Swinburne University Of Technology

Faculty of Science, Engineering and Technology

ASSIGNMENT COVER SHEET

Subject Code:	COS30023
Subject Title:	Languages in Software Development
Assignment number and title:	6, JavaCC – RPN & Stack Machine
Due date:	**October 6, 2014, 10:30, on paper**
Lecturer:	Dr. Markus Lumpe

Your name:____________________

Marker's comments:

Problem	Marks	Obtained
1	63	
Total	63	

Extension certification:

This assignment has been given an extension and is now due on __________________

Signature of Convener:_________________________

Assignment 6

COS30023 - Languages in Software Development

Daniel Parker - 971328X

October 5, 2014

1. Parser and AST

1.1. Test Input

```
load 20.0
dup
store $a
load 4.0
load 2.0
mul
dup
print "4 * 2 = "
load 1.0
add
dup
print "4 * 2 + 1 = "
store $x
```

1.2. Result

```
PCode accepted:
load 20.0
dup
store $a
load 4.0
load 2.0
mul
dup
print "4 * 2 = "
```

```
load 1.0
add
dup
print "4 * 2 + 1 = "
store $x
Running program:
4 * 2 = 8.0
4 * 2 + 1 = 9.0
Stack:
1:        20.0
Memory:
$x:        9.0
$a:        20.0
```

2. Source

2.0.1. PCodeParser.jj

```
options
{
  JDK_VERSION = "1.7";
  static = false;
  OUTPUT_DIRECTORY="parser";
}

PARSER_BEGIN(PCodeParser)
package parser;

import java.io.*;
import java.util.*;
import ast.*;
import machine.PCodeMachine;

public class PCodeParser {
        public static void main( String[] args ) {
                try {
                        PCodeParser lParser = new PCodeParser( new FileInputStream( args[0] ) );
                        ArrayList< PCode > lInstructions = lParser.Program();
                        System.out.println( "PCode accepted:" );
                        for ( PCode pc : lInstructions ) {
                                System.out.println( pc );
                        }

                        System.out.println( "Running program: ");
                        PCodeMachine lMachine = new PCodeMachine();

                        for ( PCode inst: lInstructions ) {
                                inst.accept( lMachine );
                        }

                        lMachine.printStackTrace();
                        lMachine.printMemoryTrace();
                } catch (ParseException e) {
                        System.out.println( "Syntax Error : \n"+ e.toString() );
                } catch( java.io.FileNotFoundException e ) {
                        System.err.println( e.toString() );
                }
        }
}
PARSER_END(PCodeParser)

ArrayList< PCode > Program():
{
        ArrayList< PCode > Result = new ArrayList< PCode >();
        PCode lInstruction = null;
}
{
        (lInstruction = PCodeInstruction() { Result.add( lInstruction ); })+ < EOF >
        { return Result; }
}

PCode PCodeInstruction():
{
        PCodeArgument argument;
        Token instruction;
        Token string;
        Token variable;
    }
    {
        instruction = "print" string = < STRING >
        {
            return new Print(instruction, string);
        }
    |
        instruction = "add"
        { return new Add(instruction); }
    |
        instruction = "sub"
        { return new Sub(instruction); }
    |
        instruction = "mul"
        { return new Mul(instruction); }
    |
        instruction = "div"
        { return new Div(instruction); }
    |
        instruction = "dup"
        { return new Dup(instruction); }
    |
        instruction = "load" argument = PCodeArgument()
        { return new Load( instruction, argument ); }
    |
        instruction = "store" variable = < VARIABLE >
        { return new Store( instruction, variable ); }
```

```
}

PCodeArgument PCodeArgument():
{
        Token arg;
}
{
        arg = < VARIABLE >
        { return new PCodeVariable( arg ); }
|
        arg = < NUMBER >
        { return new PCodeNumber( arg ); }
}

SKIP :
{
        " "
|
        "\r"
|
        "\t"
|
        "\n"
|
        < "//" (~["\n"])* "\n">
}

TOKEN :
{
        < VARIABLE : "$" (["a"-"z","A"-"Z","0"-"9","_"])+ >
```

```
  |
        < #EXPONENT : "E" ("+"|"-")? (["0"-"9"])+ >
  |
        < NUMBER : (["0"-"9"])+ ("." (["0"-"9"])*)? (< EXPONENT >)? >
  |
        < STRING : "\"" (~["\""])* "\"" >
}
```

2.0.2. ast.Position

```
package ast;

public class Position {
        public int fBeginLine;
        public int fBeginColumn;
        public int fEndLine;
        public int fEndColumn;
}
```

