CHAPTER V
IMPLEMENTATION AND TESTING

5.1 Implementation

This program is running in java, and used JSP as the interface. This program has five important steps, they are checking pattern of source code, divide string line into a token and describe the type of token, save the tokens in a tree and check semantic of the tokens. These are program listing the description :

1. index.jsp

This JSP code is used for show the choices of C tutorial, which is there are 3 tutorials, basic print in C, variable and operator, loop a string. When user press the 'Go Tutorial' button, index.jsp send request to proceed the C code that has been made before in file txt.

```
<html>
        <head>
        <title>Parsing</title>
        </head>
        <body>
        <center><h1>C Tutorial</h1></center>
        Choose tutorial :
        <form action="test.jsp" method="POST">
                <input type="radio" name="tutorial" value="HelloWorld" checked="checked" >Hello
World<br />
                <input type="radio" name="tutorial" value="Variable">Variable<br />
                <input type="radio" name="tutorial" value="Loop">Loop<br />
                <input type="radio" name="tutorial" value="looperror">Loop2<br />
                <input type="radio" name="tutorial" value="variableerror">Variable2<br />
                <input type="submit" value="Go to tutorial" >
        </form>
        </body>
</html>
```

2. Test.jsp

This JSP page is use for take tutorial that be selected by user from database file and shown in the textarea. In this textarea, user can change the source code.

```
<%@page import="java.io.*"%>

<html>
<head>
	<title>Parsing</title>
	<style type="text/css">
	textarea
	{
		font-size:12pt;
	}
	</style>
</head>
<body>
	<table align="center" border="1" width="80%">
	<tr>
		<th width="45%">Input Source Code</th>
	</tr>
	<tr>
		<td>
		<table border="0">
		<form action="save.jsp" method="POST">
		<tr>
			<td>
				<textarea name="inputbox" rows="30" cols="65" >
					<%
						String pilihan= request.getParameter("tutorial");
						String pilihan2=
"/usr/local/tomcat/webapps/gabung/source/"+pilihan+".txt";
						BufferedReader input = new BufferedReader(new
FileReader(pilihan2));

						String line = "";
						while ((line = input.readLine()) != null)
						{
							out.println(line);
						}
						input.close();
					%>
				</textarea>
			</td>
		</tr>
		<tr>
			<td>
				<input type="submit" value="input" >
			</td>
		</tr>
		</form>
		</table>
		</td>
	</tr>
	</table>
</body>
</html>
```

3. Save .jsp

Save.jsp receive request from Test.jsp and create a temporary file. Temporary file is a file that save source code that has been edited by user before. After create a temporary file, save.jsp will be redirected into output2.jsp to process the code.

```
<%@page import="java.io.*"%>
<%
response.sendRedirect("output2.jsp");
%>
<%

String input2=request.getParameter("inputbox");
out.println(input2);


//File creation
String strPath = "/usr/local/tomcat/webapps/gabung/source/temp.txt";
File strFile = new File(strPath);
boolean fileCreated = strFile.createNewFile();
//File appending
Writer objWriter = new BufferedWriter(new FileWriter(strFile));
objWriter.write(input2);
objWriter.flush();
objWriter.close();
%>
```

4. Output2.jsp

This jsp page is the main class of all processes. Every result of parsing algorithm also shown here. In this jsp page, also will appear error message if the source code has an error(s), though the pattern, double declare variable, or forget not declare the variable.

4.1 Checking regex

This class contain codes for checking regex pattern. These are the program listing :

```
<%@ page language="java"%>
<%@ page import="gabung.GabungRegex" %>
<%@ page import="gabung.Token" %>
<%@ page import="gabung.MyBinaryTree" %>
<%@ page import="tree.Node" %>
<%@ page import="java.io.*" %>
<%@ page import="java.util.StringTokenizer" %>
<%@ page import="java.util.*" %>
<html>
	<head>
	<title>parsing</title>
	<style type="text/css">
		textarea
		{
			font-size:12pt;
		}
	</style>
	</head>
	<body>
	<table width="80%" border="1">
		<tr>
			<th width="45%">Input</th>
			<th width="45%">Output</th>
		</tr>
		<tr>
			<td>
				<table>
					<form>
						<tr>
							<td>
<textarea readonly="true" name="input2"
rows="30" cols="65">
<%
BufferedReader teks =
new BufferedReader(new FileReader("/usr/local/tomcat/webapps/gabung/source/temp.txt"));
String line="";
while
((line=teks.readLine())!= null)
{
out.println(line);
}
%>
</textarea>
</td>
</tr>
</form>
</table>
</td>
```

This code is used to read the user code that has been edited before. This code read the input

user from temp.txt. In this page, the code can't be edited.

```
<td>

        <textarea readonly="true" name="output" rows="30" cols="65">
        <%

out.println("==========REGEX==========");
                GabungRegex jalan = new GabungRegex();

                String regInc=".+include.+";
                String regKomen="/\\*.+";
                String regMain=".+main.+";
                String regIden="int.+";
                String regIden2="char.+";
                String regPrint="printf.+";
                String regFor="for.+";
                String regReturn="return.+";
                String regKurungBuka="\\{";
                String regKurungTutup="\\}";

                int kurungBuka=0;
                int kurungTutup=0;
                int kode=0;

                String path="/usr/local/tomcat/webapps/gabung/source/temp.txt";
                int a=0;
                String delimiter=" ,;*/\"()=\\+";
                int hitung=0;
                String kata="";
                //while(kode!=1)
                //{
                        try
                        {
                        BufferedReader input = new BufferedReader(new FileReader(path));
```

This code is used for open file of temporary file, so regex can process it. This code also declare expression of regex that should be recognize. If there's no error, program will run.

```
while((line=input.readLine())!=null&&kode!=1)
{
	while(line.matches("^$"))//mengabaikan blank line
	{
		line=input.readLine();
	}
	line=line.replaceAll("\t+","");//remove tab
	if(line.matches(regInc))
	{
		jalan.regexInc(line);
		//out.println("includeeeee");
		if(jalan.regexInc(line)==1)
		{
			out.println("pattern "+line+" betul");
		}
		else
		{
			out.println("pattern "+line+" salah");
			kode=1;
		}
	}
	else if(line.matches(regKomen))
	//out.println("komeeeeeennnnn");
	jalan.regexKomen(line);
	if(jalan.regexKomen(line)==1)
	{
		out.println("pattern "+line+" betul");
	}
	else
	{
		out.println("pattern "+line+" salah");
		kode=1;
	}
}
```

This code is used for ignore new line and tab on source code. Also matches the include and comment pattern. If true, the program process to next step, if false, the program will be stopped. This code also call method in class GabungRegex.

```
else if(line.matches(regMain))
{
        //out.println("mainnnnnnnn");
        jalan.regexMain(line);
        if(jalan.regexMain(line)==1)
        {
                out.println("pattern "+line+" betul");
        }
        else
        {
                out.println("pattern "+line+" salah");
                kode=1;
        }
}
else if(line.matches(regIden)||line.matches(regIden2))
{
        //out.println("identifierrrrr");
        jalan.regexIden(line);
        if(jalan.regexIden(line)==1)
        {
                out.println("pattern "+line+" betul");
        }
        else
        {
                out.println("pattern "+line+" salah");
                kode=1;

        }
}
else if(line.matches(regFor))
{
        //out.println("forrrrrrrrr");
        jalan.regexFor(line);
        if(jalan.regexFor(line)==1)
        {
                out.println("pattern "+line+" betul");
        }
        else
        {
                out.println("pattern "+line+" salah");
                kode=1;
        }
}
```

