THE FILE JOURNALING LANGUAGE (FJL)

COMS W4115
Programming Languages and Translators
Summer 2007

John Petrella
jep2124@columbia.edu

Table of Contents

Introduction: The Whitepaper

In today's day and age the cost of hard drives are nominal and has consequently created the notion of unlimited file storage. Due to this trend, remembering a file's location in a directory structure or keeping track of which files are living on a system has become a chore. The use of updatedb and locate, has become cumbersome returning a long listing of files, most of which are typically useless. FJL was invented as a way to keep track of the files a user owns. It is eloquent and useful, providing tailored file views consisting of file type groupings and sorting mechanisms to effortlessly customize a hierarchical view. Organizing a list of your own files has never been so easy and simply presented.

Goal:

With file storage becoming seemingly endless, the necessity to clean up and delete less frequently used files has diminished. Files are now kept forever and their location easily forgotten. Currently, there is no programming language designed to help ease the burden of remembering file location or organizing files into a personalized representation. This has resulted in enormous amounts of wasted time fiddling with specialized ls, grep, and find commands.

The main goal in the design of FJL, is to provide an easy to use programming language that focuses on filesystem information and provide a handy programming notion to the user. A simple, useful organization tool for users concerned with keeping track of files living throughout their filesystem hierarchy. The FJL programming language will provide mechanisms to view you filesystem hierarchy in a personalized manner. Grouping of files and sorting through files by type or name has never been easier.
Providing a way to customize the file view of your system was paramount in the design.

More on FJL's Features:

Building FJL meant providing data types and mechanisms to make filesystem information readily and simply available to a programmer. FJL developers envisioned that a mere function call should read in a permissible directory or recursed set of directories. The FJL language supplies this mechanism in the standard library. The standard library provides extremely useful directed functions. The data types in the programming language were carefully crafted to supply the programmer with the all the file information he/she could ever have use for.

Portability:

FJL is an interpreted language, that was developed to be portable to all versions of Linux and Unix.

Details:

FJL provides several functions to enable easy access to the filesystem. A call to readdir given a directory will read in the directory specified and assign it to our dir type. Array mechanisms, have been provided to access file elements in dir types. FJL has also crafted several language specific data types. Some of the most useful being the file and dir data types. The file data type has attributes "name", "time" and "type" that provide an enormously simple way of accessing details of files. While the dir data type serves as a container for file types, it has it's own set attributes "nof" (number of files), "lt" (last touched), and "name" to give directory specific information. FJL also provides flow control with the usage of while loops and if statements, which provide the looping feature needed to access individual file attributes. Comparison operators are also made available. Examples: A brief example of the simplicity of the code is shown below. The example below shows how easy it is read in the current directory. The example also groups by a particular file type that has size over a certain threshold.

Language Tutorial

    FJL was primarily designed to interface with the filesystem more easily than ever before. Paramount in the design of FJL was the idea that simple calls could be used to access files and there associated attributes.
    To start off we will look at a simple FJL example.

```
|* A Simple Example *|

int j;
int i;
fsatt word;

word = "Assign this String";

println word;

i=5;
j=6;
i = i * j;

print "This is that value of i ";
println i;
exit;
```

The above example is a very basic one. It details how to declare variables. It also assigns a mathematical expression to an int and assigns a string to fsatt type. The variables are then printed to standard out.
The output is as follows:

```
Assign this String
This is that value of i 30
```

To get a better feel of the language's capabilities, Lets look a little further at a more intermediate example.


```
|* An Intermediate Example: *|
int i;
dir first;
file foo;

first = readdir("/home/jp/school");

while ( i < first.nof ){
foo = first[i];
print "The file name is ";
println foo;
i = i + 1;
```

```
}
exit ;
```

The FJL program above first starts off by declaring some variable types. Then using the built in function readdir reads files a given directory. The value of which is stored as a dir type. The dir type is than walked by the while loop, assigning each file to 'foo'. The filename is than printed out, with the program exiting when there are no more files held in the dir type.
The output is as follows:

```
The file name is plt
The file name is whim
The file name is temp
```

Moving on to one of the more interesting examples shows will print all the files that do not have a'.txt' file ending. It will also sum up those that do and do not have the file name ending.

```
|* A More Complicated Example: *|

int did;
int didnot;
file first;
dir second;
int i;
fsatt directory;

directory="/home/jp/school/temp";
second = readdir(directory);

while (i<second.nof) {
        first = second[i];
        if ( !(first.name -- /[.]txt$/) ){
           println first.name;
           didnot = didnot + 1;
        } else {
           did = did + 1;
        }
        i = i + 1;

}

print did;
println " Did match .txt ";
print didnot;
```

```
println " Did not match .txt";
```

The FJL program above starts off by defining some variable types. It then reads in a directory assigned to the fsatt type directory. The program continues to loop over all the files in the directory and print out the names of the files that do not have '.txt' at the end of the file using the match operator. It keeps a count of those that match and do not match and prints out the results. The contents of "/home/jp/school/temp" are { sample1.txt, sample2.txt, sample3.txt, second }.
The output is as follows:

```
second
3 Did match .txt
1 Did not match .txt
```

Language Reference Manual _____

1. Lexical Conventions:

1.1 Comments:

FJL supports only multiple line comments. Multiple line comments begin with '|*' and continue to '*|' is found. Any text living inside the comment markers are ignored by the parser.

      |* This is a comment *|
      |* Comments can span
Multiple lines and have tabs or other whitespace inside of the comment tags. *|

1.2 Identifiers:

An identifier in FJL must start with a letter. The following sequence of characters, if any, must be a letter, digit, or underscore. Identifiers are case sensitive.

1.3 Keywords:

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

int	long
type	int
nof	dir
lt	fsatt
name	else
curpath	if
true	while
false	print
readfile	println
readdir	exit
file	date

1.4 Numbers:

A number can only be an integer. An integer consists of one or more digits. No decimal points or exponents are allowed. Examples are listed below.

1.5 White Space:

White space is defined as a space, tab or newline character.

1.6 Strings:

A string is sequence of one or more characters contained within double quotes. You may not escape single quotes.

1.7 Other Tokens:

The following tokens are used by the FJL language:

$	>	<	_
==	!=	||	>=
<=	!	&&	/
-	+	;	|
"	,	(	)
{	}	[	]
=	*	?	\
.	--		

2. Types:

FJL defines 5 different data types as follows:

int	Int object which will default to 0 if not assigned.
long	Long object which will default to 0 if not assigned.
fsatt	Sequence of letters or a sequence of letters and forward slashes.
file	File object containing information about a particular file.
dir	Directory object is essentially an array, containing file objects and directory information.

