TOWARD PLAGIARISM DETECTION IN JAVA PROGRAMS

BY

DONGCHI WANG

Bachelor of Arts
Liaoning University
Shenyang, Liaoning
People's Republic of China
1990

Law degree
Law and Politics University of China
Beijing
People's Republic of China
1992

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May 2002

TOWARD PLAGIARISM DETECTION IN

JAVA PROGRAMS

Thesis Approved:

M. Samadzadeh - H.

Thesis Advisor

J Chandler

Blayne E Mayfield

Timothy J. Pettitt

Dean of the Graduate College

PRAFACE

Similarity was inspected among a group of Java programs based on comparison of program structures so that plagiarism could be detected. Research was conducted on existing approaches to plagiarism detection. Ottenstein was the first researcher who suggested checking program plagiarism automatically. Researchers had been making progress in the efficiency and accuracy of detection. After reviewing similar work done previously, carefully analyzing the attributes of Java as an Objected-Oriented programming language, and considering typical program plagiarism practices, this study utilized a new method to inspect programs and detect similarities. As a result of this study, a tool called JPD was constructed to detect program plagiarism among Java programs.

The methods or algorithms already existing in the literature were investigated and compared. Instead of using a text-based statistics-oriented detection method, this study focuses on program structure. JPD, which consists of Scanner and TSM (Token Stream Matcher), was implemented in C and Unix utility Flex. There are three passes for Scanner to scan a Java program and translate it into a token stream. Then, TSM, which utilizes dynamic common longest string-matching algorithm, compares two token streams from a pair of Java programs. Tested by using real class assignments, JPD helps to detect similar programs by comparing a group of Java programs in pairs.

ACKNOWLEDGEMENTS

I would like to extend my sincere appreciation to my thesis advisor, Dr. Mansur H. Samadzadeh, for giving me valuable guidance, encouragement, wisdom, and patience throughout my graduate studies at Oklahoma State University. My special thanks are also extended to Dr. John P. Chandler and Dr. Blayne E. Mayfield for their valuable help and participation while serving as members of my committee.

My special gratitude goes to my husband, Bo Deng, for his great support and trust in the entire journey of my graduate studies. I am also grateful to my mother, Junjiang Chi, and father, TaiShun Wang. They have contributed to my every single achievement including my educational achievements.

I also appreciate the greatest love from my sisters, Dongbai Wang, who gave me the great opportunity to study and enjoy life in America, and Dongqing Wang, who took care of my mom through her recovery.

TABLE OF CONTENT

Chapter Page

CHAPTER I

INTRODUCTION

Computer science as a field of study attracts more and more students as the need for information technology increases. However, programming generally gives students a hard time. This is especially true about entry-level students because of the precise, strict and concise syntax of programming languages, and the educational background requirements for computing which are logical design skills and a solid mathematical foundation.

Academic plagiarism and its detection have been perplexing problems in all majors including computer science. Program plagiarism in computer science normally refers to copying all or part of a program from some source and submitting the copy as one's own work. This includes collaborating and submitting similar work.

The negative effect of plagiarism is explained briefly by Harris [Harris 94] as follows.

> Students who plagiarize cheat themselves by refusing their own education and cheat the original authors by claiming the work to be theirs.

Plagiarism is a form of cheating. Any form of cheating is to be condemned, and plagiarism is no exception. Theft of intellectual work by copying that work is still theft, and should be treated as such.

Schools and instructors take every action attempting to stop the phenomenon of plagiarism. Detection techniques, administrative procedures, and penalties vary greatly. It appears that in introductory level courses, where not all students are skillful enough to make their own programs work, program plagiarism is more common. An efficient way to deal with academic dishonesty is to monitor program plagiarism by an automatic detection system.

Theoretically, similarity detection, considered as the problem of equivalence of two programs, is a decision problem, i.e., a computational problem for which every specific instance can be answered "yes" or "no". In this sense, general similarity detection is an unsolvable decision problem [Martin 97]. However, the practical problem itself is more complex than just saying "yes" or "no". If we say two programs are similar, that suggests they may not be identical syntactically, the semantic structure of the programs must also be taken into consideration.

Historically, a lot of effort has been put into plagiarism detection for procedural programming languages such as C and Pascal. A few systems have been built for Object-Oriented programming language such as C++, Java, and SmallTalk, but the underlying algorithms have not been published as of this writing.

The purpose of this research was to build a plagiarism detection system specific to Java programs. The main goal of this thesis was to design and implement a detection tool to help discourage academic dishonesty involving program plagiarism. The system is called JPD, which stands for Java Plagiarism Detection. It targets introductory Objected-Oriented programming classes at the university level. Data to test the tool was taken from actual classes and real assignments. To protect the students' privacy, their names have

been suppressed. JPD was implemented in C together with some Unix utilities such as Flex. The tool works under the Unix system.

The organization of this thesis is as follows. Chapter II provides a literature review of similar work. Chapter III describes the design and implementation issues. Chapter IV explains a sample result and its analysis. Chapter V discusses the summary and future works. Some sample results, some sample programs used for testing JPD, the source code of JPD, and the shell scripts to run JPD are attached as appendices.

CHAPTER II

LITERATURE REVIEW

2.1 General Introduction

There have been a number of attempts to detect program similarities automatically. Most of the efforts have been from academic institutes or universities.

In the 70's and early 80s, work on plagiarism detection was quite popular. The algorithms were very straightforward. Basically, they checked for textual similarities, specifically for the frequencies of occurrences of statements or operators, and the efforts focused on specific programming languages.

From the middle 80s up to now, work on plagiarism detection has been generally less popular. Detection systems have been getting generally more powerful and complex. Most of them can handle multiple programming languages such as C, C++, and Java.

Currently, web techniques become increasingly more popular, and there are a number of publicly available online genetic program plagiarism detection systems. Among them, the famous ones are MOSS from UC Berkeley and Jplag from University of Karlsruhe, Germany. They work on most high level programming languages. The two systems are further discussed in Section 2.2.2.

2.2 Methods Review

2.2.1 Earlier Methods

Ottenstein addressed the problem of automatic plagiarism detection [Ottenstein 76] by utilizing the four basic software science parameters suggested by Halstead [Halstead 77] to measure the sameness of Fortran programs. The four basic software science parameters are listed below.

(1) The number of unique operators n1.

(2) The number of unique operands n2.

(3) The total number of occurrences of operators N1.

(4) The total number of occurrences of operands N2.

With the assumption that all programs to be considered are well written, Halstead came up with several relationships and properties involving the four basic counts. For instance, he defined program length and vocabulary as follows.

vocabulary $n = n1 + n2$

program length in tokens $N = N1 + N2$

In Ottenstein's plagiarism detection system, programs with similar number of N1, N2, n1, and n2 are suspected to be involved in plagiarism. This algorithm has some shortcomings [Ottenstein 76]. The results cannot always be reliable and precise since it can only detect cosmetic changes. It does not take semantics of programs into consideration. Even a little disguise effort, such as putting in redundant operators and operands into the code, can crash the detection system, i.e., it can cause the system not to detect plagiarism. Regardless, this detection system opened the curtain on plagiarism detection work.

In 1980, Grier made a plagiarism detection system called Accuse for Pascal programs [Grier 80]. He was inspired by Ottenstein's algorithm and actually moved the research forward. He started from the four basic software science parameters and calculated twenty measurements. But as Grier himself indicated [Grier 80], his system has some drawbacks too. Although it was able to handle more sophisticated ways of intentionally changing similar code, it is still a text-based detection system with no program structure considered.

2.2.2 Contemporary Methods and Tools

From the late 80's up to middle 90's, a number of advanced detection systems appeared in the literature. Detection systems had been getting generally more complex and they worked on multiple programming languages. Some of the more recent detection tools are still under testing. Generally, they catch plagiarism with more precision. Some of them utilize the web to interact with the clients or users. Some of the popular contemporary tools are briefly described below.