This code is used for matches int main(), identifier and for pattern. If true, the program process to next step, if false, the program will be stopped. This code also call method in class GabungRegex.

```
else if(line.matches(regPrint))
{
        //out.println("printttttttttt");
        jalan.regexPrint(line);
        if(jalan.regexPrint(line)==1)
        {
                out.println("pattern "+line+" betul");
        }
        else
        {
                out.println("pattern "+line+" salah");
                kode=1;
        }
}
else if(line.matches(regReturn))
{
        //out.println("returrrrrrnnnnn");
        jalan.regexReturn(line);
        if(jalan.regexReturn(line)==1)
        {
                out.println("pattern "+line+" betul");
        }
        else
        {
                out.println("pattern "+line+" salah");
                kode=1;
        }
}
```

This code is used for matches printf and return 0 pattern. If true, the program process to next step, if false, the program will be stopped. This code also call method in class GabungRegex.

```
else if(line.matches(regKurungBuka))
{
        //out.println("kurung bukaaaaaa");
        kurungBuka=kurungBuka+1;
        out.println(line);
}
else if(line.matches(regKurungTutup))
{
        //out.println("kurung tutuppppp");
        kurungTutup=kurungTutup+1;
        out.println(line);
}
else
{
        //out.println("linee= "+line);
        //out.println("laaaaiiiinnnn");
        jalan.regexLain(line);//kata-kata yang lain
        if(jalan.regexLain(line)==1)
        {
                out.println("pattern "+line+" betul");
        }
        else
        {
                out.println("pattern "+line+" salah");
                kode=1;

        }
        }

}
```

This code is used for matches parenthesis and count how much open parenthesis and close parenthesis that used and matches pattern that not suitable with previous patterns. If true, the program process to next step, if false, the program will be stopped. This code also call method in class GabungRegex.

```
if(kurungBuka!=kurungTutup)
{
        out.println("Kurung eror");
        kode=1;
}
out.println("==========END REGEX=========");
}
catch(IOException e)
{
        out.println("error");
        kode=1;
}
```

This code is used for checking the parenthesis. If the number of open parenthesis not equals with close parenthesis the program will be error. Also catch if the Buffereader can't open the requested path, will shown error message.

```
package gabung;

import java.util.regex.Pattern;
import java.util.regex.Matcher;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.io.File;

public class GabungRegex
{
        int result=0;
        public int regexInc(String kata)
        {
        String eksInc="include <stdio.h>";
        Pattern inc = Pattern.compile(eksInc);//include
        Matcher m = inc.matcher(kata);
        if(m.find())
        {
                result=1;
        }
        else
        {
                result=0;
        }
        return result;
        }
```

Class GabungRegex is used for checking pattern of the code line. And send the value to output2 jsp. All of the code, will passed this checking.

```
public int regexFor(String kata)
{
	String eksFor="^for[(][a-zA-Z0-9(=)(;)(<)(>)(<=)(>=)(++)]*[)]";
	Pattern forr = Pattern.compile(eksFor);
	Matcher m = forr.matcher(kata);
	if(m.find())
	{
		result=1;
	}
	else
	{
		result=0;
	}
	return result;
}

public int regexKomen(String kata)
{
	String eksKomen="/\\*(.|[\r\n])*?\\*/";
	Pattern komen = Pattern.compile(eksKomen);
	Matcher m = komen.matcher(kata);
	if(m.find())
	{
		result=1;
	}
	else
	{
		result=0;
	}
	return result;
}
public int regexMain(String kata)
{
	String eksMain="int main[(][)]";
	Pattern mainn = Pattern.compile(eksMain);//int main()
	Matcher m = mainn.matcher(kata);
	if(m.find())
	{
		result=1;
	}
	else
	{
		result=0;
	}
	return result;
}
```

This code is used to check pattern of for, comment, and int main(). 0 if false and 1 if true. The result will be sent in output2.jsp.

```
public int regexIden(String kata)
{
        String eksIden="[a-z]\\s[a-zA-Z0-9(,)]*(;)|[a-z]\\s[a-zA-Z0-9](=)[0-9](;)|[a-z]\\s[a-zA-Z0-9]([)
[0-9]([])(.)";//int a,b,c|int a=6;
        Pattern iden = Pattern.compile(eksIden);//identifier
        Matcher m = iden.matcher(kata);
        if(m.find())
        {
                result=1;
        }
        else
        {
                result=0;
        }
        return result;
}

public int regexReturn(String kata)
{
        String eksReturn="return 0(;)";
        Pattern retur = Pattern.compile(eksReturn);
        Matcher m = retur.matcher(kata);
        if(m.find())
        {
                result=1;
        }
        else
        {
                result=0;
        }
        return result;
}
```

This code is used for check pattern of identifier and return 0. 0 if false and 1 if true. And send the return value to output2.jsp

```
public int regexPrint(String kata)
{
        String eksPrin="^printf[(]\"[a-zA-Z0-9\n(:)(=)\\s.*]*\"[)](;)";//printf("hello");
        String eksPrin2="printf[(]\"%d\"(,)[a-zA-Z0-9][)](;)|printf[(]\"[%d(,)]*\"(,)[a-zA-Z0-9(,)]*[)]
(;)";//printf("%d",a)|printf("%d,%d",a,b);
        String eksPrin3="printf[(]\"[a-zA-Z0-9(=)\\s]*%d.*\"(,)[a-zA-Z0-9][)](;)";//printf("a=%d",b);
        String eksPrin5="printf[(]\"[a-zA-Z0-9(([)%d(])(=)\\s]*%d.*\"(,)[a-zA-Z0-9(,)([)[a-zA-Z0-9](])*[)]]
(;)";//printf("a[%d]= %d\n",i,a[i]);
        String eksPrin4="printf[(]\".*\"[)](;)";//\n

        Pattern prin = Pattern.compile(eksPrin);//printf
        Pattern prin2 = Pattern.compile(eksPrin2);//printf
        Pattern prin3 = Pattern.compile(eksPrin3);//printf
        Pattern prin4 = Pattern.compile(eksPrin4);//printf
        Pattern prin5 = Pattern.compile(eksPrin5);//printf
        Matcher m = prin.matcher(kata);
        Matcher n = prin2.matcher(kata);
        Matcher o = prin3.matcher(kata);
        Matcher p = prin4.matcher(kata);
        Matcher q = prin5.matcher(kata);
        if(m.find())
        {
                result=1;
        }
        else if(n.find())
        {
                result=1;
        }
        else if(o.find())
        {
                result=1;
        }
        else if(p.find())
        {
                result=1;
        }
        else if(q.find())
        {
                result=1;
        }
        else
        {
                result=0;
        }
        return result;
}
```