2.1 file:

The file type is an object descriptive of Unix files. It contains the follow attributes:

name	fsatt	The name of a Unix file.
type	fsatt	The type of a Unix file.
lt	int	The last modified time of a Unix file.

2.2 dir:

The dir type is an object detailing directories in a Unix filesystem. It is essentially an array of files and a few descriptive details about the concerned directory. It contains the following attributes:

name	fsatt	The name of a Unix directory.
nof	int	The number of Unix files in a directory.
lt	int	The last modified time of a directory.

3. Expressions:

Expressions consist of identifiers, constants, element access, regular expression patterns, and operators.

3.1 Identifiers:

An identifier is a left value expression. It is evaluated to the results of a right value expression.

3.2 String Constants:

A string constant is a right value expression. It is a sequence of characters, excluding the double quote, encased in double quotes.

3.3 Number Constants:

A number constant is a right value expression. It is an expression that is evaluated to an int.

3.3 Element Expression Access:

Elements of dir types are accessed by naming the identifier followed by an integer contained in left and right square brackets.

3.4 Regular Expressions:

Regular expressions are right value expressions. The expression is compared to a left value expression and returns true or false.

3.5 Operators:

Grouping can take place by enclosing an operation in parenthesis. The expression contained in parenthesis is evaluated left to right.

3.5.1 Arithmetic Operators:

The precedence follows as multiplication, division, addition, and subtraction, respectively. These operators can only be used when evaluating an int type.

Operators of this type are contained in the below chart:

*	/
+	-

3.5.2 Relational Operators:

Their evaluated result is true or false. Type int can be evaluated by all operators in the set.

==	!=	<=
=>	<	>

4.5.3 Logical Operators:

Their evaluated result is true or false. Logical operators make comparisons based on Boolean types returned by expressions.

&&	||

4.5.4 Regular Expression Operators:

The dollar sign symbolizes the end of the string. The left and right brackets are used for grouping of the regular expression. The forward slash is representative of the start and end of the regular expression.

$	[	]	/

5. Statements:

Statements in FJL are executed sequentially unless directed otherwise by flow control statements.

5.1 Declarations:

Type declarations in FJL are defined as follows and listed in the beginning of the language.

                type Identifier ;

12

The identifier is assigned a type. The semicolon must be present in the declaration.

5. 2 Flow Control Statements:

5.2.1 Loop Statements:

A looping mechanism is provided by FJL in the form of a while statement. While statements allow you to amongst other things, walk through directory listings gracefully. They take the following form:

```
while ( expression ) {
.
.
statement(s)
.
.
}
```

5.2.2 Conditional Statements:

Conditional statements use the if and else keywords and have the following syntax:

```
if ( expression ) {

statement(s)

}

or

if ( expression ) {

statement(s)

} else {

statement(s)

}
```

5.2.4 Print Statements:

Print statements can be used any where in the program. The are used to print the contents of a single identifier or constant.

5.2.4.1 Print:

Calling the print function will print the given identifier or

constant to the line.

```
print identifier ;

or

print constant ;
```

5.2.4.2 Println:

Calling the print function will print the given identifier or constant and a newline.

```
println identifier ;

or

println constant ;
```

5.2.5 Readfile Statements:

Readfile statements are used to change to another directory. The function is called with an identifier or string constant contained in parenthesis. It returns a value of type fsatt representing the name of the current directory. The syntax is as follows:

```
file f1= readfile(identifier) ;

or

file f1 = readfile(String constant) ;
```

5.2.6 Readdir Statements:

Readdir statements are used to read the given directory supplied to the function. The function is called with an identifier or string constant contained in parenthesis. It returns a value of type dir representing the given directories Unix files. The syntax is as follows:

```
dir givendir = readdir(identifier) ;

or

dir givendir = readdir(String constant) ;
```

5.2.7 Matches Statements:

Matches statements are used to equate a give left value identifier

with some right value regular expression. The statement is evaluated
to true or false. The syntax is as follows:

```
if ( identifier -- / [.] txt$/ ){
.
statement(s)
.
} else {
.
statement(s)
.
}
```

5.2.8 Exit Statements:

An exit statement is symbolic of the end of the program. Only one can
exist in a program and when met it will terminate. The syntax is as
follows:

```
exit;
```

5.2.9 Date Statements:

A date statement was added so that the time could be accessed in FJL.
It is particularly useful when evaluating your filesystem based on
some time criteria; It is a right hand assignment.

```
[long] Identifier = date;
```

6. Structure:

The program is read starting from the first non-comment line. It will
read until an exit statement is read or the end of file is reached.
Once the exit statement is reached the program will terminate.

Project Plan

1. Identify process used for planning, specification, and development:

    The beginning of the project incorporated me watching the lectures on ANTLR and Small Examples time and time again. I worked to come up with a language construct that would allow me to detail a filesystem.
    The project plan for me, since I was working all alone on the project was to give myself plenty of time to learn Antlr and get stuff wrong. I spent a great portion of the time on this project getting stuff really wrong and fixing it although that wasn't necessarily the over the plan. The plan to give myself enough time paid off greatly.
    I started off by designing my lexer. Once I thought I had looked at enough examples and evaluated the lexer enough, I moved on to the parser quickly realizing that the lexer needed more work. After coming up with what I though was a complete lexer and parser, I figured out how to visually display the AST and then when back to the drawing board after seeing my results. With the use of the display AST window, I eventually worked out all the bugs in the lexer and parser.
    From there I pain staking read over 4 or 5 different tree walkers to figure out what the heck they were doing and how I should move on from the point. I was stuck. It took me a good amount of time, I would say on the order of 20-25 hours or so to really start to be able to read the tree walker code and apply it for my needs. It took me even longer to get vaguely correct.
    I would develop java objects and expressional functions as I made my way through the tree walker. I started off very small at first. I wrote FJL code consisting of just declarations and then tried to walk that. Once I got that correct I proceeded in the same fashion first with assignments, then with built in functions and then moved on to expressions and flow control. I found that by making to many great leaps, left me destined for hours of debugging. Slow and steady, testing at all intervals was paramount.
    After finishing the tree walker and doing a lot of testing and reworking of my functions, I wrote the final report you are reading now.

2. Programming Style Guide:

2.1 Antlr Conventions:

    When coding the tree walker, parser and lexer, I attempted to clearly define rules and definitions in as concise manor as possible.

I worked to write my .g file so that everything was spaced as close
vertically together and horizontally similar so that I would be able
to quickly glance and read the code. I found that by not adhering to
these rules left searching and scanning lines to find rules and
definitions.