- Wen-yu Fu [Fu 86] developed a software tool that detects plagiarism in C programs. This system collects and analyzed measurable properties of C programs to check for similarities. It uses flowchart-like diagrams to show the calling relationship among functions. It also considers some common changes plagiarists could make to disguise the traces of copying, such as interchanging "for" statement and "while" statement, or "switch case" statements and "if ... else if..." statements. For example, if there are a number of case statements in a program and the other program has a sequence of if statements, the detector will consider this a similarity. It seems that this is the first

reported attempt at trying to catch program similarity according to the analysis of the semantic structure of programs.

- SIM from Wichita State University [Gitchelll and Tran 99]: SIM is a utility for detecting similarity in computer programs. It utilizes the string-matching technology originally developed to detect similarity of DNA strings [Huang et al 90] [Myers and Miller 88] to compare structural similarity between two C programs. Compared to the earlier methods, which just detected textual matches and ignored the possibility of function reorder, variable name changing, and redundant components, SIM made big progress. However, to handle the common copy strategy of function reorder, it takes every permutation of the orders to compare with the target code. It is neither time efficient nor precise since a random reordering of the functions may not be in the same sequence of the calling order as the program is actually executed. Hence, under this system, two C programs with high similarity rate as judged by this system may not always have something to do with plagiarism.

- Jplag from University of Karlsruhe in Germany [Malpohl 00]: Jplag is an online system from University of Karlsruhe in Germany. It checks for similarities among a large set of programs. Not only can it detect program syntax similarities but also program structural similarities. It works on C, C++, Java, and Scheme. It takes submission from users and returns some statistics.

- MOSS from UC Berkeley [MOSS 00]: MOSS stands for a Measure of Software Similarity. It is a generic system for detecting software plagiarism. It works on C, C++, Java, Pascal, Ada, ML, Lisp, and Scheme programs. It was developed by UC Berkeley in 1994 and claimed to be a significant improvement over other cheating detection

algorithms known. The algorithm of MOSS has not been made public intentionally since "while there is a big difference between a good cheating detection algorithm and a bad one, all such algorithms can be fooled if one knows how they work. It is best if we don't say too much here about the ideas behind Moss." [MOSS 00]. The user has to submit a request for an account and only the instructors will be granted access. It also requires writing a large shell script to submit programs to be tested for plagiarism.

CHAPTER III

DESIGN AND IMPLEMENTATION ISSUES

3.1 Java as an OO Programming Language

Java Plagiarism Detector (JPD) is the plagiarism detection system designed and implemented in this thesis. Since JPD is specific to Java programs, it is necessary to briefly mention the characteristics of Java as a programming language. Java is a platform-neutral, Object Oriented programming language that provides a large number of predefined library classes to simplify common programming tasks [Litwak 99].

The basic concept in Java is class. All executable code must be contained within a class. Unlike C/C++, Java has no global functions that are defined outside of any classes or friend functions that are defined in other classes [Litwak 99]. Conceptually, "class" is similar to "structure" in C/C++. They differ in that a structure does not have method members and the data members of a structure are accessible to the entire program in a C/C++ program, while a class in Java has method members and all data members are accessible only to that class by default. Members in a Java class may have different accessibilities, such as private, protected, or public. This is an important feature of Objected Oriented programming languages that is called data encapsulation. In Java, inheritance allows classes to be written as extensions of other classes. A Java program may have several classes that are organized by having "is a" or "has a" relationships. A

well-designed Java program is arguably hard to plagiarize successfully without understanding what the Java program is doing and how it is organized, since the similarity cannot be easily disguised if the program structure and organization are not significantly modified.

Like programs written in other programming languages, a Java program consists of fundamental statements that embody the programming style and control flow of the program. Java as a specific programming language has a number of unique keywords. Statements in Java can be classified as follows [Palmer 00].

1. Declaration, assignment, and basic operation statements that are followed by a semicolon:

1) Assignment of a value to a variable, e.g., V = 4;
2) Use of increment and decrement operators, e.g., I ++;
3) Method invocation, e.g., System.out.println ("Hello, world.");
4) Creation of a new object, e.g., InputStreamReader stdin = new InputStreamReader (System.in);

2. Various sorts of control statements:

1) if ... else
2) while
3) do-while
4) for
5) break
6) continue
7) switch-cases
8) conditional statement operator, ? :
9) return

3. Exception handling statements (keywords are unique to Java):

1) try-catch block
2) finally
3) throw
4) throws

4. The "synchronized" modifier that is unique to Java and is used to protect

simultaneous accesses to a critical section.

