b;
        output(a OR b AND c OR d);
};

func varfalse()
{
        a = true;
        b = NOT a;
        c = a AND b;
        d = a OR b;
        output(a AND b OR c AND d);
};

output("testing NOTtrue");
NOTtrue();
output("testing NOTfalse");
NOTfalse();
output("testing NOTtruevar");
NOTtruevar();
output("testing NOTfalsevar");
NOTfalsevar();

output("testing trueANDtrue");
trueANDtrue();
output("testing trueANDfalse");
trueANDfalse();
output("testing falseANDtrue");
falseANDtrue();
output("testing falseANDfalse");
falseANDfalse();

output("testing trueORtrue");
trueORtrue();
```

```
output("testing trueORfalse");
trueORfalse();
output("testing falseORtrue");
falseORtrue();
output("testing falseORfalse");
falseORfalse();

output("testing mixedtrue");
mixedtrue();
output("testing mixedfalse");
mixedfalse();
output("testing mixedfalse1");
mixedfalse1();
output("testing mixedtrue1");
mixedtrue1();

output("testing eqftfalse");
eqftfalse();
output("testing eqtttrue");
eqtttrue();
output("testing eqtffalse");
eqtffalse();
output("testing eqfftrue");
eqfftrue();

output("testing neqfttrue");
neqfttrue();
output("testing neqttfalse");
neqttfalse();
output("testing neqtftrue");
neqtftrue();
output("testing neqfffalse");
neqfffalse();

output("testing vartrue");
vartrue();
output("testing varfalse");
varfalse();

endprogram
```

10.2.19 testiop.pford

```
startprogram
/***************************************
*@Author: Shringika, Deepti,Bhagyashree
*Test file for pford string
*@version 1.1
*Date:12/04/2004, 12.06pm
****************************************/

func add8()
{
      output(4.0+4);
};

func add6()
{
      output(+4 + +2.0);
};

func sub0()
{
      output(2.0-2);
};

func sub4()
{
      output(2 - -2.0);
};

func subneg2()
{
      output(-2.0 - -2.0);
};

func mult10()
{
      output(2.0*5);
};

func multneg12()
{
      output(3*-4.0);
};

func div50()
{
      output(100/2);
};

func divneg50()
{
      output(100/-2);
};

func div4dot5()
```

```
{
      output(9.0/2);
};

func divneg4dot5()
{
      output(-9.0/2);
};

func mod400()
{
      output(9%2.0);
};

func geqgtrue()
{
      output(5>=3.0);
};

func geqeqtrue()
{
      output(3.0>=3.0);
};

func geqfalse()
{
      output(3.0>=5);
};

func leqltrue()
{
      output(3<=5.0);
};

func leqeqtrue()
{
      output(3.0<=3);
};

func leqfalse()
{
      output(5<=3.0);
};

func gttrue()
{
      output(5>3.0);
};

func gtfalse1()
{
      output(3.0>3);
};

func gtfalse2()
{
      output(3>5.0);
```

```
};


func lttrue()
{
      output(3.0<5);
};

func ltfalse1()
{
      output(3.0<3);
};

func ltfalse2()
{
      output(5<3.0);
};

func eqfalse1()
{
      output(5.0==3);
};

func eqtrue()
{
      output(3==3.0);
};

func eqfalse2()
{
      output(3.0==5);
};

func neqtrue1()
{
      output(3!=5.0);
};

func neqfalse()
{
      output(3!=3.0);
};

func neqtrue2()
{
      output(5.0!=3);
};

output("testing add8");
add8();
output("testing add6");
add6();

output("testing sub0");
sub0();
output("testing sub4");
sub4();
```

```
output("testing subneg2");
subneg2();

output("testing mult10");
mult10();
output("testing multneg12");
multneg12();

output("testing div4dot5");
div4dot5();
output("testing divneg4dot5");
divneg4dot5();
output("testing mod400");
mod400();

output("testing div50");
div50();
output("testing divneg50");
divneg50();

output("testing geqgtrue");
geqgtrue();
output("testing geqeqtrue");
geqeqtrue();
output("testing geqfalse");
geqfalse();

output("testing leqltrue");
leqltrue();
output("testing leqeqtrue");
leqeqtrue();
output("testing leqfalse");
leqfalse();

output("testing gttrue");
gttrue();
output("testing gtfalse1");
gtfalse1();
output("testing gtfalse2");
gtfalse2();


output("testing lttrue");
lttrue();
output("testing ltfalse1");
ltfalse1();
output("testing ltfalse2");
ltfalse2();

output("testing eqtrue");
eqtrue();
output("testing eqfalse1");
eqfalse1();
output("testing eqfalse2");
eqfalse2();

output("testing neqfalse");
```

```
neqfalse();
output("testing neqtrue1");
neqtrue1();
output("testing neqtrue2");
neqtrue2();

endprogram
```

10.2.20 testif2.pford

```
startprogram
/**************************************
*@Author: Shringika, Deepti,Bhagyashree
*Test file for pford if statement
*version 1.1
*Date: 04-Dec-2004 11.15 am
**************************************/


func oneTestIf()
 {
 a = 1;
 b = 2;
if (true) then
{
      a = 1;
}
elsedo
{
      a = 2;
};
output a;