2.2 Java Conventions:

	I found that condensing objects and functions vertically was the
best format for me during the project. Because of the vast number of
files, and consequently objects defined, being able to see as much of
the code as possible was at the same time made it easier to go move
along in the project.
	I was not limited to a defined set of rules to follow as there
was no one else reading my code, but I followed classic Java style
formatting and indentations. Naming conventions followed a style I
like to code with, which is a short and non descriptive naming style.
Strings for example were typically name s1, s2 … .

2.3 General:

	Conventions weren't necessarily taken into consideration to much
while working on the project. Since I did all the work my self, I
found that just keeping an organized view of the assignment was the
best way I could help myself out. Organizing classes that extended
other classes and maintaining a strong visual for progression in the
project helped me out the most.

3. Project Timeline:

The project time line hoped to follow the below chart. I deviated
some due to a mild case of indirection when learning how to write a
tree parser.

June 11, 2007	Whitepaper
June 25, 2007	Finish Lexer & Parser
June 27, 2007	LRM
July 10, 2007	Version 1 of tree walker
July 17, 2007	Version 2 or tree walker w/ working programs
July 24, 2007	Version 3 and all testing done
July 31, 2007	Documentation Finished

4. Project Log:

The actual project log is shown below.

June 9, 2007	Project Proposal
June 10, 2007	Project Proposal
June 11, 2007	Whitepaper
June 20, 2007	First Version of Lexer
June 24, 2007	Updated Lexer, First Version of Parser
June 26, 2007	Version 2 of Parser and Lexer Done and testing
June 27, 2007	LRM
July 3, 2007	Tree walker research
July 7, 2007	Tree walker research
July 8, 2007	Tree walker research
July 14, 2007	Coding of interpreter and tree walker
July 15, 2007	Continued work on interpreter and walker
July 20, 2007 – August 5, 2007	Continued work on interpreter and walker, and testing
August 6, 2007 – August 9, 2007	Documentation & Testing

5. Team Responsibilities

The team worked very well with eachother. Every group member stayed in close contact with his self. John organized the team and as leader handed out jobs to all the other members. Briefly John worked on the lexer and parser. John also implemented the backend java interpreter and designed, wrote out and fixed all the bugs in the tree walker. John also worked on the test suite and wrote all the documentation. John worked very hard on all aspects and was very dedicated toward finishing the project.

6. Development Environment:

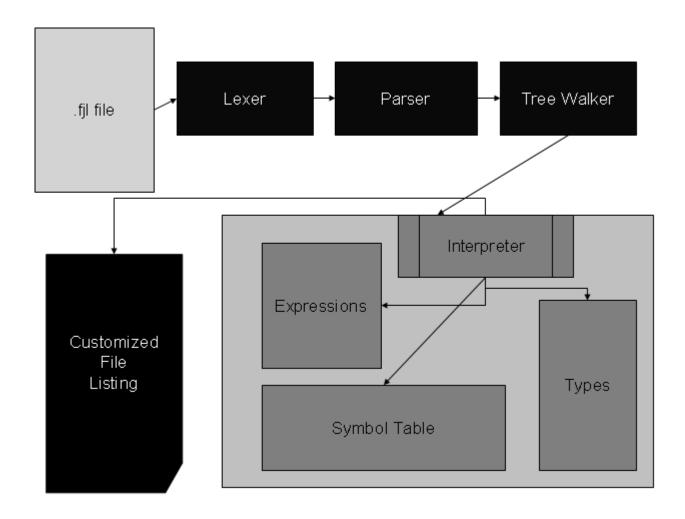All development took place on a single Ubuntu linux machine running the 2.6.17-12-generic kernel. I compiled everything with using java version 1.4.2-02 as that is the native version of java on my Ubuntu distribution. I used Antlr v2 (2.7.6-6) which was suggested by my package manager as a stable edition for my distribution version. I also used GNU Make version 3.81 to run make which I used as a compilation tool.

Architectural Design

      The FJL interpreter has a number of components that work together to produce the journaling output. After creating your .fjl file it is sent through the lexer and then to the parser after which whitespace and comments have been removed and the FJL program is represented as an AST tree. The AST is then walked by the tree walker checking the tree semantically. The tree walker will store variables inside of the Symbol Table for retrieval. It will then evaluate expressions and statements using functions that live in the Type and Expression classes as well as some that are defined within the tree walker code itself.

      The bulk of the work can be seen in the FJL grammer.g file. It contains the lexer, parser and walker classes for the FJL language. New objects are instantiated from with the walker classes.

      The three major set of classes that are used are the Expr, Type and SymbolTable classes. These classes do all the work that is not implemented in the grammer.g file. The Expr class is extended by the Id and Constant classes. The Type class is extended by several other classes, one for each type. The Dir, Fsatt, Int, JLong, and JFile classes all extend Type. The SymbolTable class does most of the work storing objects and retrieving them so that expressions can be evaluated.

The FJL Architectural Design

     As the only member of my team, I, John implemented all the classes from start to finish. He worked very hard.

Test Plan

     The testing for FJL was done incrementally at every phase of the assignment. Test programs were written to test the functionality of all small changes made to the FJL interpreter. The following tests were used to make the assertion that FJL was ready for its version 1.0 release.

     I wrote a perl script to continually loop through all my tests as I progressed through the testing phases. The following is the source of each of the tests along with the output if any was outputted. These test cases were all chosen so that I could be assured that as I was progressing other things did not break. During all the program testing I worked to assert that expressions were all returning the correct values and that flow control was working properly.

     Again as the sole developer of FJL, I did all the work. This program defines variables in FJL.

```
=====suite/decls.fjl========
|* This is My 1st Comment *|
int foo;
int bar;
file first;
dir barbaz;
dir second;
long time;
fsatt directory;
=====/suite/decls.fjl======
```

```
=========suite/decls.fjl Output======
=====End suite/decls.fjl Output======
```

This program defines variables in FJL.
It then assigns the variables and prints them out using the println and print built-ins.

```
=====suite/easy.fjl======
|* A Smaller Example *|

int j;
int i;
fsatt word;

word = "Assign this String";

println word;
```

```
i=5;
j=6;
i = i * j;

print "This is that value of i ";
println i;

exit ;
=====/suite/easy.fjl======
```

```
=========suite/easy.fjl Output=======
Assign this String
This is that value of i 30
=====End suite/easy.fjl Output=======
```

This program defines variables in FJL.
It then assigns the variables and prints them out using the println
and print built-ins.
Flow control is tested in this example with if and if/else
statements.

```
=====suite/flow.fjl=======
int i;
int j;
fsatt f1;
fsatt f2;

i = 6;
j = 7;

if ( i < j ){
  println "I < J";
  i = i + 1;
}

if ( i < j ){
  println "This should not be printed.";
} else {
  println "Incrementing worked and this should be printed.";
  j = j + 1;
  if ( i < j ) {
     println "I < J --- Again";
  }
}

exit;
=====/suite/flow.fjl======
```

```
========suite/flow.fjl Output======
I < J
Incrementing worked and this should be printed.
I < J --- Again
=====End suite/flow.fjl Output======
```