5. A block of statements that is a series of statements enclosed within curly brackets.

6. Variable declaration, e.g., int Int1;

7. A labeled statement that consists of a label used to identify the statement, a colon, and a statement.

3.2 Sample Java Program

What follows is a sample Java program from an introductory Java programming class that generates random license plate numbers and tag names for a vehicle. This program was written by a student in an introductory computer science class. A plagiarized version of this program appears in Section 3.3.2, followed by a discussion of the similarities and modifications.

```
1)     //=======================================
2)     // Program: Sample_Program 1
3)     //
4)     // Author: Sample_Student1
5)     //=======================================

6)     import java.io.*;
7)     import java.util.*;

8)     public class Sample1
9)     {
10)    //-----------------------------------------------------
11)    //This method generates a random license plate number
12)    //that contains letters and numbers
13)    //-----------------------------------------------------
14)
15)    public static String tagNumber()
16)    {
17)              Random tag = new Random();
18)              int i;
19)              String tagnum = "";
20)              String alpha = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
21)              String tagletters = "";

22)    //  Generate random characters

23)        for (int charcount = 0; charcount < 3; charcount++)
24)        {
25)                       i = Math.abs (tag.nextInt()) % 26;
26)                       tagnum = tagnum + alpha.charAt(i);
27)        }
```

```
28)                 tagnum = tagnum + '-';

29)     //  Generate random numbers

30)         for (int count = 0; count < 3; count++)
31)         {
32)                   i = Math.abs (tag.nextInt()) % 10;
33)                   tagnum = tagnum + i;
34)         }
35)                    return tagnum;
36)     }

37)     //-----------------------------------------------------
38)     // This method generates a nine-digit random driver's
39)     // license number
40)     //-----------------------------------------------------

41)     public static String licenseNumber()
42)     {
43)                 Random license = new Random();
44)                 int i;
45)                 String licensenum = "";
46)                 for (int count = 0; count < 9; count++)
47)                 {
48)                     i = Math.abs (license.nextInt()) % 10;
49)                     licensenum = licensenum + i;
50)                 }
51)                 return licensenum;
52)     }

53)     //-----------------------------------------------------
54)     // This method determines the cost of a customer's tag
55)     // depending on how old his/her car is
56)     //-----------------------------------------------------

57)     public static float tagCost(int year)
58)     {
59)            GregorianCalendar cal = new GregorianCalendar();
           int systemyear = cal.get(Calendar.YEAR);
                   int caryear = (systemyear - year);
60)                 float costoftag = 0;
61)                  if (caryear >= 0 && caryear <= 3)
62)                                costoftag = 85;
63)                 else if (caryear >= 4 && caryear <= 7)
64)                                 costoftag = 75;
65)                  else if (caryear >= 8 && caryear <= 11)
66)                                costoftag = 55;
67)                  else if (caryear >= 12 && caryear <= 15)
68)                                costoftag = 35;
69)                  else
70)                                costoftag = 15;
71)                  return costoftag;
72)     }

73)     //-----------------------------------------------------
74)     // This method takes a customer's name and converts
75)     // it into a string that has only one white space in
76)     // between its tokens
77)     //-----------------------------------------------------

78)     public static String minimizeWhitespace(StringTokenizer t)
79)     {
80)            String namestring = "";
```

```
81)          namestring = (t.nextToken());
82)          while (t.hasMoreTokens())
83)          namestring = namestring + " " + (t.nextToken());
84)          return namestring;
85)    }

86)    //-------------------------------------------------
87)    // This method calculates a customer's age based on the
88)    // current calendar and the driver's birth date.
89)    //-------------------------------------------------

90)    public static int calcAge(String date)
91)    {
92)          GregorianCalendar cal = new GregorianCalendar();
93)          int systemmonth = cal.get(Calendar.MONTH);
94)          systemmonth = systemmonth + 1;
95)          int systemdate = cal.get(Calendar.DATE);
96)          int systemyear = cal.get(Calendar.YEAR);

97)        int drivermonth = Integer.parseInt(date.substring(0,2));
98)        int driverdate = Integer.parseInt(date.substring(3,5));
99)        int driveryear = Integer.parseInt(date.substring(6));

100)       int ageinyears = (systemyear - driveryear);
101)     if (systemmonth <= drivermonth && systemdate < driverdate)
102)                       ageinyears = ageinyears - 1;
103)       return ageinyears;
104)   }

105)   public static void main(String[] args) throws IOException
106)   {
107)    BufferedReader stdin = new BufferedReader(new InputStreamReader
(System.in));
108)           String driver = stdin.readLine();
109)           String drivername = "";
110)           String requesttype = "";
111)           String requesttypeout = "";
112)           String make = "";
113)           String style = "";
114)           String vehicleyear = "";
115)           String driverdatein = "";
116)           int pass = 0;
117)           int tagcostout = 0;
118)           int numoftags = 0;
119)           int numoflicenses = 0;

120)   //--------------------------------
121)   // Read input while input exists
122)   //--------------------------------

123)         while (driver != null)
124)        {
125)               StringTokenizer tokenizer = new StringTokenizer (driver);
126)                requesttype = (tokenizer.nextToken());

127)               //-------------------------------------
128)               // If it is a tag request, perform tag
129)               // methods
                   //-------------------------------------
130)
131)              if (requesttype.equalsIgnoreCase("tag"))
132)              {
133)                        make = (tokenizer.nextToken());
```

```
134)                          style = (tokenizer.nextToken());
135)                          vehicleyear = (tokenizer.nextToken());
136)                          pass = Integer.parseInt(vehicleyear);
137)
138)                     while (tokenizer.hasMoreTokens())
139)                      drivername = drivername + " " + (tokenizer.nextToken());
140)                     StringTokenizer tagnametokenizer = new StringTokenizer
(drivername);
141)                          System.out.print(requesttype.toUpperCase() + ": ");
142)                          tagcostout = (int) tagCost(pass);
143)                          System.out.print("$" + tagcostout + " ");
144)                          System.out.print(tagNumber() + " " + make + " " +
style + " " + vehicleyear + " -- ");
145)                     System.out.println(minimizeWhitespace(tagnametokenizer));
146)                          numoftags = numoftags + 1;
147)                          drivername = "";
148)                  }
149)    //-------------------------------------------------
150)    // If it is a license request, perform license
151)    // methods
152)    //-------------------------------------------------

153)               else
154)               {
155)                   driverdatein = (tokenizer.nextToken());
156)                   while (tokenizer.hasMoreTokens())
157)                     drivername = drivername + " " + (tokenizer.nextToken());
158)                   StringTokenizer licensenametokenizer = new StringTokenizer
(drivername);
159)                   requesttypeout = requesttype.substring (0,3);
160)                   System.out.print(requesttypeout.toUpperCase() + ": $5 ");
161)                   System.out.print(licenseNumber() + " " + driverdatein + "
" + calcAge(driverdatein) + " -- ");
162)                System.out.println(minimizeWhitespace(licensenametokenizer));
163)                     numoflicenses = numoflicenses + 1;
164)                     drivername = "";
165)                }
166)                     driver = stdin.readLine();
167)          }
168)       //-------------------------------------------------
169)       // If no input is given, print a warning message
170)       //-------------------------------------------------

171)       System.out.println();
172)       if (numoftags == 0 && numoflicenses == 0)
173)                    System.out.println("NO INPUT GIVEN!!");
174)       else
175)          //-----------------------------------------------
176)          // Print total number of tags/licenses issued
177)          // in this run
178)          //-----------------------------------------------
179)      {
180)        System.out.println("Number of tags issued: " + numoftags);
181)        System.out.println("Number of licenses issued: " + numoflicenses);
182)       }
183)     }
184)   }
```

3.3 Problem Analysis

As the Internet technology has grown, Java as a platform independent programming language is getting increasingly popular. Most universities and colleges list Java as a required introductory course for Computer Science majors. These courses typically cover basic programming skills and objected-oriented programming concepts. The programming assignments designed by instructors are usually short and simple, and sometimes the Java class names or even the method names are given to the students. Since the students are new to programming, plagiarism is known to happen in these classes.

3.3.1 Typical Plagiarism Practices in Student Programs

It is necessary to list some common practices encountered in program plagiarism. Since JPD is designed for introductory programming classes, only basic plagiarism strategies are concerned, such as variable renaming, adding redundant statements or method reordering. Specifically, JPD catches the following cases.

(1) Variables Renaming: This is the most common action used to disguise program plagiarism. It is also the easiest one. By the assistance of a text editor, an individual can easily find and replace any specific word or sequence of words.

(2) Methods Reordering: This approach does not need any understanding of programming. Normally, students simply cut and paste some methods and try to reorder the original sequence of methods. It is quite vulnerable by human inspection, but it can easily escape from some Unix utilities such as diff.

(3) Interchanging Equivalent Statements: Some compound statements can be substituted by others, such as "if … else if" statements and switch-case statements, or "while" loops and "for" loops. For example,the following statement,

```
Switch (i)
    {
                case 0:  …
                case 1:  …
                case 2:  …
                default: …
    }
```

can be easily substituted by the following "if…else if" statement

```
if (i == 0)
…
else if (i == 1)
…
else if (i == 2)
…
else
…
```

(4) Redundant Variables Declared or Additional Methods Added: Some students may try to declare more variables without referencing them later in the program. Or, they may put more method definitions without calling them in the program. Once the program compiles, these modifications would not do anything to the program. This can easily eliminate the effect of changes to some text-based detector.

(5) Combination or Decomposition of Statements: Students with more programming knowledge may substitute a sequence of function calls by a nested function calls or vice versa.

(6) Adding More Comments: By adding more comments, the original code would not be changed but the size of the modified program may be significantly different from the original one.

Certainly, students may use more complex methods to disguise the trace of copying, such as splitting a method into several methods or even redesigning the program. However, it can be argued that most people attempting to plagiarize would be either not capable of modifying a program in that way or they are in such a hurry that there is not enough time to modify a program significantly. Hence usually not much time is spent eliminating the effect of changes. Also, presumably a student intelligent enough to take out all traces of plagiarism would not need to copy from others in the first place.