This code is used for check pattern of printf. 0 if false and 1 if true. And send the return value to output2.jsp

```
public int regexLain(String kata)
{
        String eksVar="[a-zA-Z0-9(=)]+[0-9](;)";//a=5
        String eksVar2="[a-zA-Z0-9](=)+[0a-zA-Z0-9(+)\\-(/)\\*]*(;)";//a=6+7+b
        String eksVar3="[a-zA-Z0-9(=)]+[0a-zA-Z0-9(+)(-)]+[0a-zA-Z0-9]";
        Pattern var1 = Pattern.compile(eksVar);
        Pattern var2 = Pattern.compile(eksVar2);
        Pattern var3 = Pattern.compile(eksVar3);
        Matcher m = var1.matcher(kata);
        Matcher n = var2.matcher(kata);
        Matcher o = var3.matcher(kata);
        if(m.find())
        {
                result=1;
        }
        else if(n.find())
        {
                result=1;
        }
        else if(o.find())
        {
                result=1;
        }
        else if(kata.matches("^$"))
        {
                result=1;
        }
        else
        {
                result=0;
        }
        return result;
        }
}
```

This code is used for check pattern of declare variable or do operator. 0 if false and 1 if true. And send the return value to output2.jsp

4.2 Divide into Token and Describe the Tokenization

In this part, listing code will break the line into tokens and describe the type of the tokens. These are the listing program :

```
if(kode!=1)
{
        out.println("==============START TOKEN==============");
        Token abc= new Token();
        String path2="/usr/local/tomcat/webapps/gabung/source/temp.txt";

        int panjang=0;
        int index=0;
        int index1=0;

        char result=0;
        char result1=0;
        char space=32;
        char titikkoma=59;
        char lbhkecil=60;
        char smdengan=61;
        char lbhbesar=62;
        char koma=44;
        char kali=42;
        char tambah=43;
        char minus=45;
        char titik=46;
        char krgbuka=40;
        char krgttp=41;
        char petikdua=34;
        char backSlash=92;
        String b="";
        String hasil2="";

        int i=0;
        int hasil1=0;
        int g=0;
        int h=0;
        int gb=0;
        int gj=0;
        int ulang=0;
        String pathKeyword="/usr/local/tomcat/webapps/gabung/source/tblKeyword.txt";
        String pathSymbol="/usr/local/tomcat/webapps/gabung/source/tblSymbol.txt";
        String pathOperator="/usr/local/tomcat/webapps/gabung/source/tblOperator.txt";

        String lineKeyword="";
        String lineSymbol="";
        String lineOperator="";
```

This code is used for declare variable that needed. The tokens that must recognized before also read in this code. This code also call class Token.

```
String[] keyword = new String[50];
String[] symbol = new String[50];
String[] operator = new String[50];
String[] hasilnya = new String[200];
String[] cekDeclare = new String[20];

int k=0;
int s=0;
int o=0;
int cekvar  0;
String dobel="";
int[] simpanBatasAkhir= new int[100];//untuk menyimpan batas nilai pada tree
int[] simpanBatasAwal= new int[100];//untuk menyimpan batas nilai pada tree

String[] gabungToken=new String[500];
try
{
        BufferedReader input = new BufferedReader(new FileReader(path2));
        BufferedReader inputKeyword = new BufferedReader(new FileReader(pathKeyword));
        BufferedReader inputSymbol = new BufferedReader(new FileReader(pathSymbol));
        BufferedReader inputOperator = new BufferedReader(new FileReader(pathOperator));
        while((lineKeyword=inputKeyword.readLine())!=null)
        {
                keyword[k]=lineKeyword;
                k++;
        }
        while((lineSymbol=inputSymbol.readLine())!=null)
        {
                symbol[s]=lineSymbol;
                s++;
        }
        while((lineOperator=inputOperator.readLine())!=null)
        {
                operator[o]=lineOperator;
                o++;
        }
```

This code is used for read file text that contain tokens that must describe. This recognized tokens is used for categorize the tokens. And also checking type of the tokens.

```
int print=0;
String simpan="printf";
while((line=input.readLine())!=null)
{
        while(line.matches("^$"))//mengabaikan blank line
        {
                line=input.readLine();
        }
        line=line.replaceAll("\t+","");
        if(line.matches(regMain))
        {
                line=input.readLine();
        }
        if(line.matches(regInc))
        {
                line=input.readLine();
        }
        if(line.matches(regKomen))
        {
                line=input.readLine();
        }
        if(line.matches(regReturn))
        {
                line=input.readLine();
        }
        b= line+" ";
        out.println("==NEW LINE==");
        out.println(line);

        Index=0;
        while(index < line.length())
        {
                result=b.charAt(index);

                int r=0;
                int hitungResult=0;
                result1=result;
                index1=index;
                int p=k;
                k=0;
                o=0;
                s=0;
                int cek=0;
                out.println("==TOKEN==");
```

This code is used for ignore many lines that doesn't needed. Such as include <stdio.h>, blank line, comment, etc. Because that lines doesn't important on tokenization process.

```
while(result1!=space&&result1!='\\'&&result1!=titikkoma&&result1!=koma&&result1!
=tambah&&result1!=minus&&result1!='*'&&result1!='('&&result1!=')'&&result1!='<'&&result1!
='>'&&result1!='"'&&result1!='.'&&result1!=smdengan&&result1!='/'&&result1!='{'&&result1!='}')
{
        hitungResult++;
        index1++;
        result1=b.charAt(index1);
}
char[] resultArray=new char[hitungResult];
while(result!=space&&result!='\\'&&result!=titikkoma&&result!=koma&&result!=tambah&&result!
=minus&&result!='*'&&result!='('&&result!=')'&&result!='<'&&result!='>'&&result!='"'&&result!
='.'&&result!=smdengan&&result!='/'&&result!='{'&&result!='}')
{
        resultArray[r]=result;
        index=index+1;
        result=b.charAt(index);
        r=r+1;
}
if(r!=0)
{
                String resultString= new String(resultArray);
                out.println(resultString);
                gabungToken[i]=resultString;
                //out.println("print= "+gabungToken[i]);
                if(gabungToken[i].equals(simpan))
                {
                                print=1;
                                //out.println("print= "+print);
                }
                //out.println(keyword[3]);
                //out.println(gabungToken[i]);
```

This code is used for check if they are not include in token that described before. If not included, code create a string from array of character and saved into gabungToken array. This code also check if the liner is inside of printf, set the print=1;

```
while(keyword[k]!=null)
{
        if(gabungToken[i].equals(keyword[k]))
        {
                out.println("==TOKENIZATION==");
                out.println(gabungToken[i]+" = keyword");
                hasilnya[i]="keyword";
                cek=1;
                break;
        }
        else
        {
                k=k+1;
        }
}
if(cek==0)
{
        String aaa=abc.cekToken(gabungToken[i],gabungToken[i-1],print);
        out.println("==TOKENIZATION==");
        out.println(gabungToken[i]+" = "+aaa);
        hasilnya[i]=aaa;
        if(hasilnya[i].equals("variable"))
        {
                if(hasilnya[i-1].equals("keyword"))
                {
                        cekDeclare[g]=gabungToken[i];
                        g++;
                }
        }
        //out.println("not equal");
        //k ++;
}
i=i+1;
}
if(result==space)
{
        //index=index+1;
        abc.pecahSpace(b);
        out.println(abc.pecahSpace(b));
        result=b.charAt(index);
}
```