This program defines variables in FJL.
It then assigns the variables and prints them out using the println
and print built-ins.
Assignment of operators is the major test here. Also this program
tests reassignment
This program also uses the built in readdir function and uses array
access. It also uses
readfile and date builtins.

```
=====suite/assignment.fjl======
|* This is My 1st Comment *|
int foo;
file first;
dir second;
int i;
long time;
fsatt directory;


foo = 10+50*1/4-5;
println foo;
time = date;
println time;
directory="/home/jp/school/temp";
second = readdir(directory);
println second.curpath;
while (i<second.nof) {
        first = second[i];
        println first.name;
          println first.type;
      println first.lt;
      i = i + 1;
}

first = readfile("/home/jp/school/plt/fjl/test.txt");
println first.lt;
println first.type;
println first.name;

=====/suite/assignment.fjl=====
```

```
========suite/assignment.fjl Output======
17
1186686758544
/home/jp/school/temp
sample3.txt
file
1185653070000
second
directory
1186420043000
sample1.txt
file
1185653637000
sample2.txt
file
1185653072000
1186684399000
file
test.txt
=====End suite/assignment.fjl Output======
```

This program defines variables in FJL.
It then assigns the variables and prints them out using the println
and print built-ins.
Flow control is tested in this example with if/else statements and
while loops.
This program also uses the built in readdir function and uses array
access.

```
=====suite/bigger.fjl======
|* Simple Example: *|
int i;
dir first;
file foo;

first = readdir("/home/jp/school/temp");

while ( i < first.nof ){
    foo = first[i];
    if ( foo.name -- /txt$/ ){
        print "The file name is --- ";
        println foo.name;
    } else {
        print "This file name did not match --- ";
        println foo.name;
    }
```

```
     i = i + 1;
}
exit ;

=====/suite/bigger.fjl======



=========suite/bigger.fjl Output======
The file name is --- sample3.txt
This file name did not match --- second
The file name is --- sample1.txt
The file name is --- sample2.txt
=====End suite/bigger.fjl Output======
```

This program defines variables in FJL.
It then assigns the variables and prints them out using the println
and print built-ins.
Flow control is tested in this example with if/else statements and
while loops.
This program also uses the built in readdir function and uses array
access.
Amongst other things this program is an example of using matching,
built-in functions,
displaying file attributes.

```
=====suite/complicated.fjl======
int did;
int didnot;
file first;
dir second;
int i;
fsatt directory;

directory="/home/jp/school/temp";
second = readdir(directory);



while (i<second.nof) {
     first = second[i];
     if ( !(first.name -- /[.]txt$/) ){
        print first.name;
        print " has file type ";
        println first.type;
        didnot = didnot + 1;
     } else {
        did = did + 1;
     }
```

```
    if ( first.type -- /file/ ){
        print first.name;
        print " has file type ";
        println first.type;
    }
    i = i + 1;
}


print did;
println " Did match .txt ";
print didnot;
println " Did not match .txt";

exit ;
=====/suite/complicated.fjl======




========suite/complicated.fjl Output======
sample3.txt has file type file
second has file type directory
sample1.txt has file type file
sample2.txt has file type file
3 Did match .txt
1 Did not match .txt
=====End suite/complicated.fjl Output======
```

Lessons Learned

I learned that developing your own language is a lengthy practice. It takes a lot of understanding to get all the little parts of the process to talk to each other in the correct manor. I first started off with a very large set of rules and attempted tackle them all at the same time with out really completing any of them. That is a terrible idea. The biggest lesson I learned from this project was that you should start small and grow from there. Get the smallest piece working properly and then move on from there.

Since I did the entire program by myself from beginning to end, I was forced to overcome every obstacle by myself. In order to do so I was forced to read a bunch of the older projects and try to evaluate the code. It was a *very* lengthy process. I started early on the project and if I didn't I would never have completed.

Another very important lesson I learned is that taking this course through CVN with out TA's available made the project more time consuming that it had to be. I think that a little advising here and there would have moved me along a lot quicker. Teammates to share the some of work load would have helped as well.

In the end the project is very rewarding. It allows you to get a good feel of what it takes to actually build a useful language. I will take away a good understanding of the amount time it takes to learn Antlr and build a language using this tool. Because of this project I have really understood the idea of following a time line and getting to work early. It would have been disastrous had I waited to the last minute.

My advice to future teams is to start early and learn how the treewalker class really works.

Appendix

```
1.1
/**************************************************
*
*   Author: John Petrella – jep2124@columbia.edu
*
*   Date:  August 9, 2007
*
*   Filename: Arith.java
*
**************************************************/


public class Arith extends Op {
    public Expr expr1;
    public Expr expr2;
    public Arith(String op, Expr x1, Expr x2) {
        super(op, null);
     expr1 = x1;
     expr2 = x2;
        type = Type.max(expr1.type, expr2.type);
        if (type == null) error("type error");
    }
    public Expr gen() {
     return new Arith(s, expr1.reduce(), expr2.reduce());
    }
    public String toString() {
        return expr1.toString() + " " + s + " " + expr2.toString();
    }
}




1.2
/**************************************************
*
*   Author: John Petrella – jep2124@columbia.edu
*
*   Date:  August 9, 2007
*
*   Filename: Constant.java
*
**************************************************/


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

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

1.3
```
/*************************************************
*
*   Author: John Petrella – jep2124@columbia.edu
*
*   Date:  August 9, 2007
*
*   Filename: Dir.java
*
*************************************************/


import java.io.*;

public class Dir extends Type {

    public String curpath;
    public int nof;
    public long lt;
    public File[] files;

    public Dir(){
     super("dir");
     this.files = null;
     this.curpath = "";
     this.nof = 0;
     this.lt = 0;
    }

    public Dir(String directory){
     super("dir");
     this.curpath = directory;
     this.files = null;
     this.nof = 0;
     this.lt = 0;
    }
```