3.3.2 Sample Plagiarized Program

This following is a Java program plagiarized from the original version that appeared in Section 3.2. The plagiarized version was written by a student in an introductory computer science class.

```
1)      //=======================================
2)      // Program: Sample_Program2
3)      //
4)      // Author: Sample_Student2
5)      //=======================================

6)      import java.io.*;
7)      import java.util.*;

8)      public class sample2
9)      {

10)         public static void main(String[] args) throws IOException
11)         {

12)          InputStreamReader ir = new InputStreamReader(System.in);
13)          BufferedReader stdin = new BufferedReader(ir);

14)                         int Ps = 0;
15)                         int numoflicenses = 0;
16)                         int TcostOut = 0;
17)                         int numoftags = 0;
18)                         String Drname = "";
19)                         String RTout = "";
20)                         String make = "";
21)                         String style = "";
22)                         String Vyear = "";
23)                         String Rtype = "";
24)                         String driverdatein = "";
25)                         String driver = stdin.readLine();
26)                 /*
```

```
27)                     process input until hit the end of the string
28)                */
29)      while (driver != null)
30)      {
31)        StringTokenizer tokenizer = new StringTokenizer (driver);
32)        Rtype = (tokenizer.nextToken());
33)
34)
35)            /*
36)              Test if it is a tag request, if yes then perform
37)              tag methods
38)               */
39)            if (Rtype.equalsIgnoreCase("tag"))
40)            {
41)                make = (tokenizer.nextToken());
42)                style = (tokenizer.nextToken());
43)                Vyear = (tokenizer.nextToken());
44)                Ps = Integer.parseInt(Vyear);
45)                while (tokenizer.hasMoreTokens())
46)                Drname = Drname + " " + (tokenizer.nextToken());
47)          StringTokenizer tagnametokenizer = new StringTokenizer (Drname);
48)                System.out.print(Rtype.toUpperCase() + ": ");
49)                 TcostOut = (int) tagCost(Ps);
50)                 System.out.print("$" + TcostOut + " ");
51)                 System.out.print(tagNumber() + " " + make + " " + style + "
" + Vyear + " -- ");
52)                 System.out.println(minimizeWhitespace(tagnametokenizer));
53)                 numoftags = numoftags + 1;
54)                 Drname = "";
55)                           }
56)                /*
57)                 Otherwise if it is a license request, manipulate
58)                  the license methods
59)                */
60)          else
61)          {
62)            driverdatein = (tokenizer.nextToken());
63)            while (tokenizer.hasMoreTokens())
64)            Drname = Drname + " " + (tokenizer.nextToken());
65)            StringTokenizer licensenametokenizer = new StringTokenizer
(Drname);
66)            RTout = Rtype.substring (0,3);
67)            System.out.print(RTout.toUpperCase() + ": $5 ");
68)            System.out.print(licenseNumber() + " " + driverdatein + " "
69)     + calcAge(driverdatein) + " -- ");
70)            System.out.println(minimizeWhitespace(licensenametokenizer));
71)            numoflicenses = numoflicenses + 1;
72)            drivername = "";
73)              }
74)              driver = stdin.readLine();
75)            }
76)            System.out.println();
77)            if (numoftags == 0 && numoflicenses == 0)
78)            System.out.println("NO INPUT GIVEN!!"); //give warning if no input
79)          else
80)          {
81)                //otherwise print information
82)            System.out.println("Number of tags issued: " + numoftags);
83)            System.out.println("Number of licenses issued: " + numoflicenses);
84)          }
85)           }
```

```
86)             /*
87)               To determines the cost of the customer's tag
88)                  */

89)           public static float tagCost(int year)
90)         {
91)             float Ctag = 0;
92)             int Syear = Calendr.get(Calendar.YEAR);
93)             int Cyear = (Syear - year);

94)             GregorianCalendar Calendr = new GregorianCalendar();
95)             if (Cyear >= 0 && Cyear <= 3)
96)                       Ctag = 85;
97)             else if (Cyear >= 4 && Cyear <= 7)
98)                       Ctag = 75;
99)             else if (Cyear >= 8 && Cyear <= 11)
100)                      Ctag = 55;
101)            else if (Cyear >= 12 && Cyear <= 15)
102)                      Ctag = 35;
103)            else
104)                      Ctag = 15;
105)            return Ctag;
106)        }
107)         /*
108)            To gernerates a random license plate number that
109)            contains letters and numbers
110)         */
111)         public static String tagNumber()
112)        {
113)                    String Tnum = "";
114)                    int k;
115)                    int i;
116)                    String Aph;
117)                    String Tletter = "";

118)                    Random tag = new Random();
119)                    Aph = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
120)            for (int charcount = 0; charcount < 3; charcount++)
121)                             /*  Generate random characters */
122)            {
123)                        i = tag.nextInt();
124)                        i = Math.abs (tag.nextInt()) % 26;
125)                        Tnum += Aph.charAt(i);
126)             }
127)               Tnum = Tnum + '-';

128)               for (int count = 0; count < 3; count++)
129)                 /*  Generate random numbers */
130)              {
131)                    i = tag.nextInt();
132)                    i = Math.abs (tag.nextInt()) % 10;
133)                    Tnum = Tnum + i;
134)              }
135)              return Tnum;
136)          }

137)            /*
138)              To generates a nine digit random driver's license
139)              number
140)             */
141)    public static String licenseNumber()
142)    {
143)                  int j;
```

```
144)                    String Lnum = "";

145)                    Random license = new Random();
146)                    for (int count = 0; count < 9; count++)
147)                   {
148)                            j = license.nextInt();
149)                            j = Math.abs (license.nextInt()) % 10;
150)                            Lnum = Lnum + j;
151)                    }
152)                    return Lnum;
153)    }

154)              /*
155)                Convert customer's name which has mutiple spaces
156)                in between into a string
157)                that has only one white space in between
158)             */
159)       public static String minimizeWhitespace(StringTokenizer n)
160)       {
161)                String Name = "";
162)                Name = (n.nextToken());       /* get the first name */
163)                while (n.hasMoreTokens())
164)                   Name = Name + " " + (n.nextToken());  /* get     the last
name*/
165)                return Name;
166)    }

167)              /*
168)                     Driver's age calculation
169)               */
170)       public static int calcAge(String date)
171)       {
172)              int Smonth = Calendr.get(Calendar.MONTH);
173)              GregorianCalendar Calendr =new GregorianCalendar();

174)              int Syear = Calendr.get(Calendar.YEAR);
175)              Smonth = Smonth + 1;
176)              int Sdate = Calendr.get(Calendar.DATE);
177)              int Drmonth = Integer.parseInt(date.substring(0,2));
178)              int Dryear = Integer.parseInt(date.substring(6));
179)              int Drdate = Integer.parseInt(date.substring(3,5));

180)              int Agyears = (Syear - driveryear);
181)              if (Smonth <= Drmonth && Sdate < Drdate)
182)                   Agyears - = 1;
183)              return Agyears;
184)         }
185)    }
```

Although these two programs (the original version in Section 3.2 and the plagiarized version above) are not exactly the same, they are very similar. The second one varies from the first one only in the following ways.

(1) Variable Renaming. Most variables have been renamed in this program. This is the most expected change students could make.

(2) Method Reordering . The original program ends with the main() method with all other member methods ahead of it. The second version starts with the main() method followed by other member methods in a varied order. This modification can easily defeat most text-based matching detectors.

(3) Variable Reordering and Additional Variable Declarations. Not only did the second one modify the order of variable declarations, but also put in some useless variables. In Sample 2, at line 114, the integer variable k is declared but never referred to later in that module.

(4) Decomposition of Nested Methods Calls. In Sample 1, there is a nested methods call at line 107 in the main() method,

```
107)   BufferedReader stdin = new BufferedReader(new InputStreamReader
(System.in));
```

The line is decomposed as the two method calls in line 12 and line 13 in Sample 2:

```
12)     InputStreamReader ir = new InputStreamReader(System.in);
13)     BufferedReader stdin = new BufferedReader(ir);
```

(5) Different Documentation Styles. Sample 1 uses the symbol // while Sample 2 uses /* ... */ for commenting.

It is worth noting that since typically a plagiarist does not know much about programming and/or the specific program, it is very hard for him/her to rename the original variable names as something meaningful. Normally, he/she would just take some abbreviation or compaction of the original variable names.

3.4 Data Collection—Scanner

3.4.1 Design Idea

One thing that makes computer programs different from other text documents is their concise and precise statement flow. Several programs with different style and

organization may do the same thing. However, once a program is set, it is hard to plagiarize it without any trace since the flow of control and operators cannot be easily changed thoroughly. For example, it can be argued that changing a Java program's structure is not all that different from reprogramming.