This code is used for do tokenization and compare with database keyword txt that contain type of tokens. If not equals call method Token cekToken. In this code also save variables that declared, co can be checked if there are same variable declared. Also ignore space.

```
else if(result==backSlash)
{
        //out.println(result);
        if(b.charAt(index+1)==110||b.charAt(index+1)=='t')
        {
                out.println("");
                abc.pecahKata(result,b.charAt(index+1));
                String hasil=abc.pecahKata(result,b.charAt(index+1));
                gabungToken[i]=hasil;
                out.println(hasil);
                while(keyword[k]!=null)
                {
                        if(gabungToken[i].equals(keyword[k]))
                        {
                                out.println("==TOKENIZATION==");
                                out.println(gabungToken[i]+" = keyword");
                                hasilnya[i]="keyword";
                                break;
                        }
                        else
                        {
                                //out.println("not equal");
                                k++;
                        }
                }
                //out.println(i);
                i=i+1;
                //out.println(i);
                index=index+1;
                result=b.charAt(index);
        }
        else
        {
                out.println(" ");
                out.print(result);
                //index=index+1;
                gabungToken[i]=Character.toString(result);
                i=i+1;
                //out.println(i);
                result = b.charAt(index);
                out.println(" ");
        }
}
```

This code is used for do token of backslash code . This code also checking is it a new line code, or only a backslash symbol. After that, the backslash saved into tokenization.

```
else if(result=='=')
{
        if(b.charAt(index+1)=='=')
        {
                out.println("");
                abc.pecahKata(result,b.charAt(index+1));
                String hasil=abc.pecahKata(result,b.charAt(index+1));
                gabungToken[i]=hasil;
                i=i+1;
                out.println(hasil);
                while(operator[o]!=null)
                {
                        if(gabungToken[i].equals(operator[o]))
                        {
                                out.println("==TOKENIZATION==");
                                out.println(gabungToken[i]+" = operator");
                                hasilnya[i]="operator";
                                break;
                        }
                        else
                        {
                                //out.println("not equal");
                                o++;
                        }
                }
                index=index+1;
                result=b.charAt(index);
        }
        else
        {
                out.println(" ");
                out.println(result);
                gabungToken[i]=Character.toString(result);
                while(operator[o]!=null)
                {
                        if(gabungToken[i].equals(operator[o]))
                        {
                                out.println("==TOKENIZATION==");
                                out.println(gabungToken[i]+" = operator");
                                hasilnya[i]="operator";
                                break;
                        }
                        else
                        {
                                //out.println("not equal");
                                o++;
                        }
                }
```

This code is used for do token and tokenization equal operator . This code also checking is it followed by another equal symbol or not. And saved it into tokenization array.

```
i=i+1;
//out.println(i);
result = b.charAt(index);
out.println(" ");
}
}
else if(result==tambah)
{
        //out.println(result);
        if(b.charAt(index+1)==tambah)
        {
                out.println("");
                abc.pecahKata(result,b.charAt(index+1));
                String hasil=abc.pecahKata(result,b.charAt(index+1));
                gabungToken[i]=hasil;
                out.println(hasil);
                while(operator[o]!=null)
                {
                        if(gabungToken[i].equals(operator[o]))
                        {
                                out.println("==TOKENIZATION==");
                                out.println(gabungToken[i]+" = operator");
                                hasilnya[i]="operator";
                                break;
                        }
                        else
                        {
                                //out.println("not equal");
                                o++;
                        }
                }
                //out.println(i);
                i=i+1;
                //out.println(i);
                index=index+1;
                result=b.charAt(index);
        }
        else
        {
                out.println(" ");
                out.println(result);
                //index=index+1;
                gabungToken[i]=Character.toString(result);
```

This code is used for do token and tokenization + operator . This code also checking is it followed by another + symbol or not. After that saved it into tokenization array.

```
while(operator[o]!=null)
{
        if(gabungToken[i].equals(operator[o]))
        {
                out.println("==TOKENIZATION==");
                out.println(gabungToken[i]+" = operator");
                hasilnya[i]="operator";
                break;
        }
        else
        {
                //out.println("not equal");
                o++;
        }
}
i=i+1;
//out.println(i);
result = b.charAt(index);
out.println(" ");
}
}
else if(result==minus)
{
        //out.println(result);
        if(b.charAt(index+1)==minus)
        {
                out.println("");
                abc.pecahKata(result,b.charAt(index+1));
                String hasil=abc.pecahKata(result,b.charAt(index+1));
                gabungToken[i]=hasil;
                out.println(hasil);
                while(operator[o]!=null)
                {
                        if(gabungToken[i].equals(operator[o]))
                        {
                                out.println("==TOKENIZATION==");
                                out.println(gabungToken[i]+" = operator");
                                hasilnya[i]="operator";
                                break;
                        }
                        else
                        {
                                //out.println("not equal");
                                o++;
                        }
                }
```

```
//out.println(i);
i=i+1;
//out.println(i);
index=index+1;
result=b.charAt(index);
}
else
{
        out.println(" ");
        out.println(result);
        //index=index+1;
        gabungToken[i]=Character.toString(result);
        while(operator[o]!=null)
        {
                if(gabungToken[i].equals(operator[o]))
                {
                        out.println("==TOKENIZATION==");
                        out.println(gabungToken[i]+" = operator");
                        hasilnya[i]="operator";
                        break;
                }
                else
                {
                //out.println("not equal");
                o++;
                }
        }
        i=i+1;
        //out.println(i);
        result = b.charAt(index);
        out.println(" ");
        }
}
```

This code is used for do token and tokenization of - operator . This code also checking is it followed by another - symbol or not. And saved it into tokenization array.

```
else if(result==koma||result=='*'||result=='{'||result=='}'||result=='('||result==')'||result=='<'||result=='>'||
result=='"'||result=='.'||result=='/')
{
        out.println(" ");
        out.println(result);
        //index=index+1;
        gabungToken[i]=Character.toString(result);
        while(operator[o]!=null||symbol[s]!=null)
        {
                if(gabungToken[i].equals(operator[o]))
                {
                        out.println("==TOKENIZATION==");
                        out.println(gabungToken[i]+" = operator");
                        hasilnya[i]="operator";
                        break;
                }
                else if(gabungToken[i].equals(symbol[s]))
                {
                        out.println("==TOKENIZATION==");
                        out.println(gabungToken[i]+" = symbol");
                        hasilnya[i]="symbol";
                        break;
                }
                else
                {
                        //out.println("not equal");
                        o++;
                        s++;
                }
        }
        //out.println(i);
        i=i+1;
        //out.println(i);
        result = b.charAt(index);
        out.println(" ");
        }
```