```java
    public File[] getFiles(){
     return this.files;
    }

    public File getFile(int i){
     return this.files[i];
    }

    public int getFileCount(){
     return this.files.length;
    }

    public String getCurPath(){
     return this.curpath;
    }

    public int getNof(){
     return this.nof;
    }

    public long getLt(){
     return this.lt;
    }

    public void setFiles(File file){
     this.files=file.listFiles();
    }

    public void setCurPath(String curpath){
     this.curpath=curpath;
    }

    public void setNof(int nof){
     this.nof=nof;
    }

    public void setLt(long lt){
     this.lt=lt;
    }

    public String toString(){
     return this.curpath;
    }
}
```

1.4
/***

```
*
*   Author: John Petrella - jep2124@columbia.edu
*
*   Date:  August 9, 2007
*
*   Filename: Expr.java
*
*************************************************/


public class Expr {
    public String s;
    public Type type;
    Expr(String tok, Type p) { s = tok; type = p;  }
    public String getToken() { return this.s; }
    public Type getType(){return this.type;}
    private Int i,j;
    private JLong l,k;

    public Expr arith(int op, Expr e){
     if ( this.type.getName().equals("int") &&
          e.getType().getName().equals("int") ){
         i = (Int)this.type;
         j = (Int)e.getType();
         int c=0;
         switch (op){
         case 1:
          c = i.getValue() + j.getValue();
          return new Expr(c+"",new Int(c));
         case 2:
          c = i.getValue() - j.getValue();
          return new Expr(c+"",new Int(c));
         case 3:
          c = i.getValue() * j.getValue();
          return new Expr(c+"",new Int(c));
         case 4:
          c = i.getValue() / j.getValue();
          return new Expr(c+"",new Int(c));
         }
      } else if ( this.type.getName().equals("long") &&
          e.getType().getName().equals("long") ){
         l = (JLong)this.type;
         k = (JLong)e.getType();
         long c=0;
         switch (op){
         case 1:
          c = l.getValue() + k.getValue();
          return new Expr(c+"",new JLong(c));
         case 2:
          c = l.getValue() - k.getValue();
          return new Expr(c+"",new JLong(c));
```

31

```java
       case 3:
        c = l.getValue() * k.getValue();
        return new Expr(c+"",new JLong(c));
       case 4:
        c = l.getValue() / k.getValue();
        return new Expr(c+"",new JLong(c));
       }

  }
  return null;
 }

 public Expr not(){
  if ( this.type.getName().equals("bool") ){
      String s1 = this.s;
      if ( s1.equals("true") ){
       return Constant.False;
      } else {
       return Constant.True;
      }
  }
  return Constant.False;
 }

 public Expr boollogic(int op, Expr e){
  if ( this.type.getName().equals("bool") &&
       e.getType().getName().equals("bool") ){
      String s1 = this.s;
      String s2 = e.getToken();;
      int c=0;
      switch (op){
      case 1:
       if ( s1.equals("true") &&
            s2.equals("true") )
          return Constant.True;
       else
          return Constant.False;
      case 2:
       if ( s1.equals("true") ||
            s2.equals("true") )
          return Constant.True;
       else
          return Constant.False;
      }
  }
  return Constant.False;
 }

 public Expr matches(Expr e){
  if ( this.type.getName().equals("fsatt") &&
       e.getType().getName().equals("fsatt") ){
```

```
        Fsatt f1 = (Fsatt)this.type;
        Fsatt f2 = (Fsatt)e.getType();
        String s1 = f1.getValue();
        String s2 = f2.getValue();
        int length = s2.length();
        if ( s2.indexOf('[') != -1 ){
            int m = s2.indexOf(']');
            if ( s1.indexOf(s2.substring(1,m)) != -1 ){
             if ( m == length-1 ){return Constant.True;}
             else {
                 m=m+1;
                 if (s2.substring(length-1,length)
                  .equals("$")){
                  if (s1.endsWith(s2.substring(m,length-1)))
                      return Constant.True;
                 } else
                  if (s1.indexOf(s2.substring(m,length)) != -1){
                      return Constant.True;
                  }
             }//else
            }
        }//Starts with []
        else if ( s2.substring(length-1,length).equals("$") ) {
            if ( s1.endsWith(s2.substring(0,length-1) ) )
             return Constant.True;
        }//Does not Start with [] but has $ at end
        else if ( s1.indexOf(s2) != -1 ){
            return Constant.True;
        }//Does not Start with [] or have a $ at end
 }
 return Constant.False;
}



public Expr logic(int op, Expr e){
 if ( this.type.getName().equals("int") &&
      e.getType().getName().equals("int") ){
     i = (Int)this.type;
     j = (Int)e.getType();
     int c=0;
     switch (op){
     case 1:
      if ( i.getValue() == j.getValue() ) return Constant.True;
       else return Constant.False;
     case 2:
      if ( i.getValue() != j.getValue() ) return Constant.True;
       else return Constant.False;
     case 3:
      if ( i.getValue() < j.getValue() )  return Constant.True;
```

```
          else return Constant.False;
        case 4:
         if ( i.getValue() <= j.getValue() ) return Constant.True;
         else return Constant.False;
        case 5:
         if ( i.getValue() > j.getValue() ) return Constant.True;
         else return Constant.False;
        case 6:
         if ( i.getValue() >= j.getValue() ) return Constant.True;
         else  return Constant.False;
        }
      } else   if ( this.type.getName().equals("long") &&
         e.getType().getName().equals("long") ){
         l = (JLong)this.type;
         k = (JLong)e.getType();
         long c=0;
         switch (op){
         case 1:
          if ( l.getValue() == k.getValue() ) return Constant.True;
          else return Constant.False;
         case 2:
          if ( l.getValue() != k.getValue() ) return Constant.True;
          else return Constant.False;
         case 3:
          if ( l.getValue() < k.getValue() )  return Constant.True;
          else return Constant.False;
         case 4:
          if ( l.getValue() <= k.getValue() ) return Constant.True;
          else return Constant.False;
         case 5:
          if ( l.getValue() > k.getValue() ) return Constant.True;
          else return Constant.False;
         case 6:
          if ( l.getValue() >= k.getValue() ) return Constant.True;
          else  return Constant.False;
         }
      }
      return null;
     }
}




1.5
/************************************************
*
*  Author: John Petrella - jep2124@columbia.edu
*
*  Date:  August 9, 2007
*
```