As mentioned previously, JPD is Java Plagiarism Detector. First, JPD has a scanner that tokenizes meaningful and comparable tokens based on the calling order of statements for each candidate program. The scanner tokenizes each program into a stream of tokens that may be taken as its identity ready to be compared with others.

The key words and reserved words take very important roles in deciding the flow of control of a program. Operators and method calls are also very valuable in shaping a program's structure. JPD recognizes Java key words, operators, and method calls, while it skips the variables and the comments.

Some previous work skipped comments but tokenized each word including variable names [Gitchell and Tran 99]. There is a pitfall about this idea. Unlike a C programmer, a Java program can declare variables anywhere, even among executable code statements right ahead of referring it in the same block. If a plagiarist puts redundant declaration randomly without calling them afterwards in the module or even calling them by adding some meaningless statements, this would significantly change the size and layout of the token stream. Thus the result from this kind of scanner may not be accurate and the detection system may not work well.

JPD skips variable declarations, variable names, and comments. It even does not keep track of the number of variables. For method calls, JPD keeps the operators, ' . ', and the method names regardless of weather they are library methods or programmer-

defined methods. In the case of encountering programmer-defined method calls, JPD expands the method in place. JPD focuses on the flow of key words and operators since they could not be changed significantly unless the plagiarist redesigns or recodes the entire program. To do that, a plagiarist must be experienced in programming and understand what the program does and how it works, and these take a lot of time. We can assume that such a person would rather program by him/herself instead of taking a lot of time copying, studying and modifying somebody else's program.

Constant values also are considered since they have a direct relationship with the algorithm and the design of the code, and they are hard to change. Unless the plagiarist knows the program well, he/she can only make some superficial changes to the original program.

Here is a segment of code from Sample Program 2 given in Section 3.3.2:

```
178)    int Dryear = Integer.parseInt(date.substring(6));
179)    int Drdate = Integer.parseInt(date.substring(3,5));

180)    int Agyears = (Syear - driveryear);
181)    if (Smonth <= Drmonth && Sdate < Drdate)
182)          Agyears - = 1;
183)    return Agyears;
```

The way JPD scans this piece of Java source code is as follows.

= .parseInt (.substring (6)) ; = . parseInt (.substring (3 , 5)) ; = (-) ; if (< = & & <) - = 1 ; return ;

Here is the original code segment from Sample Program 1given in Section 3.2.

```
98)     int driverdate = Integer.parseInt(date.substring(3,5));
99)     int driveryear = Integer.parseInt(date.substring(6));

100)     int ageinyears = (systemyear - driveryear);
101)    if (systemmonth <= drivermonth && systemdate < driverdate)
102)                    ageinyears = ageinyears - 1;
103)    return ageinyears;
```

The token stream that the scanner produces is as follows:

= . parseInt (.substring (3, 5) ; = (.parseInt (.substring (6)) ; = (-) ; if (< = && <) = - 1 ; return ;

Obviously, the two code segments are very similar based on comparing their longest common streams.

There are 34 tokens in the LCS (Longest Common Stream) and the number of tokens in each segment is 39. The similarity between them is calculated in the following way: number of tokens in the LCS / number of tokens in Segment1 / number of tokens in Segment2.

In this case, we have:

34 / 39 / 39

As we have seen, the two code segments have the same size and also the number of tokens in their longest common streams is very close to the average size of the two programs. Similar size and a large number of tokens in longest common streams in a group of pair-wise program comparisons are important signs of plagiarism that will be explained in Section 4.3.

3.4.2 Discussion and Conclusions

JPD can handle most of the techniques that are typically used to disguise changes made to programs when they are copied and plagiarized, especially for the common practices mentioned earlier. A list of these practices and how JPD behave in response to them is given below.

(1) Variables renaming

JPD is immune to variable renaming since it does not take variables into consideration.

(2) Method reorder

Since JPD tokenizes the source code in the execution order, it is able to handle the reordering of the methods.

(3) Interchanging equivalent statements

When tokenizing programs, JPD assigns a common identifier for the 'switch...case' and 'if...else' keywords. "While" loop and "for" loop as well as all other interchangeable keywords also get the same identifiers.

(4) Redundant variables declared or extra methods added

Since JPD skips variable declarations and tokenizes programs in the execution order, any uncalled method would be left untouched.

(5) Combination or decomposition of method calls

This technique is very tricky, especially if programmer-defined method calls are nested or decomposed. If this change is made to a program, assume JPD tokenizes the program in the text layout order and expands programmer-defined method calls in place, the result will be fairly misleading. However, by using a stack structure, JPD is able to deal with nested programmer-defined method calls.

(6) More comments

JPD skips comments, so it is immune to this kind of change.

Each program in a suspected plagiarism case involving two programs may contain one or more different classes from the counterpart program while each class consists of one or more methods. The output from JPD's scanner for each input program is a stream of tokens that is used by the second phase of JPD, TSM (Token Stream Matcher), as input for further inspection.

3.4.3 Implementation Issues

In JPD, there are three passes of scanning programs. They all have been implemented using Unix facilities such as Flex and shell scripts, together with C. Appendices N, O, P, and Q contain the source code of JPD. The first pass of JPD is for scanning all the programmer-defined class names and method names, then storing them into a linked list. The second pass is for tokenizing each method in all classes and outputting the token stream into a file by the name of the respective method. During this process, when encountering a programmer-defined method being called in the middle of another method, JPD just leaves the method name alone and goes on. Then, in the third pass, JPD starts from the token stream file of the main method, and copies each token into a file called "total". When encountering a programmer-defined method call, JPD just expends the token stream of that method in place until exit from the main method. JPD may need to run the third pass multiple times since one method may call another utility method that in turn may call others.

3.5 Token Stream Matching - TSM

The second phase of JPD is to match each pair of token streams from two programs in order to detect the similarity between each pair. By utilizing the dynamic programming technique of locating the longest common stream (LCS), TSM (Token Stream Matcher) has been implemented. Appendix Q is the source code of TSM.

Assume there are two strings Xi and Yj. The entry c[i, j] is defined as the length of an LCS of the sequences Xi and Yj. We have the recursive optimal substructure of the LCS problem as follows [Cormen et al. 90]:

$$c[i, j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ c[i-1, j-1] + 1 & \text{if } i, j > 0 \text{ and } Xi = Yj \\ \text{Max}(c[i, j-1], c[i-1, j]) & \text{if } i, j > 0 \text{ and } Xi \neq Yj \end{cases}$$

The psudo code of the LCS program is as follows.

```
LCS-LENGTH(X, Y)

1 m ← length[X]
2 n ← length[Y]
3 for i ← 1 to m
4       do c[i, 0] ← 0
5 for j ← 0 to n
6       do c[0, j] ← 0
7 for i ← 1 to n
8       do if j ←1 to n
9             do if Xi = Yj
10                  then c[i, j] ← c[i – 1, j – 1] + 1
11                       b[i, j] ← "↖"
12                  else if c[i – 1, j] ≥ c[i, j – 1]
13                       then c[i, j] ← c[i – 1, j]
14                            b[i, j] ← "|"
15                       else c[i, j] ← c[i, j – 1]
16                            b[i, j] ← "←"
17 return c and b
```

Here, b[1..m, 1..n] is used to simplify the construction of an optimal solution. The element b[i, j] points to the table entry corresponding to the optimal sub problem solution chosen when computing c[i, j].

The actual implementation takes token streams from two files that are to be compared. There is an integer variable count added in the algorithm to keep track of the

number of tokens in LCS, which is retrieved later to be used for investigation of similarity.

3.6 Overall Result Collection

To automatically detect plagiarism among a set of programs, a shell script was implemented. First, it calls Scanner to tokenize each Java program into a token stream file, then it calls TSM to take each program as the target and calculate the LCS with every other program, finally it outputs the distribution of the similarities for all pairs. The final result can be used for detecting plagiarism and locating suspicious cases. Appendices C, G, and K are execution results for a group of pair-wise comparisons of Java programs that were caught by JPD as having suspicious pairs. The Java programs of the pairs being caught have been attached in Appendixes D, E, H, I, L and M. By utilizing charts to plot the data from execution results, the similarity distribution can be clearly shown. Appendices B, F, and J illustrate the similarity distributions.