This code is used for do token and tokenization symbol and operator. This code compare the tokens with database keyword. After that save it into array tokenization.

```
else if(result==titikkoma)
{
        out.println(" ");
        out.println(result);
        gabungToken[i]=Character.toString(result);
        while(symbol[s]!=null)
        {
                if(gabungToken[i].equals(symbol[s]))
                {
                        out.println("==TOKENIZATION==");
                        out.println(gabungToken[i]+" = symbol");
                        hasilnya[i]="symbol";
                        break;
                }
                else
                {
                        //out.println("not equal");
                        s++;
                }
        }
        //out.println(i);
        i=i+1;
}
//out.print(index);
//i=i+1;
index=index+1;
}
gb=i;
int jk=0;
for(jk=gj;jk<gb;jk++)//untuk mengetahui token apa saja dalam 1 line digunakan untuk tree
{
        //System.out.println("jk = "+jk);
}
//System.out.println("jk 2= "+jk);
simpanBatasAkhir[ulang]=jk;
//System.out.println("gj 2= "+gj);
simpanBatasAwal[ulang]=gj;
gj=gb;
ulang++;
print=0;
}
```

This code is used for do token and tokenization symbol semicolon. And for know the start token and end token on each line. That be used for syntax analyzer.

```
int y=i;
//i=0;
//==========check double variable==========
g=0;
while(cekDeclare[g]!=null)
{
        //out.println(cekDeclare[g]);
        g++;
}
h=0;
cekvar=g;
g=0;
for(int w=0;w<cekvar;w++)
{
        for(int p=0;p<cekvar;p++)
        {
                if(w!=p&&cekDeclare[w].equals(cekDeclare[p]))
                {
                        h=1;
                        dobel=cekDeclare[w];
                }
        }
}
if(h==1)//if double declare error
{
        out.println("eror double declare variable "+dobel);
}
}
catch(IOException e)
{
        out.println("error");
}
```

This code is used for check if in source code found same declare variable. The variable that has been declared before were saved in array cekDeclare. If found same declare variable, the program stopped and show error message.

```
package gabung;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.io.File;

public class Token
{
        int index=0;
//      String kata="";
//      String kata2="";
        String hasil="";
        char result=0;
        public String pecahSpace(String kata)
        {
        hasil=" ";
        return hasil;
        }
        public String pecahKata(char kata,char kata2)
        {
        hasil=kata+""+kata2;
        return hasil;
        }


}
```

This class Token is used for make token from source code program. This class also combine char that doesn't describe as a token before. After divide into token, this class sent the result into Output2.jsp.

4.3 Creating Tree to Save the Tokens

```
//==========creating tree ==========
if(h!=1)
{
        out.println("======Tree======");
        i=0;
        ulang=1;
        int jarak=0;
        int j=1;
        int batasFor=0;
        int cekFor=0;
        int pembatas=0;
        int tentuRoot=0;
        int simpanJ=0;
        int cekSemantic=1;
        String hasilPrint="";
        int [] keywordPersen= new int[200];
        int persen=0;
        int [] as= new int[200];
        String [] hasilOutput= new String[200];
        int output=0;
        int cek=0;//utk cek var sudah ada blm
        int xyz=0;//isi dari int [] variable1
        String [] variable1 = new String[200];//variable name
        int [] variable2 = new int[200];
        String variable="";
        while(simpanBatasAkhir[ulang]!=0&&simpanBatasAkhir[ulang]!=simpanBatasAwal[ulang])
        {
                MyBinaryTree treeAbc= new MyBinaryTree();
                pembatas=simpanBatasAkhir[ulang]-simpanBatasAwal[ulang]-1;
                tentuRoot=(pembatas/2)+simpanBatasAwal[ulang];
                //out.println("jarak= "+jarak);
                //out.println("tentu = "+tentuRoot);
                treeAbc.add(tentuRoot,tentuRoot);
                for(jarak=simpanBatasAwal[ulang];jarak<(simpanBatasAkhir[ulang]);jarak++)
                {
                        //out.println("aaaa = "+gabungToken[i]);
                        //out.println("token= "+gabungToken[i]+" i = "+i);
                        if(i!=tentuRoot)
                        {
                                treeAbc.add(i,tentuRoot);
                        }

                        //treeAbc.printInOrder(treeAbc.root);
                        i++;
                }
```

This code is used for make a binary tree from token that got before. In this code, call class MyBinaryTree to saved the token. This step is called Syntax Analyzer.

```
as=treeAbc.printInOrder(treeAbc.root);
int root1 = as[0];
out.println("root : "+root1+" = "+gabungToken[root1]);
//out.println("tree :"+as[10]);
//====print token tree=====
while(j<as.length)
{
        if(as[j]==0)
        {
                j++;
        }
        else
        {
                //out.println("tree : "+j+" = "+as[j]);
                int d=as[j];
                if((d-1)==0)
                {
                        out.println("token : "+(d-1)+" = "+gabungToken[d-1]);//untuk ngeprint int di int
main()
                }
                out.println("token : "+d+" = "+gabungToken[d]);
                j++;
        }//j++;
}
j=1;//mulai dri 1 karena 0 diisi root
```

This code is used for read token from binary tree. The way for read the tree is use InOrder (left-root-right). In this code, call class MyBinaryTree to print the order of the tree.

```
package gabung;

import tree.Node;
public class MyBinaryTree
{
	Node root=null,baru;
	int [] temp1=new int[200];
	int j=1;
	public int getRoot()
	{
	return root.isi;
	}

	public void add(int isi,int tentuRoot)
	{
	baru = new Node(isi);
	if(root==null) root = baru;
	else insert(root,baru,tentuRoot);
	}

	public void insert(Node parent, Node baru,int tentuRoot)
	{
	if(baru.isi<= parent.isi)
	{
		if(parent.ki!=null)
			insert(parent.ki,baru,tentuRoot);
		else
		{
			parent.ki=baru;
		}
	}
	else //if(baru.isi>parent.isi)
	{
		if(parent.ka!=null)
		{
			insert(parent.ka, baru,tentuRoot);
		}
		else
		{
			//System.out.println(" Tambah " +baru.isi+ " di kanan node " +parent.isi);
			parent.ka=baru;
		}
	}
}
```

This code is used for save the tree and also access Node class. If the children value is smaller than the parent, the child will be on the left branch side of parent. If larger, the child will be on the right branches.

```
public int [ ] printInOrder(Node node)
{
        if(node !=null)
        {
                if(node==root)
                {
                        temp1[0]=node.isi;
                        //j=j+1;
                }
                printInOrder(node.ki);
                temp1[j]=node.isi;
                //j=j+1;
                printInOrder(node.ka);
                //j=j+1;
        }
        //else
                //System.out.println(" j="+j);
                //System.out.println(" j="+j+" = "+temp1[j]);
                j=j+1;
                return temp1;
        }

}
```

This code is used for read the tree data structure. The way of read the tokens is InOrder. Which is Left – Node – Right.

```
package tree;
public class Node
{
	public int isi;
	public Node ki,ka;
	public int i;
	public int tentuRoot;
	public int temp;
	public Node(int isi)
	{
		this.isi=isi;
		ki=null;
		ka=null;;
	}
	public Node()
	{
		isi=0;
		ki=null;
		ka=null;
		tentuRoot=0;
	}

	public int getIsi()
	{
		return isi;
	}
	public Node getKa()
	{
		return ka;
	}
	public Node getKi()
	{
		return ki;
	}
	public void setIsi(int i)
	{
		isi=i;
	}
	public void setKi(Node node)
	{
		ki=node;
	}

	public void setKa(Node node)
	{
		ka=node;
	}
}
```

This code is for make the node of tree. This code used to save the tokens, include root and the child Node. This class will be called by MyBinaryTree class.