```
*   Filename: FJLMain.java
*
**************************************************/


import java.io.*;
import antlr.CommonAST;
import antlr.collections.AST;
import antlr.debug.misc.*;
import antlr.collections.AST;
import antlr.RecognitionException;
import antlr.TokenStreamException;
import antlr.TokenStreamIOException;


public class FJLMain {
    public static void main(String args[]) {
      try {
          String file = args[0];
          if (!file.substring(file.length()-
4,file.length()).equals(".fjl")){
                System.err.println("Wrong file type");
                System.exit(1);
          }
          FileInputStream filename = new FileInputStream(file);
          DataInputStream input = new DataInputStream(filename);
          FJLAntlrLexer lexer = new FJLAntlrLexer(input);
          FJLAntlrParser parser = new FJLAntlrParser(lexer);
          parser.decls(); // "file" is the main rule in the parser
          if ( lexer.error > 0 || parser.error > 0 ){
           System.err.println("Could not parse FJL Program");
           System.exit(1);
          }
          CommonAST parseTree = (CommonAST)parser.getAST();
          FJLAntlrWalker walker = new FJLAntlrWalker();
          Stmt s = walker.program(parseTree);
      } catch( RecognitionException e ) {
          System.err.println( "Recognition exception: " + e );
      } catch( TokenStreamException e ) {
          System.err.println( "TokenStream exception: " + e );
      } catch(Exception e) {
          System.err.println("Exception In Main: "+e);
      }
    }
}


1.6
/**************************************************
*
*   Author: John Petrella – jep2124@columbia.edu
```

```
*
*  Date:  August 9, 2007
*
*  Filename: Fsatt.java
*
**************************************************/


public class Fsatt extends Type {
    public String value ;

    public Fsatt(){
     super("fsatt");
     this.value = "";
    }

    public Fsatt(String value){
     super("fsatt");
     this.value = value;
    }

    public String toString(){
     return this.value;
    }

    public void setValue(String value){
     this.value = value;
    }

    public String getValue(){
     return this.value;
    }
}



1.7
/**************************************************
*
*  Author: John Petrella – jep2124@columbia.edu
*
*  Date:  August 9, 2007
*
*  Filename: grammer.g
*
**************************************************/


class FJLAntlrParser extends Parser;
options {
exportVocab = FJLAntlr;
```

```
buildAST = true;
k=2;
}
tokens { DECLS; STMT;}
{
    int error = 0;
    public void reportError( String s ) {
        super.reportError( s );
        error++;
    }
    public void reportError( RecognitionException e ) {
        super.reportError( e );
        error++;
    }
}



decls: (decl)* (stmt)* { #decls = #([DECLS, "DECLS"], #decls); } ;

decl
    : ( "int" | "long" | "fsatt" | "dir" | "file" ) ID SEMI!    ;

stmt
    : loc ASSIGN^ bool SEMI!
    | "print"^ ( ( ID | STRING ) | ext ) SEMI!
    | "println"^ ( ( ID | STRING ) | ext ) SEMI!
    | "while"^ LPAREN! bool RPAREN! stmt
    | "if"^ LPAREN! bool RPAREN! stmt
        ( options {greedy = true;}: "else"! stmt )?
    | LBRACE! (stmt)* RBRACE!
        {#stmt = #([STMT,"STMT"], stmt); }
    | "exit" SEMI!
    ;



loc       : ID^ (LBRACK! bool RBRACK!)* ;
bool      : join (OR^ join)* ;
join      : equality (AND^ equality)* ;
equality  : rel ((EQ^ | NE^) rel)* ;
rel       : expr ((LT^ | LE^ | GT^ | GE^ | MATCH^) expr)* ;
expr      : term ((PLUS^ | MINUS^) term)*  ;
term      : unary ((MULT^ | DIV^) unary)* ;
unary     : NOT^ unary | factor ;
factor
    : loc | NUMBER | "true" | "false"  | ext | "date"
    | STRING | LPAREN! bool RPAREN! | REGEX
    | ( "readdir" | "readfile" ) LPAREN! ( ID | STRING ) RPAREN!
    ;
```

```
ext : ID DOT^ ( "curpath" | "name" | "type" | "nof" | "lt" )  ;


class FJLAntlrLexer extends Lexer;
options {
testLiterals = false;
k = 2;
charVocabulary = '\3'..'\377';
}
{
    int error = 0;
    public void reportError( String s ) {
        super.reportError( s );
        error++;
    }
    public void reportError( RecognitionException e ) {
        super.reportError( e );
        error++;
    }
}

REGEX : ( '/'! ( (LBRACK (DOT)? ( LETTER | DIGIT )* RBRACK)?
                ( LETTER ( LETTER | DIGIT )+ ( '$' )? )? ) '/'! );
WS  :   ( ' ' | '\t' | '\n' { newline(); } | '\r' )
{ $setType(Token.SKIP); }     ;
STRING : '"'! ( ~'"' )* '"'! ;
protected LETTER  : 'A' .. 'Z' | 'a' .. 'z' ;
protected DIGIT : '0' .. '9' ;
NUMBER : ( DIGIT )+ ;
ID options { testLiterals = true; } : LETTER ( US | DIGIT | LETTER )*
;
COMMENT : ( (PIPE MULT) (options { greedy=false;} :.)* (MULT PIPE) )
 { $setType(Token.SKIP); } ;

DS : '$' ;          GT : '>' ;          LT : '<' ;          US : '_' ;
EQ : "==" ;         NE : "!=" ;         OR : "||" ;         GE : ">="
;
LE : "<=" ;         NOT : '!' ;         AND : "&&" ;    DIV : '/' ;
MINUS : '-' ;  PLUS : '+' ;   SEMI : ';' ;    PIPE : '|' ;
QUOTE : '"' ;  COMMA : ',' ;   KARAT : '^' ;   LPAREN : '(' ;
RPAREN : ')' ; LBRACE : '{' ; RBRACE : '}' ; LBRACK : '[' ;
RBRACK : ']' ; ASSIGN : '=' ; MULT : '*' ;    QUESTION : '?' ;
BSLASH : '\\' ;     SQUOTE : '\'' ;      DOT : '.' ;      MATCH : "--"
;


{
import java.util.Calendar;
import java.io.File;
}
class FJLAntlrWalker extends TreeParser;
```

```
options {
    importVocab = FJLAntlr;
}
{
    SymbolTable top = null;
    int scope=0;
}

program returns [Stmt s]
{ s = null; Type t = null; top = new SymbolTable();}
  : #(DECLS
        (t=type ID {
                    top.put(#ID.getText(), t, scope); }
            )*
        s=stmts
        )
  ;

type returns [Type t]
{ t = null; }
    : ( "bool"   { t = Type.Bool;   }
        | "fsatt"  { t = new Fsatt();     }
        | "int"    { t = new Int();    }
        | "file"   { t = new JFile();   }
        | "dir"    { t = new Dir();   }
        | "long"    { t = new JLong();   }
        )
    ;

stmts returns [Stmt s]
{ s = null; Stmt s1; }
    : (s=stmt)*
;

stmt returns [Stmt s]
{ Expr e1, e2;
    s = null;
    Stmt s1, s2;
}
    : #(ASSIGN e1=expr e2=expr
            { if ( ( ( e1 != null && e2 != null ) &&
                    ( e1 instanceof Id ) ) ){
                if ( !top.set(e1.getToken(),e2) )
                    top.error(" Type mismatch");
              } else {
                  top.error(" Assignment contains errors");
              }
            }
        )
    | #("while" whileexpr:. whilestmt:.
            { e1 = expr(#whileexpr);
```

39