};



func twoTestIf()
{

a = 1;
b = 2;
if ((a == 1) AND (b == 2)) then
{
      c = 1;
}
elsedo
{
      c = 2;
};
output c;
};


func ifAnd()
 {
a = 1;
b = 2;
if ((a == 2) AND (b == 2)) then
{
      c = 2;
}
elsedo
{
      c = 1;
```

```
};
output c;
};

func ifOr()
{
a = 1;
b = 2;
if ((a == 1) OR (b == 2)) then{
      c = 1;
}
elsedo
{
c = 2;
};
output c;
};

func ifLt()
{
a = 1;
      b = 2;
if (a < b)then
{
      c = 1;
};
output c;
};

func ifGt()
 {
a = 2;
      b = 1;
if (a > b) then
{
      c = 1;
};
output c;

};

func ifLtEq()
{
a = 1;
      b = 2;
if (a <= b) then
{
      c = 1;
};
output c;
};

func ifGtEq()
 {
a = 2;
      b = 1;
if (a >= b) then
```

```
{
      c = 1;
};
output c;
};

func ifNeq()
{
a = 1;
b = 2;
c=5;
if (a != b) then
{
      c = 1;
};
output c;
};


oneTestIf();
twoTestIf();
ifOr();
ifAnd();
ifGtEq();
ifLtEq();
ifLt();
ifGt();
ifNeq();

endprogram
```

10.2.21 testunary.pford

```
startprogram
/**************************************
*@Author: Shringika, Deepti,Bhagyashree
*Test file for pford unary operators
*version 1.1
*Date: 04-Dec-2004 11.20 am
**************************************/


func plusEq()
 {
 a = 1;
a += 1;
output a;
};


func minusEq()
 { a = 1;
a -= 1;
output a;
};


func multEq()
 { a = 9;
a *= 2;
output a;
};

func divEq()
{
      a = 9;
      a /= 2;
      output a;
};


func doubleMeq()
{
 a = 9.0;
a *= 2;
output a;
};


func doubleDeq ()
{
 a = 9.0;
a /= 2;
output a;
};

plusEq();
minusEq();
```

```
multEq();
divEq();
doubleMeq();
doubleDeq();


endprogram
```

PFORD **Future:**

Future

In the words of Alfred North Whitehead, “Civilization advances by extending the number of important operations which we can perform without thinking.” The whole philosophy behind PFORD adds a little to this maxim. We believe that one should be able to do the maximum with the minimum amount of coding.

To facilitate this, future versions of PFORD will support all languages instead of just C and Java. Ideally we would like to be supporting Perl, Python, Awk, etc….

The power of PFORD. The following snippet prints out common values between two files and also values found in one and not the other- all in just 6 lines.

```
PERL:
Set A= read_file (“file.txt”, “|”, 0, “\n”);
Set B= read_file (“file1.txt”, “|”, 0,“\n”);
Set common= A and B;
Set diff_A= A-common;
PFORD(future version):
Output “Elements common to both file and file1 are \n”, common;
Output “Elements in file that are not in file1 are \n”, diff_A;
```

Another future development is PFORD will provide some highly easy to use and optimized libraries for creating complex data structures and performing string manipulations. These features are not supported by C and Java’s version is not very intuitive. Hence, we have incorporated simple to use data structures, namely arrays, hashes and sets which have been inspired by the PERL programming language. Some elements include stacks, queues and binary tree functions.

PFORD **Summary:**

Summary

PFORD is quick, simple and easy to learn. It provides a powerful edge over the other programming languages because it enables the novice to learn programming effortlessly as well as provide advanced programmers the power to write code once and switch to C++/Java easily. It gives the users 3 languages at their disposal, and in future it will give them all languages and abstract complicated code as single word functions.

PFORD **References:**

- Some of our test cases were developed using the professor example files to test associativity, scoping and binding rules.
  http://www1.cs.columbia.edu/~sedwards/classes/2004/w4115-fall/index.html

- Earlier class projects posted at class website
  http://www1.cs.columbia.edu/~sedwards/classes/2004/w4115-fall/project.html

- Anltr reference
  www.antlr.org

- Mx code was very useful in helping with code development
  http://www1.cs.columbia.edu/~sedwards/classes/2003/w4115/Mx.final.pdf

- definitions
  http://dictionary.reference.com/search?q=lexical%20scope