4.4 Check Semantic of a program

```
//====== check semantic====
while(j<as length&&cekSemantic!=0)
{
	if(as[j]==0)
	{
		j++;
	}
	else
	{
		int d=as[j];//nyimpen token k brp
		j++;
		int ppp=d;
		//==========parse printf==========
		if(gabungToken[d].equals("printf")&&cekFor==0)
		{
			//out.println(pembatas);
			int bates=pembatas+d;
			while(d<bates&&hasilnya[d]!=null)
			{
				//out.println(hasilnya[d]);
				//out.println("d= "+d);
				if(hasilnya[d].equals("word"))
				{
					//out.println("bbb "+gabungToken[d]);
					hasilPrint=hasilPrint+" "+gabungToken[d];
				}
				else if(hasilnya[d].equals("operator"))
				{
					hasilPrint=hasilPrint+" "+gabungToken[d];
				}
				else if(hasilnya[d].equals("keyword")&&gabungToken[d].equals("%d"))
				{
					int cekTest=0;
					//out.println("aaaaaaaaaaaaaaa"+gabungToken[d]);
					keywordPersen[persen]=d;
					persen++;
					int test=d+1;
```

This code is used for translate the meaning of the program. In here, the code translate printf, so can print some words and operator inside of print. If found %d, this code also check is the variable that used available or not.

```
while(test < simpanBatasAkhir[ulang])
{
        //out.println("tesssssssss= "+hasilnya[test]);
        cek=0;
        if(hasilnya[test].equals("variable"))
        {
                out.println("ceeeeekTesss="+gabungToken[test]);
                cekTest++;
                int ga=0;
        //      =====check if variable has been declare or not======
                while(variable1[ga]!=null)
                {
                        if(variable1[ga].equals(gabungToken[test]))
                        {
                                cek=cek+1;
                                break;
                        }
                        ga++;
                }
                if(cek==0)//blm dideclare
                {
                        hasilPrint="variable "+gabungToken[test]+" blm dideclare";
                        hasilOutput[output]=hasilPrint;
                        hasilPrint="";
                        output++;
                        cekSemantic=0;
                        break;
                }
                break;
        }
        test++;
}
```

This code is used for check the variable inside of the printf word. This code also check is the variable has been declared or not. If not declared before, the program will be show error message.

```
//out.println("cccceeeeekkkkkk "+cekTest);
        if(cekTest==0)
        {

        hasilPrint="errorrr. tidak ada variable yang di print";

        cekSemantic=0;

        }

        else

        {

        int bbb=0;

        //out.println("ajajajaja");

        while(variable1[bbb]!=null)

        {

                //out.println("ajajajaja="+variable1[bbb]);

                if(variable1[bbb].equals(gabungToken[test]))

                {

                        //out.println("cocoookkk");

                        out.println("nilaaii 2= "+variable2[bbb]);

                        hasilPrint=hasilPrint+" "+variable2[bbb];

                        break;

                }
                bbb++;
                }
        }
        //out.println("cccceeeeekkkkkk "+cekTest);
```

This code will check the variable inside of the printf word. Is it has been declared or not. This code also save the value of a variable.

```
}
d++;
}
hasilOutput[output]=hasilPrint;
out.println("output= "+output);
output++;
hasilPrint="";
break;
}

//=========parse variable==========

//=========declare=================
else if(hasilnya[d].equals("keyword")&&gabungToken[d].equals("int"))
{
        out.println("declare int");
        out.println("var = "+gabungToken[d+1]);
        variable1[xyz]=gabungToken[d+1];
        variable2[xyz]=0;//awal smua diisi 0
        //hasilPrint=variable1[xyz]+" = "+variable2[xyz]+";";
        //hasilOutput[output]=hasilPrint;
        //out.println("output= "+output);
        //output++;
        xyz++;
        hasilPrint="";
        break;
}
//=========parse operator=========

//=========hitung=================
else if(hasilnya[d].equals("variable")&&hasilnya[d+1].equals("operator")||
hasilnya[d].equals("operator")&&hasilnya[d+1].equals("variable"))
{
        //out.println("cekkk operatorrr "+gabungToken[d]);
        int bates=pembatas+d;
        int ga=0;
```

This code is used for declare the variable. Every variable that declared will be saved into a cekDeclare variable. The variable and also will be given a default value 0.

```
while(variable1[ga]!=null)
{
        //out.println("varrr "+variable1[ga]);
        //out.println("varrr baru "+gabungToken[d]);
        if(variable1[ga].equals(gabungToken[d])||variable1[ga].equals(gabungToken[d+1]))
        {
                cek=1;
                break;
        }
        else
        {
                cek=0;
        }
        ga++;
}
//out.println("cek "+cek);
if(cek==0)//blm dideclare
{
        if(cek==0)
        {
                hasilPrint="variable "+gabungToken[d]+" blm dideclare";
        }
        hasilOutput[output]=hasilPrint;
        output++;
        cekSemantic=0;
        //break;
```

This code is used for checked is the variable already declare or not. If declared, the program will running. If not, the program will be stopped.

```
else
{
        int df=0;
        int [] nilaiSementara=new int[100];
        int jh=d;
        while(jh<simpanBatasAkhir[ulang])
        {
                //out.println("vkcdlljdj= "+gabungToken[d]+" , "+hasilnya[d]);
                int xh=0;
                if(gabungToken[jh].equals("=")&&hasilnya[jh+1].equals("number"))//a=5
                {
                        out.println("integerrr= "+gabungToken[jh+1]);
                        while(variable1[xh]!=null)
                        {
                                if(variable1[xh].equals(gabungToken[jh-1]))
                                {
                                        break;
                                }
                                xh++;
                                        }
        variable2[xh]=Integer.parseInt(gabungToken[jh+1]);
        //hasilPrint=gabungToken[jh-1]+" = "+variable2[xh];
        //hasilOutput[output]=hasilPrint;
        //output++;
        hasilPrint="";
        break;
}
jh++;
}
jh=d;
```

This code is used for store number into a variable declare. The number will be used for operating syntax. This code also check is the variable has been declared or not.

```
int xk=0;
int nilaiVariable=0;
int [] simpanNilai=new int[80];
while(jh< simpanBatasAkhir[ulang])
{
        if(gabungToken[jh].equals("="))
        {
        while(jh<simpanBatasAkhir[ulang])
        {

        if(hasilnya[jh].equals("variable"))
        {
                while(variable1[xk]!=null)
                {
                        if(variable1[xk].equals(gabungToken[jh]))
                        {
                                break;
                        }
                        xk++;
                }

        simpanNilai[nilaiVariable]=variable2[xk];

        NilaiVariable++;
}
else if(hasilnya[jh].equals("number"))
{
        simpanNilai[nilaiVariable]=Integer.parseInt(gabungToken[jh]);
        NilaiVariable++;
}
jh++;
}
nilaiVariable=0;
}
jh++;
        }
```