```
                if ( e1 != null ){
                    while ( e1.getToken().equals("true") ){
                        s=stmt(#whilestmt);
                        e1 = expr(#whileexpr);
                    }
                } else {
                    top.error(" While expression contains errors");
                }
            }
        )
    | #("if" e1=expr thenp:. (elsep:.)?)
        { if ( (e1 != null) &&
            ( thenp != null ) &&
            (e1.getToken().equals("true")) ){
                s=stmt(#thenp);
        } else if ( null != elsep ){
            s=stmt(#elsep);
        }
    }
    | #("print" e1=expr
        {   if ( e1 != null ) {
                if ( e1 instanceof Id )
                    System.out.print(e1.getType().toString());
                else if ( e1.getToken().equals("file") )
                    System.out.print(e1.getType().toString());
                else if ( e1.getToken().equals("dir") )
                    System.out.print(e1.getType().toString());
                else if
( e1.getType().getName().equals("fsatt") )
                    System.out.print(e1.getToken());
            } else {
                top.error(" please declare and assign value");
            }
        }
    )
    | #("println" e1=expr
        {   if ( e1 != null ) {
                if ( e1 instanceof Id )
                    System.out.println(e1.getType().toString());
                else if ( e1.getToken().equals("file") ||
                    e1.getToken().equals("date") )
                    System.out.println(e1.getType().toString());
                else if ( e1.getToken().equals("dir") )
                    System.out.println(e1.getType().toString());
                else if
( e1.getType().getName().equals("fsatt") )
                    System.out.println(e1.getToken());
            } else {
                top.error(" please declare and assign value");
            }
        }
```

```
        )
    | #("exit" {System.exit(0);} )
    | #(STMT stmt:.
            {
                s=stmts(#stmt);
            }
        )
    ;


expr returns [Expr e]
{
  Expr a, b;
  e = null;
}
    : #(EQ      a=expr b=expr { e = a.logic(1,b); } )
    | #(NE      a=expr b=expr { e = a.logic(2,b); } )
    | #(LT      a=expr b=expr { e = a.logic(3,b); } )
    | #(LE      a=expr b=expr { e = a.logic(4,b); } )
    | #(GT      a=expr b=expr { e = a.logic(5,b); } )
    | #(GE      a=expr b=expr { e = a.logic(6,b); } )
    | #(MATCH   a=expr b=expr { e = a.matches(b); } )
    | #(NOT     a=expr        { e = a.not(); } )
    | #(AND     a=expr b=expr { e = a.boollogic(1,b); } )
    | #(OR      a=expr b=expr { e = a.boollogic(2,b); } )
    | #(PLUS    a=expr b=expr { e = a.arith(1,b); } )
    | #(MINUS   a=expr b=expr { e = a.arith(2,b); } )
    | #(MULT    a=expr b=expr { e = a.arith(3,b); } )
    | #(DIV     a=expr b=expr { e = a.arith(4,b); } )
    | NUMBER    { String s=#NUMBER.getText();
            e = new Expr(s, new Int(Integer.parseInt(s)));    }
    | STRING    { String s=#STRING.getText();
            e = new Expr(s, new Fsatt(s)); }
    | "true"    { e = Constant.True; }
    | "false"   { e = Constant.False; }
    | "readdir" a=expr {
            if ( a instanceof Id ){
                String directory = a.getType().toString();
                e = new Expr(directory, new Dir(directory));
            } else if ( a.getType().getName().equals("fsatt") ){
                e = new Expr(a.getToken(), new Dir(a.getToken()));
            } else {
                top.error(" error in readdir assign value");
            }
        }
    | "readfile" a=expr {
            try {
                File file = null;
                if ( a instanceof Id ){
                    file = new File(a.getType().toString());
                } else if ( a.getType().getName().equals("fsatt") ){
                    file = new File(a.getToken());
```

41

```
                }
                e = new Expr("file", new JFile(file));
            } catch (NullPointerException ne){
                System.out.println("Bad File Assignment");
                System.exit(1);
            }
        }
    | "date" {
            long l = Calendar.getInstance().getTimeInMillis();
            e = new Expr(""+l,new JLong(l));
            }
    | #(ID
            { Id i = top.get(#ID.getText());
                if (i == null) System.out.print(#ID.getText() + "
undeclared");
                e = i;
            }
            (a=expr
                {if ( a.getType().getName().equals("int") )
                    if (e.getType().getName().equals("dir") ){
                        Dir d = (Dir)e.getType();
                        Int j = (Int)a.getType();
                        e = new Expr("file",new
JFile(d.getFile(j.getValue())));
                    }
                }
            )?
        )
    | #(DOT a=expr b=expr
            {  if ( a instanceof Id ){
                    String type = a.getType().getName();
                    String ext = b.getToken();
                    if ( type.equals("dir") ){
                        Dir d = (Dir)a.getType();
                        if ( ext.equals("nof") ){
                            e = new Expr("dir", new Int(d.getNof()));
                        } else if ( ext.equals("curpath") ){
                            e = new Expr("dir", new
Fsatt(d.getCurPath()));
                        } else if ( ext.equals("lt") ){
                            e = new Expr("dir", new
JLong(d.getLt()));
                        }
                    } else if ( type.equals("file") ){
                        JFile jfile = (JFile)a.getType();
                        if ( ext.equals("name") ){
                            e = new Expr("file", new
Fsatt(jfile.getFileName()));
                        } else if ( ext.equals("type") ){
                            e = new Expr("file", new
Fsatt(jfile.getFileType()));
```

```
                              } else if ( ext.equals("lt") ){
                                  e = new Expr("file", new
JLong(jfile.getLt()));
                              }
                          }
                      }
                  }
              )
        | #(REGEX
                { String holder = #REGEX.getText();
                    e = new Expr(holder, new Fsatt(holder)); }
            )
        | "nof" { e = new Expr("nof", new Fsatt("nof")); }
        | "curpath" { e = new Expr("curpath", new Fsatt("curpath")); }
        | "type" { e = new Expr("type", new Fsatt("type")); }
        | "name" { e = new Expr("name", new Fsatt("name")); }
        | "lt" { e = new Expr("lt", new Fsatt("lt")); }
        ;