CHAPTER IV

RESULT AND DISSCUSION

4.1 Input Data Sources

The data sources that were used to test JPD were all from real computer science classes. To protect the students' privacy, all names were discarded. A total of 34 Java programs were tested.

4.2 Sample Result from Scanner

After a candidate Java program is scanned, the token stream is output to a file called "total". The following is the token stream of "Sample_Program1" mentioned earlier in Section 3.2.

```
= new ( new ( . ) ) ; = .readLine ( ) ; = " " ; = " " ; = " " ; = " " ; = " " ;
= " " ; = " " ; = 0 ; = 0 ; = 0 ; = 0 ; While_For ( ! = ) = new ( ) ; = (
.nextToken ( ) ) ; If_Else_Switch_Case ( .equalsIgnoreCase ( " " ) ) = (
.nextToken ( ) ) ; = ( .nextToken ( ) ) ; = ( .nextToken ( ) ) ; = .parseInt (
) ; While_For ( .hasMoreTokens ( ) ) = + " " + ( .nextToken ( ) ) ; = new ( ) ;
. .print ( .toUpperCase ( ) + " : " ) ; = ( ) = new ( ) ; = .get ( . ) ; = ( -
) ; = 0 ; If_Else_Switch_Case ( > = 0 & & < = 3 ) = 8 5 ; Else_Default
If_Else_Switch_Case ( > = 4 & & < = 7 ) = 7 5 ; Else_Default
If_Else_Switch_Case ( > = 8 & & < = 11 ) = 55 ; Else_Default
If_Else_Switch_Case ( > = 1 2 & & < = 1 5 ) = 3 5 ; Else_Default = 1 5 ; return
; ; . .print ( " $ " + + " " ) ; . . print ( = new ( ) ; ; = " " ; = " " ; = "
" ; While_For ( = 0 ; < 3 ; + + ) = .nextInt ( ) ; = .abs ( .nextInt ( ) ) % 2
6 ; = + .charAt ( ) ; = + ' - ' ; While_For ( = 0 ; < 3 ; + + ) = .nextInt ( )
; = .abs ( .nextInt ( ) ) % 1 0 ; = + ; return ; + " " + + " " + + " " + + " -
- " ) ; . .println ( = " " ; = ( .nextToken ( ) ) ; While_For ( .hasMoreTokens
( ) ) = + " " + ( .nextToken ( ) ) ; return ; ; = + 1 ; = " " ; Else_Default =
( .nextToken ( ) ) ; While_For ( .hasMoreTokens ( ) ) = + " " + ( .nextToken (
) ) ; = new ( ) ; = .substring ( 0 , 3 ) ; . .print ( .toUpperCase ( ) + " : $
5 " ) ; . .print ( = new ( ) ; ; = " " ; While_For ( = 0 ; < 9 ; + + ) =
.nextInt ( ) ; = .abs ( .nextInt ( ) ) % 1 0 ; = + ; return ; + " " + + " " +
= new ( ) ; = .get ( . ) ; = + 1 ; = .get ( . ) ; = .get ( . ) ; = .parseInt (
```

```
.substring ( 0 , 2 ) ) ; = .parseInt ( .substring ( 3 , 5 ) ) ; = .parseInt (
.substring ( 6 ) ) ; = ( - ) ; If_Else_Switch_Case ( < = & & < ) = - 1 ; return
;  ; . .println ( = " " ; = ( .nextToken ( ) ) ; While_For ( .hasMoreTokens ( )
) = + " " + ( .nextToken ( ) ) ; return ;  ; = + 1 ; = " " ; = .readLine ( ) ;
. .println ( ) ; If_Else_Switch_Case ( = = 0 & & = = 0 ) . .println ( " ! ! " )
; Else_Default . .println ( " : " + ) ; . .println ( " : " + ) ;
```

The total number of tokens is 674.

The token stream of "Sample_Program2" is listed below.

```
= new ( . ) ; = new ( ) ; = 0 ; = 0 ; = 0 ; = 0 ; = " " ; = " " ; = " " ; = " "
; = " " ; = " " ; = " " ; = .readLine ( ) ; While_For ( ! = ) = new ( ) ; = (
.nextToken ( ) ) ; If_Else_Switch_Case ( .equalsIgnoreCase ( " " ) ) = (
.nextToken ( ) ) ; = ( .nextToken ( ) ) ; = ( .nextToken ( ) ) ; = .parseInt (
) ; While_For ( .hasMoreTokens ( ) ) = + " " + ( .nextToken ( ) ) ; = new ( ) ;
. .print ( .toUpperCase ( ) + " : " ) ; = ( ) = 0 ; = .get ( . ) ; = ( - ) ; =
new ( ) ; If_Else_Switch_Case ( > = 0 & & < = 3 ) = 8 5 ; Else_Default
If_Else_Switch_Case ( > = 4 & & < = 7 ) = 7 5 ; Else_Default
If_Else_Switch_Case ( > = 8 & & < = 11 ) = 55 ; Else_Default
If_Else_Switch_Case ( > = 1 2 & & < = 1 5 ) = 3 5 ; Else_Default = 1 5 ; return
;  ; . .print ( " $ " + + " " ) ; . .print ( = " " ; ; ; ; = " " ; = new ( ) ;
= " " ; While_For ( = 0 ; < 3 ; + + ) = .nextInt ( ) ; = .abs ( .nextInt ( ) )
% 2 6 ; = + .charAt ( ) ; = + ' - ' ; While_For ( = 0 ; < 3 ; + + ) = .nextInt
( ) ; = .abs ( .nextInt ( ) ) % 1 0 ; = + ; return ; + " " + + " " + + " " + +
" - - " ) ; . .println ( = " " ; = ( .nextToken ( ) ) ; While_For (
.hasMoreTokens ( ) ) = + " " + ( .nextToken ( ) ) ; return ;  ; = + 1 ; = " " ;
Else_Default = ( .nextToken ( ) ) ; While_For ( .hasMoreTokens ( ) ) = + " " +
( .nextToken ( ) ) ; = new ( ) ; = .substring ( 0 , 3 ) ; . .print (
.toUpperCase ( ) + " : $ 5 " ) ; . .print ( ; = " " ; = new ( ) ; While_For ( =
0 ; < 9 ; + + ) = .nextInt ( ) ; = .abs ( .nextInt ( ) ) % 1 0 ; = + ; return ;
+ " " + + " " + = .get ( . ) ; = new ( ) ; = .get ( . ) ; = + 1 ; = .get ( . )
; = .parseInt ( .substring ( 0 , 2 ) ) ; = .parseInt ( .substring ( 3 , 5 ) ) ;
= .parseInt ( .substring ( 6 ) ) ; = ( - ) ; If_Else_Switch_Case ( < = & & < )
= - 1 ; return ;  ; . .println ( = " " ; = ( .nextToken ( ) ) ; While_For (
.hasMoreTokens ( ) ) = + " " + ( .nextToken ( ) ) ; return ;  ; = + 1 ; = " " ;
= .readLine ( ) ; . .println ( ) ; If_Else_Switch_Case ( = = 0 & & = = 0 ) .
.println ( " ! ! " ) ; Else_Default . .println ( " : " + ) ; . .println ( " : "
+ ) ;
```

The number of tokens in this stream is 678.