This code is used for save the value of token that used in operator. Writer use temporary array for save the value. The value will be take and save new value from operating syntax.

```
//out.println("nilaaaaaaaiiii 1 "+simpanNilai[2]);
int nilaiTotal=0;
nilaiVariable=0;
int xh=0;
int yy=0;
while(d<simpanBatasAkhir[ulang])
{
        if(hasilnya[d].equals("operator"))
        {
                if(gabungToken[d].equals("+"))
                {
                        yy=simpanNilai[nilaiVariable]+simpanNilai[nilaiVariable+1];
                        simpanNilai[nilaiVariable+1]=yy;
                        nilaiVariable++;
                        //break;
                        //out.println("yyyyyyyy= "+yy);
                }
                else if(gabungToken[d].equals("-"))
                {
                        yy=simpanNilai[nilaiVariable]-simpanNilai[nilaiVariable+1];
                        simpanNilai[nilaiVariable+1]=yy;
                        nilaiVariable++;
                        //break;
                        //out.println("yyyyyyyy= "+yy);
                }
        }
        d++;
}
xk=0;
if(yy!=0)
{
        while(variable1[xk]!=null)
        {
                if(variable1[xk].equals(gabungToken[ppp]))
                {
                        break;
                }
                xk++;
        }
        variable2[xk]=yy;
        out.println("sebeluuummmm= "+variable2[xk]);
        out.println("yyyyyyyy2= "+yy);
        }
}
```

This code is used for do operator + and - in the code. This code will read value of the variable. And saved new value for those variable.

```
//=======  == parse for=======
else if(gabungToken[d].equals("for"))
{
        //out.println("forrrrrrrrrrrrrrrrrrrrr");
        int gh=d;
        String []
cekUrutanFor={"keyword","symbol","variable","operator","number","symbol","variable","operator","nu
mber","symbol","variable","operator","symbol"};
        //out.println("urutaaaan = "+cekUrutanFor[2]);
        int cekUrut=0;
        //  ==========cek format for==========
        while(gh<simpanBatasAkhir[ulang])
        {
                //out.println("cekUrutan = "+cekUrutanFor[cekUrut]);
                //out.println("isiiiiii = "+hasilnya[gh]);
                if(!hasilnya[gh].equals(cekUrutanFor[cekUrut]))
                {
                        hasilPrint="salah format for !";
                        hasilOutput[output]=hasilPrint;
                        output++;
                        hasilPrint="";
                        cekSemantic=0;
                        break;
                }
                cekUrut++;
                gh++;
        }
//======  ======end checking format for==========
```

This code is used for check the format of For keyword. If the format false, will shown error message and program will be stopped. If true, the program will running the loop syntax.

```
int xk=0;
cekFor=1;
while(d<simpanBatasAkhir[ulang])
{
        if(hasilnya[d].equals("variable"))
        {
                //out.println("variableeeeeeeeeeeee");
                while(variable1[xk]!=null)
                {
                        if(variable1[xk].equals(gabungToken[ppp]))
                        {
                                break;
                        }
                        xk++;
                }
                if(gabungToken[d+1].equals("=")&&hasilnya[d+2].equals("number"))
                {
                        //out.println("variableeeeeeeeeeeee");
                        variable2[xk]=Integer.parseInt(gabungToken[d+2]);
                        //out.println("ddddddddddddddd= "+variable2[xk]);
                }
                else if(gabungToken[d+1].equals("<")&&hasilnya[d+2].equals("number"))
                {
                        batasFor=Integer.parseInt(gabungToken[d+2]);
                        //out.println("sssssssssssssssssss= "+batasFor);
                }
        }
        d++;
}
}
```

This code is used for save the started number and ended number of loop. The start and ended number will be saved in variable. This variable will be used to execute how many program will do loop.

```
else if(gabungToken[d].equals("printf")&&cekFor==1)//masuk for
{
        //out.println("masukkkk forrr");
        int bates=pembatas+d;
        String [] printt=new String[200];
        int ok=d;
        String simpanIsi="";
        while(ok<simpanBatasAkhir[ulang])
        {
                if(hasilnya[ok].equals("word")||hasilnya[ok].equals("operator"))
                {
                        simpanIsi=simpanIsi+gabungToken[ok]+" ";
                }
                ok++;
        }
        out.println("hassisiill for= "+simpanIsi);
        while(d<bates&&hasilnya[d]!=null)
        {
                int xk=0;
                if(hasilnya[d].equals("word")||hasilnya[d].equals("operator"))
                {
                        while(variable1[xk]!=null)
                        {
                                if(variable1[xk].equals(gabungToken[d]))
                                {
                                        break;
                                }
                                xk++;
                        }
                        out.println("batassss for= "+batasFor);
                        out.println("mulaiiiiiiii "+variable2[xk]);
                        for(int po=variable2[xk];po<batasFor;po++)
                        {
                                hasilOutput[output]=simpanIsi;
                                output++;
                        }
                        break;
                }
                //out.println("batassss for= "+batasFor);
                d++;
        }
        cekFor=0;
        }
        }//j++;
```

This code is used for execute printf inside of loop. CekFor is used as a sign if the print is inside of loop. If cekFor has value 1, that means the printf is inside of loop

```
j=1;
out.println("===New tree==");
ulang++;
}
output=0;
//===hasil output===//
out.println("===Output==");
while(hasilOutput[output]!=null)
{
        out.println(hasilOutput[output]);
        output++;
}
}
}
%>
</textarea>
</td>
</tr>
</table>
</body>
</html>
```

This code will show the final output of user source code input. The final output will be shown in different text area with the parsing steps. The error message also will be shown in this code.

5.2 Testing

To test the program works properly, writer make a test. This test consist of 26 point for total point if the program is work properly. In this test,every item will be tested, if the result is match , then it will get 2 point, if it doesn't match then it will get 1 point, and if fail will get 0 point.

No.	Action	Result	Score
1	Run the program in browser	Program running	2
2	Choose the tutorial using radio button	Show the C code tutorial that chosen	2
3	Edit the C tutorial	Tutorial can be edited	2
4	Show the regex matcher pattern result	Show the regex result	2
5	Divide the source code into token and do tokenization	Show token and tokenization result	2

6	Make a tree from tokens	Tree created	2
7	Check semantic for code	Check the for format and show error message if wrong	2
8	Check variable	Show error message if variable not declared before	2
9	Parse printf syntax	Show the result, but only can print word and operator	1
10	Parse operator syntax	Do the operator, but only can do + and -	1
11	Parse loop (for) operator	Can show the result but only can do first line and can't do double loop	1
12	Show the error message if input code wrong	Show error message	2
13	Show the final result in different textarea	Show the final result	2
		Total testing point	**23**
		Maximum point	**26**
		Percentage	**88,00%**

Table 5.1 Testing Table

5.3 Application Interface

The interface of this program :

The first page is the choices of the tutorial. User can choose one from five tutorial that available. And click button tutorial to go the source code of the program.

Figure 5.1 Choose Tutorial Interface

In the next page, the source code of tutorial that selected before will be shown on text area. In this page, user can edit the source code. After done with editing program, user can click input button

to proceed the code.