1.8
/************************************************
*
*   Author: John Petrella - jep2124@columbia.edu
*
*   Date:  August 9, 2007
*
*   Filename: Id.java
*
************************************************/


public class Id extends Expr{
    public String s;
    public Type type;
    public int scope;

    public Id(String id, Type p, int scope) {
     super(id,p);
     this.scope=scope;
    }

    public int getScope(){
     return this.scope;
    }
}


1.9
```

```java
/*************************************************
 *
 *   Author: John Petrella - jep2124@columbia.edu
 *
 *   Date:  August 9, 2007
 *
 *   Filename: Int.java
 *
 *************************************************/


public class Int extends Type {
    public int value ;

    public Int() {
     super("int");
     this.value = 0;
    }

    public Int(int value) {
     super("int");
     this.value = value;
    }

    public String toString(){
     return ""+this.value;
    }

    public void setValue(int value){
     this.value = value;
    }

    public int getValue(){
     return this.value;
    }
}


1.10
/*************************************************
 *
 *   Author: John Petrella - jep2124@columbia.edu
 *
 *   Date:  August 9, 2007
 *
 *   Filename: JFile.java
 *
 *************************************************/


import java.io.*;
```

```java
public class JFile extends Type {
    public long lt;
    public String fname;
    public String ftype;
    public File jfile;

    public JFile() {
     super("file");
     this.fname = "";
     this.ftype = "";
     this.lt = 0;
     this.jfile = null;
    }

    public JFile(File f1) {
     super("file");
     this.jfile = f1;
     this.fname = f1.getName();
     this.lt = f1.lastModified();
     if ( f1.isDirectory() ) this.ftype = "directory";
     else if ( f1.isFile() ) this.ftype = "file";
     else if ( f1.isHidden() ) this.ftype = "hidden file";
     else this.ftype = "Unknown";
    }

    public String toString(){
     return this.fname;
    }

    public void setFileName(String s){
     this.fname=s;
    }

    public void setFileType(String s){
     this.ftype=s;
    }

    public void setLt(long lt){
     this.lt=lt;
    }

    public void setJFile(File f1){
     this.jfile=f1;
    }

    public String getFileName(){
     return this.fname;
    }

    public String getFileType(){
```

```java
     return this.ftype;
     }

     public long getLt(){
      return this.lt;
     }
}




1.11
/*************************************************
 *
 *  Author: John Petrella - jep2124@columbia.edu
 *
 *  Date:  August 9, 2007
 *
 *  Filename: JLong.java
 *
 *************************************************/


public class JLong extends Type {
     public long value ;

     public JLong() {
      super("long");
      this.value = 0;
     }

     public JLong(long value) {
      super("long");
      this.value = value;
     }

     public String toString(){
      return ""+this.value;
     }

     public void setValue(long value){
      this.value = value;
     }

     public long getValue(){
      return this.value;
     }
}
```

1.12
```
/***********************************************
*
*   Author: John Petrella - jep2124@columbia.edu
*
*   Date:  August 9, 2007
*
*   Filename: Stmt.java
*
************************************************/


public class Stmt {
    public Stmt() {}
}
```


1.13
```
/***********************************************
*
*   Author: John Petrella - jep2124@columbia.edu
*
*   Date:  August 9, 2007
*
*   Filename: SymbolTable.java
*
************************************************/


import java.util.*;
import java.lang.Runtime.*;
import java.io.*;

public class SymbolTable {
    private Hashtable table;
    protected SymbolTable outer;
    public SymbolTable(){
     table=new Hashtable();
     outer=null;
    }
    public SymbolTable(SymbolTable st) {
        table = new Hashtable();
        outer = st;
    }
    public Id get(String token) {
     Id id = (Id)(table.get(token));
     if ( id != null ) return id;
     return null;
    }
```

```java
public void put(String token, Type t, int scope) {
 if ( get(token) == null ){
     table.put(token, new Id(token, t, scope));
 } else {
     System.err.println(token+" is already defined in fjl");
     System.exit(1);
 }
}

public boolean checkType(Type t, Type e){
 return t.getName().equals(e.getName());
}

public void error(String s){
 System.out.println(s);
 System.exit(0);
}

public String getType(Type t){return t.getName();    }

public boolean set(String token, Expr e){
 Id id = (Id) get(token);
 boolean same = checkType(id.getType(),e.getType());
 if ( same ){
     String value = getType(id.getType());
     if ( value.equals("fsatt") ){
      Fsatt f1 = (Fsatt)id.getType();
      Fsatt f2 = (Fsatt)e.getType();
      f1.setValue(f2.getValue());
      return true;
     } else if ( value.equals("int") ){
      Int i = (Int)id.getType();
      Int m = (Int)e.getType();
      i.setValue(m.getValue());
      return true;
     } else if ( value.equals("long") ){
      JLong l = (JLong)id.getType();
      JLong k = (JLong)e.getType();
      l.setValue(k.getValue());
      return true;
     } else if ( value.equals("dir") ){
      Dir dir = (Dir)id.getType();
      Dir d = (Dir)e.getType();
      try {
          File t1 = new File(d.curpath);
          boolean isDir = t1.isDirectory();
          if ( isDir ){
           dir.setCurPath(d.curpath);
           dir.setLt(t1.lastModified());
              dir.setFiles(t1);
           dir.setNof(dir.getFileCount());
```

```java
                } else {
                    error("Directory Assignment Failed for Identifier
"+token);
                }
            } catch (NullPointerException ne){
                System.out.println("Null Pointer In Set");
            } catch (SecurityException se){
                System.out.println("Security In Set");
            }
            return true;
        } else if ( value.equals("file") ){
            JFile jf1 = (JFile)id.getType();
            JFile jf2 = (JFile)e.getType();
            try {
                jf1.setJFile(jf2.jfile);
                jf1.setFileName(jf2.fname);
                jf1.setFileType(jf2.ftype);
                jf1.setLt(jf2.lt);
            } catch (NullPointerException ne){
                System.out.println("Null Pointer In Set");
            }
            return true;
        }
    }
    return false;
}

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

public String toString(){
    String s = ""; String key = "";
    boolean debug = true;
    if ( debug ){
        Enumeration keys = table.keys();
        while ( keys.hasMoreElements() ){
            key = (String)keys.nextElement();
            Id id = (Id) get( key );
            System.out.println(key+" "+id.getType().toString());
        } // end while
    } else {
        s = table.toString();
    }
    return s;
}
}
```

1.14
```
/***********************************************
*
*   Author: John Petrella – jep2124@columbia.edu
*
*   Date:  August 9, 2007
*
*   Filename: Type.java
*
***********************************************/


public class Type {
    public String name = "";
    public Type(String s) {
     name = s;
    }
    public static final Type Bool = new Type("bool");
    public String getName(){
     return this.name;
    }
}
```