2.0.3. ast.PCode

```
package ast;

import machine.PCodeVisitor;
import parser.Token;

public abstract class PCode extends Position {
        public PCode( Token aInstruction ) {
                fBeginLine = aInstruction.beginLine;
                fBeginColumn = aInstruction.beginColumn;
                fEndLine = aInstruction.endLine;
                fEndColumn = aInstruction.endColumn;
        }

        public abstract String toString();

        public abstract void accept( PCodeVisitor aVisitor );
}
```

2.0.4. ast.PCodeArgument

```
package ast;

import machine.PCodeVisitor;

public abstract class PCodeArgument extends Position{
        public abstract String toString();

        public abstract Double accept( PCodeVisitor aVisitor );
}
```

2.0.5. ast.Add

```
package ast;

import machine.PCodeVisitor;
import parser.Token;
```

```
public class Add extends PCode {
        Token fToken;

        public Add(Token aInstruction) {
                super(aInstruction);

                fToken = aInstruction;
        }

        @Override
        public String toString() {
                return fToken.image.toString();
        }

        @Override
        public void accept(PCodeVisitor aVisitor) {
                aVisitor.visit(this);
        }
}
```

2.0.6. ast.Sub

```
package ast;

import machine.PCodeVisitor;
import parser.Token;

public class Sub extends PCode {
        Token fToken;
        public Sub(Token aInstruction) {
                super(aInstruction);

                fToken = aInstruction;
        }

        @Override
        public String toString() {
                return fToken.image.toString();
        }

        @Override
        public void accept(PCodeVisitor aVisitor) {
                aVisitor.visit(this);
        }
}
```

2.0.7. ast.Mul

```
package ast;

import machine.PCodeVisitor;
import parser.Token;

public class Mul extends PCode {
        Token fToken;

        public Mul(Token aInstruction) {
                super(aInstruction);
                fToken = aInstruction;
        }

        @Override
        public String toString() {
                return fToken.image.toString();
        }

        @Override
        public void accept(PCodeVisitor aVisitor) {
                aVisitor.visit(this);
        }
}
```

2.0.8. ast.Div

```
package ast;

import machine.PCodeVisitor;
import parser.Token;

public class Div extends PCode {
        Token fToken;

        public Div(Token aInstruction) {
                super(aInstruction);

                fToken = aInstruction;
        }
        @Override
        public String toString() {
                return fToken.image.toString();
        }

        @Override
        public void accept(PCodeVisitor aVisitor) {
                aVisitor.visit(this);
        }
}
```

2.0.9. ast.Dup

```
package ast;

import machine.PCodeVisitor;
import parser.Token;

public class Dup extends PCode {
        Token fToken;

        public Dup(Token aInstruction) {
                super(aInstruction);

                fToken = aInstruction;
        }

        @Override
        public String toString() {
                return fToken.image.toString();
        }

        @Override
        public void accept(PCodeVisitor aVisitor) {
                aVisitor.visit(this);
        }
}
```

2.0.10. ast.Load

```
package ast;

import machine.PCodeVisitor;
import parser.Token;

public class Load extends PCode {
        private PCodeArgument fArgument;

        public Load(Token aInstruction, PCodeArgument aArgument ) {
                super(aInstruction);

                fArgument = aArgument;

                this.fEndColumn = aArgument.fBeginColumn;
                this.fEndLine = aArgument.fEndLine;
        }

        public PCodeArgument getArgument() {
                return fArgument;
        }
        @Override
        public String toString() {
                StringBuilder sb = new StringBuilder();

                sb.append( "load " );
                sb.append( fArgument.toString() );

                return sb.toString();
        }

        @Override
        public void accept(PCodeVisitor aVisitor) {
                aVisitor.visit(this);
        }
}
```

2.0.11. ast.Store

```
package ast;

import machine.PCodeVisitor;
import parser.Token;

public class Store extends PCode {
        private String fVariableName;

        public Store(Token aInstruction, Token aVariable) {
                super(aInstruction);
        fEndLine = aVariable.endLine;
        fEndColumn = aVariable.endColumn;

        fVariableName = aVariable.image;
    }

    public String getVariableName() {
        return fVariableName;
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();

        sb.append( "store " );
        sb.append( fVariableName );

        return sb.toString();
    }

    @Override
    public void accept(PCodeVisitor aVisitor) {
        aVisitor.visit(this);
    }
}
```

2.0.12. ast.Print

```
package ast;
```

```
import machine.PCodeVisitor;
import parser.Token;

public class Print extends PCode{
        private String fMessage;

        public Print(Token aInstruction, Token aMessage) {
                super(aInstruction);

                fEndLine = aMessage.endLine;
                fEndColumn = aMessage.endColumn;

                fMessage = aMessage.image.subSequence(1, aMessage.image.length() - 1).toString();
        }

        @Override
        public String toString() {
                StringBuilder sb = new StringBuilder();

                sb.append( "print " );
                sb.append( "\"" + fMessage + "\"" );
                return sb.toString();
        }

        public String getMessage() {
                return fMessage;
        }

        @Override
        public void accept(PCodeVisitor aVisitor) {
                aVisitor.visit(this);
        }
}
```