The LCS of "Sample_Program1" and "Sample_ Program2" is given below.

```
= new ( . ) ; = ( ) ; = ; = ; = ; = " " ; = " " ; = " " ; = " " ; = ; = ; = ; =
; While_For ( ! = ) = new ( ) ; = ( .nextToken ( ) ) ; If_Else_Switch_Case (
.equalsIgnoreCase ( " " ) ) = ( .nextToken ( ) ) ; = ( .nextToken ( ) ) ; = (
.nextToken ( ) ) ; = .parseInt ( ) ; While_For ( .hasMoreTokens ( ) ) = + " " +
( .nextToken ( ) ) ; = new ( ) ; . .print ( .toUpperCase ( ) + " : " ) ; = ( )
= ; = .get ( . ) ; = ( - ) ; = ; If_Else_Switch_Case ( > = 0 & & < = 3 ) = 8 5
; Else_Default If_Else_Switch_Case ( > = 4 & & < = 7 ) = 7 5 ; Else_Default
If_Else_Switch_Case ( > = 8 & & < = 1 1 ) = 5 5 ; Else_Default
If_Else_Switch_Case ( > = 1 2 & & < = 1 5 ) = 3 5 ; Else_Default = 1 5 ; return
; ; . .print ( " $ " + + " " ) ; . .print ( = ; ; = " " ; = ; = " " ; While_For
( = 0 ; < 3 ; + + ) = .nextInt ( ) ; = .abs ( .nextInt ( ) ) % 2 6 ; = +
.charAt ( ) ; = + ' - ' ; While_For ( = 0 ; < 3 ; + + ) = .nextInt ( ) ; = .abs
( .nextInt ( ) ) % 1 0 ; = + ; return ; + " " + + " " + + " " + + " - - " ) ; .
.println ( = " " ; = ( .nextToken ( ) ) ; While_For ( .hasMoreTokens ( ) ) = +
```

```
" " + ( .nextToken ( ) ) ; return ; ; = + 1 ; = " " ; Else_Default = (
.nextToken ( ) ) ; While_For ( .hasMoreTokens ( ) ) = + " " + ( .nextToken ( )
) ; = new ( ) ; = .substring ( 0 , 3 ) ; . .print ( .toUpperCase ( ) + " : $ 5
" ) ; . .print ( ; = " " ; While_For ( = 0 ; < 9 ; + + ) = .nextInt ( ) ; =
.abs ( .nextInt ( ) ) % 1 0 ; = + ; return ; + " " + + " " + = new ( ) ; = .get
( . ) ; = + 1 ; = .get ( . ) ; = .parseInt ( .substring ( 0 , 2 ) ) ; =
.parseInt ( .substring ( 3 , 5 ) ) ; = .parseInt ( .substring ( 6 ) ) ; = ( - )
; If_Else_Switch_Case ( < = & & < ) = - 1 ; return ; ; . .println ( = " " ; = (
.nextToken ( ) ) ; While_For ( .hasMoreTokens ( ) ) = + " " + ( .nextToken ( )
) ; return ; ; = + 1 ; = " " ; = .readLine ( ) ; . .println ( ) ;
If_Else_Switch_Case ( = = 0 & & = = 0 ) . .println ( " ! ! " ) ; Else_Default .
.println ( " : " + ) ; . .println ( " : " + ) ;
```

The length of LCS (Longest Common Stream) of these two programs is 640. Thus the similarity coefficient can be showed in the following way.

Number of tokens in LCS / number of tokens in Sample_Program1 / number of tokens in Sample_Program2 = 640 / 674 / 678

As can we have in this case, not only are the sizes of both programs very close, but also the number of tokens in LCS is close to the size of either program.

4.3 Pair-Wise Comparison Result Discussion

Output from ShellScript1 (APPENDIX R) and ShellScript2 (APPENDIX S) is an overall similarity indicator for each pair-wise comparison. This indicator consists of seven components: number of tokens in LCS, directory name of the target program, directory name of the candidate program being compared, number of tokens in the target token stream file, the file name of the target token stream file, the number of tokens in the candidate token stream file, and the name of the candidate token stream file being compared.

Each tuple looks like the following:

NT1 SD1 SD2 NT2 ../target NT3 total

NT1 is the number of tokens in the longest common token stream for this pair

SD1 is the target student directory name that can be used to locate the student

SD2 is the student directory name where the program that is being compared to the target program resides

NT2 is the total number of tokens in the target token stream file

NT3 is the total number of tokens in the candidate token stream file which is being compared to the target.

The components "../target" and "total" are the name of token stream files being compared.

Appendices C, G, and K are samples of execution result. Appendices B, F, and J contain plots for these results. There is a small circle identifying the highest similarity in each chart. The two programs involved have close sizes and have the highest number of tokens in terms of their LCS among all pair-wise comparisons in the group. Six pairs of programs were identifies by JPD as being involved in plagiarism. Three of the six have being attached as appendices in this thesis. Appendices D and E, H and I, and L and M are the similar pairs of Java programs caught by JPD. By carefully inspecting each pair, evidence of copying was noticed.

There are no definitive and conclusive criteria that can be used to judge whether or not there is plagiarism for each group of pair-wise comparisons. The most judicious way is to locate pairs of programs with similar sizes and relatively large number of tokens in their respective LCS, and then to try to manually examine them.

CHAPTER V

CONCLUSION AND FUTURE WORK

JPD has been shown to work well by testing it on 34 Java programs written by students in computer science classes. There were a total six pair of programs that were caught. They were easily located by examining the charts generated from the data taken from the execution results of JPD.

The following are limitations of JPD.

1. All programs being tested must be compiled with no syntax errors.
2. The number of recognized tokens cannot exceed 1000. However, one can make a small change in the dimensions of the token array in the source code of TSM to overcome the limitation.
3. Depend on the situation, a user may have to write his/her own shell script to test Java programs automatically.

Some areas of future work that can be done to improve the detection system are listed below.

1. JPD can handle multiple classes as long as no two methods share the same name. If method overriding happens in the Java programs being investigated, JPD cannot distinguish which method it calls.
2. A graphical user interface for JPD can be designed and implemented.

REFERENCES

[Cormen, et al. 90] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest, *Introduction to Algorithm*, MIT Press, Cambridge, MA, 1989.

[Fu 86] Wen-yu Fu, "Design and Implementation of a Software Tool that Detects Plagiarism in C Programs," Master of Science Thesis, Department of Computer Science, University of Houston - University Park, Houston, TX, May 1986.

[Gitchell and Tran 99] David Gitchell and Nicholas Tran, "Sim: A Utility for Detecting Similarity in Computer Programs", *Proceedings of the Thirtieth SIGCSE Technical Symposium on Computer Science Education*, pp. 266-270, New Orleans, LA, March 1999.

[Grier 80] Samuel L. Grier, Jr., "A Tool for Detecting Plagiarism in Pascal Programs," Master of Science Thesis, Department of Computer Science, University of Colrado, Boulder, CO, 1980.

[Halstead 77] M. H. Halstead, *Elements of Software Science*, Elsevier North Holland, New York, NY, 1977.

[Harris 94] James K. Harris, "Plagiarism in Computer Science Courses", *Proceedings of the Conference on Ethics in the Computer Age*, pp. 133 – 135, Galtinburg, TN, November 1994.

[Huang, et al. 90] X. Huang, R. C. Hardison, and W. Miller, "A Space-Efficient Algorithm for Local Similarities", *Computer Applications in the Biosciences*, Vol. 1, No. 2, pp. 373-381, June 1990.

[Litwak, 99] Kenneth Litwak, *Pure Java 2 a Code-Intensive Premium Reference*, Sams Pulishing Companies, Inc., Indianapolis, IN, 1999.

[Malpohl 00] Guido Malpohl, "Jplag Detection Software Plagiarism WEB Page", http://wwwipd.ira.uka.de:2222, last modified July 2000, access date October 2000.

[Martin 97] John C. Martin, *Introduction to Languages and the Theory of Computation*, McGraw-Hill Companies, Inc. New York, NY, 1997.

[MOSS 00] MOSS, "A System for Detecting Software Plagiarism",

http://www.cs.berkeley.edu/~aiken/moss.html, last modified April 2000, access date October 2000.