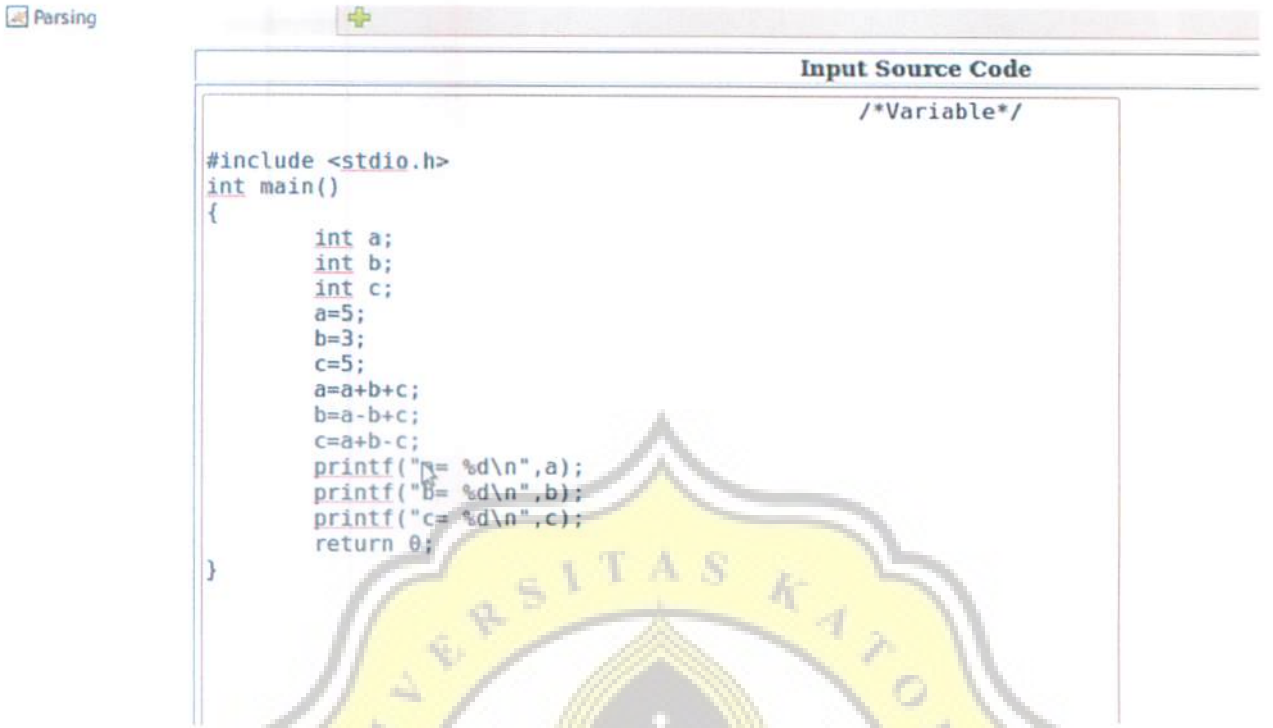

Figure 5.2 Show the Tutorial

In output2 jsp, all of parsing process result will be shown here. If the program error, the compiler will shown error message. If true, the compiler will shown the result.

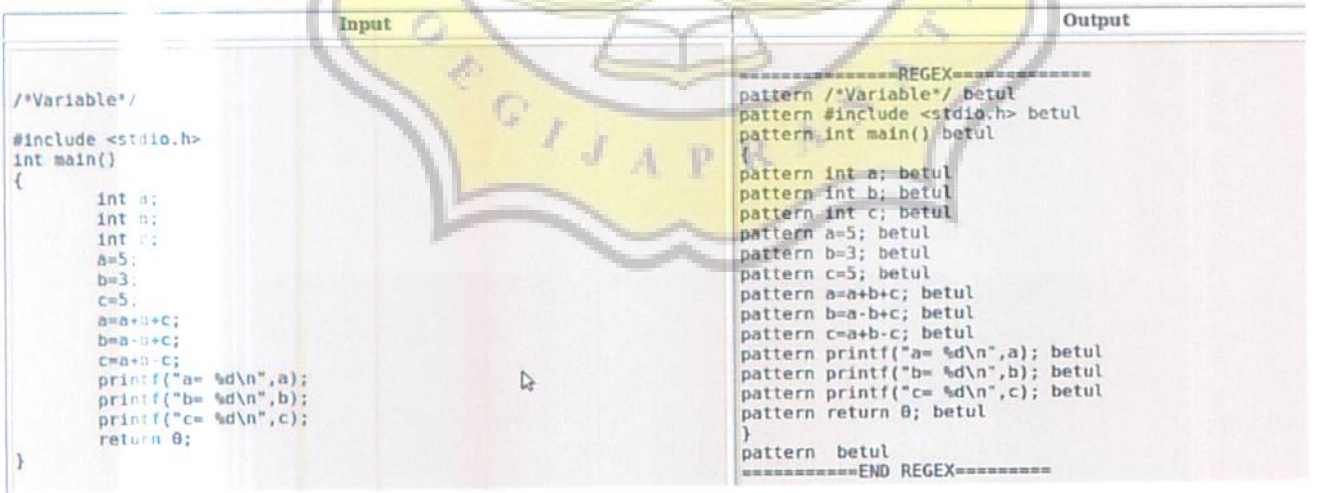

Figure 5.3 Show Regex Result from Source Code

After regex process, the line of syntax will be divided into token. Each token will be categorized into tokenization. All of the token and tokenization will be shown here.

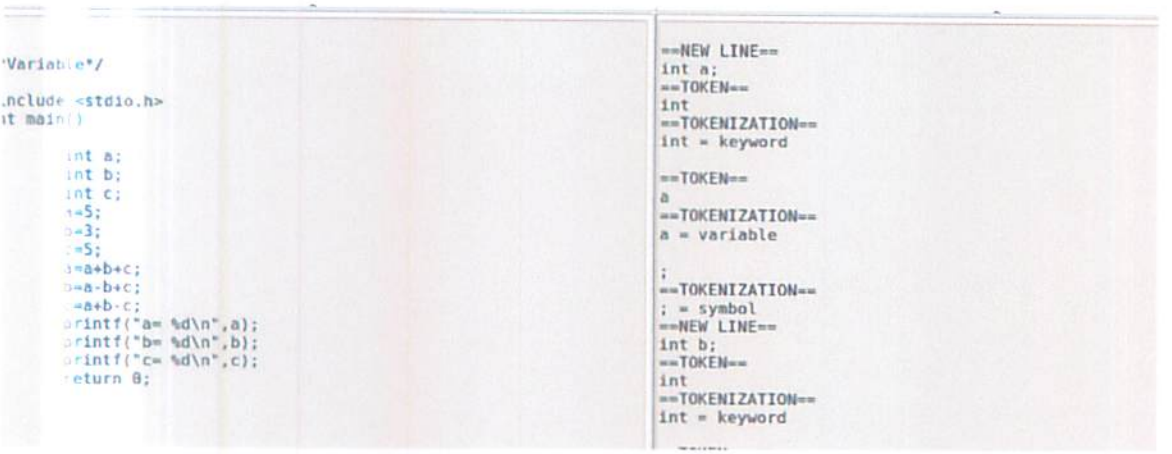

Figure 5.4 The Code Divided into Token and Categorized use Tokenization

This image will show the syntax analyzer result. The tree that created will be show here. Including the root and the child node.

Figure 5.5 Show the Tree of Tokens

And Finally, the final result of the parsing process will be shown in Hasilnya textarea. If there a false syntax while checking the semantic, will be show here too. The result is separated from parsing process.

Figure 5.6 Final Output