2.0.13. ast.PCodeNumber

```
package ast;

import machine.PCodeVisitor;
import parser.Token;

public class PCodeNumber extends PCodeArgument{
        private Double fValue;

        public PCodeNumber(Token aNumber) {
                fBeginLine = aNumber.beginLine;
                fBeginColumn = aNumber.beginColumn;
                fEndLine = aNumber.endLine;
                fEndColumn = aNumber.endColumn;

                fValue = Double.parseDouble(aNumber.image);
        }

        public Double getValue() {
                return fValue;
        }

        @Override
        public String toString() {
                return fValue.toString();
        }

        @Override
        public Double accept(PCodeVisitor aVisitor) {
                return aVisitor.visit(this);
        }
}
```

2.0.14. ast.PCodeVariable

```
package ast;

import machine.PCodeVisitor;
import parser.Token;

public class PCodeVariable extends PCodeArgument{
        private String fVariableName;

        public PCodeVariable(Token aVariable) {
                fBeginLine = aVariable.beginLine;
                fBeginColumn = aVariable.beginColumn;
                fEndLine = aVariable.endLine;
                fEndColumn = aVariable.endColumn;

                fVariableName = aVariable.image;
        }

        public String getValue() {
                return fVariableName;
        }
        @Override
        public String toString() {
                return getValue();
        }

        @Override
        public Double accept(PCodeVisitor aVisitor) {
                return aVisitor.visit(this);
        }
}
```

2.0.15. machine.PCodeVisitor

```
package machine;

import ast.*;

public interface PCodeVisitor {
        public void visit( Add aInstruction );
        public void visit( Sub aInstruction );
        public void visit( Mul aInstruction );
        public void visit( Div aInstruction );
        public void visit( Dup aInstruction );
        public void visit( Print aInstruction );
        public void visit( Load aInstruction );
        public void visit( Store aInstruction );

        public Double visit( PCodeVariable aArgument );
        public Double visit( PCodeNumber aArgument );
}
```

2.0.16. machine.PCodeMachine

```
package machine;

import java.util.*;

import ast.Add;
import ast.Div;
import ast.Dup;
import ast.Load;
import ast.Mul;
import ast.PCodeNumber;
import ast.PCodeVariable;
import ast.Print;
import ast.Store;
import ast.Sub;

public class PCodeMachine implements PCodeVisitor {
        Stack< Double > fStack;
        Hashtable< String, Double > fMemory;

        public PCodeMachine() {
                fStack = new Stack< Double >();
                fMemory = new Hashtable< String, Double >();
        }

        public void printStackTrace() {
```

```
        System.out.println("Stack:");
        int i = 1;
        while (!fStack.empty()) {
                System.out.println(i + ":\t" + fStack.pop());
                i++;
        }
}

public void printMemoryTrace() {
        System.out.println("Memory:");
        for (String key: fMemory.keySet()) {
                System.out.println(key + ":\t" + fMemory.get(key));
        }
}

@Override
public void visit(Add aInstruction) {
        Double result = fStack.pop() + fStack.pop();
        fStack.add(result);
}

@Override
public void visit(Sub aInstruction) {
        Double result = fStack.pop() - fStack.pop();
        fStack.add(result);
}

@Override
public void visit(Mul aInstruction) {
        Double result = fStack.pop() * fStack.pop();
        fStack.add(result);
}

@Override
public void visit(Div aInstruction) {
        Double right = fStack.pop();
        Double left = fStack.pop();
        if (right == 0.0) {
                System.out.println( "Division by Zero, expression starting in line" + aInstruction.fBeginLine
                System.exit(1);
        }
        Double result = left / right;
        fStack.add(result);
}

@Override
public void visit(Dup aInstruction) {
        Double original = fStack.peek();
        fStack.push(original);
}

@Override
public void visit(Print aInstruction) {
        System.out.println(aInstruction.getMessage() + fStack.pop());
}

@Override
public void visit(Load aInstruction) {
        fStack.add(aInstruction.getArgument().accept(this));
}
```

```
        @Override
        public void visit(Store aInstruction) {
                fMemory.put(aInstruction.getVariableName(), fStack.pop());
        }

        @Override
        public Double visit(PCodeVariable aArgument) {
                return fMemory.get(aArgument.getValue());
        }

        @Override
        public Double visit(PCodeNumber aArgument) {
                return aArgument.getValue();
        }

}
```