[Myers and Miller 88] E. W. Myers and W. Miller, "Optimal Alignments in Linear Space", *Computer Applications in the Biosciences*, Vol. 3, No. 2, pp. 11-17, April 1988.

[Ottenstein 76] K. J.Ottenstein, "An Algorithmic Approach to the Detection and Prevention of Plagiarism," *SIGCSE Bulletin*, Vol. 8, No. 4, December 1976.

[Palmer 00] Grant Palmer, *Java Programmer's Reference*, Wrox Press Ltd., Acock's Green, Birmingham, UK, 2000.

APPENDICES

APPENDIX A

GLOSSARY

Flex	Fast lexical analyzer generator, a tool for generating programs that performs pattern-matching on text files.
JPD	Java Plagiarism Detector, the plagiarism detection tool implemented as part of this thesis.
LCS	Longest Common Stream, an algorithm to look up the longest common stream in two streams, equivalent to Longest Common Subsequence [Cormen et al. 90, pp 314 – 315].
OOP	Objected Oriented Programming, writing programs in one of a class of programming languages and techniques based on the concept of an "object", which is a data structure (abstract data type) encapsulated with a set of routines called "methods" that operate on the data.
Parser	An algorithm or program to determine the syntactic structure of a sentence or a string of symbols in some language. A parser normally takes as input a sequence of tokens that are output by a lexical analyzer or scanner. A parser produces abstract syntax trees as output.
Scanner	Also called lexical analyzer, a program that recognizes valid tokens and symbols in a program for a specific language.
TSM	Token Stream Matcher, one of the components of JPD (Appendix Q).

APPENDIX B

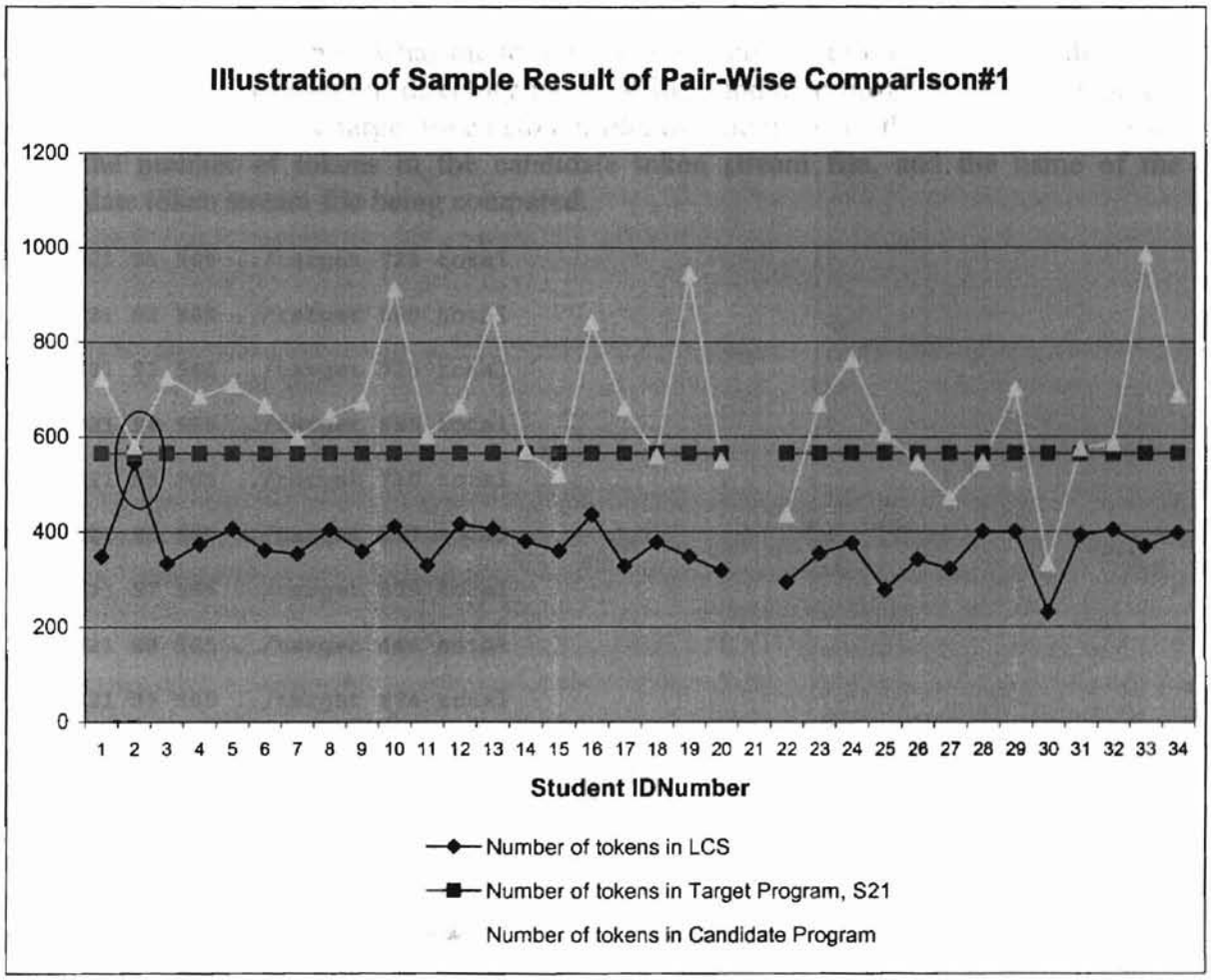
Illustration of Sample Result of Pair-Wise Comparison#1
1200
1000
800
600
400
200
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34
Student IDNumber
Number of tokens in LCS
Number of tokens in Target Program, S21
Number of tokens in Candidate Program

APPENDIX C

Sample Execution Result of Pair-Wise Comparison #1

Each row of the result below has the following order: number of tokens in LCS, directory name of the target program, directory name of the candidate program being compared, number of tokens in the target token stream file, the file name of the target token stream file, the number of tokens in the candidate token stream file, and the name of the candidate token stream file being compared.

```
347 S21 S1 565 ../target 723 total

545 S21 S2 565 ../target 580 total

333 S21 S3 565 ../target 725 total

373 S21 S4 565 ../target 685 total

405 S21 S5 565 ../target 710 total

361 S21 S6 565 ../target 667 total

353 S21 S7 565 ../target 599 total

404 S21 S8 565 ../target 646 total

358 S21 S9 565 ../target 674 total

411 S21 S10 565 ../target 912 total

329 S21 S11 565 ../target 603 total

416 S21 S12 565 ../target 662 total

405 S21 S13 565 ../target 861 total

379 S21 S14 565 ../target 568 total

359 S21 S15 565 ../target 520 total

436 S21 S16 565 ../target 841 total

327 S21 S17 565 ../target 661 total

377 S21 S18 565 ../target 560 total

347 S21 S19 565 ../target 946 total

318 S21 S20 565 ../target 550 total

293 S21 S22 565 ../target 435 total

353 S21 S23 565 ../target 669 total
```

```
375 S21 S24 565 ../target 763 total
277 S21 S25 565 ../target 607 total
341 S21 S26 565 ../target 547 total
321 S21 S27 565 ../target 472 total
400 S21 S28 565 ../target 547 total
400 S21 S29 565 ../target 702 total
231 S21 S30 565 ../target 332 total
392 S21 S31 565 ../target 576 total
404 S21 S32 565 ../target 586 total
368 S21 S33 565 ../target 986 total
397 S21 S34 565 ../target 689 total
```

APPENDIX D

Program from student S21 that was identified by JPD as being involved in plagiarism with the program of student S2 which appears in Appendix E.

```
//=============================================================
//
//     Program: Target Program #1
//
//     Author: S21
//
//=============================================================
   import java.io.* ;
   import java.util.* ;
   import java.text.NumberFormat;
   import java.text.DecimalFormat;

   public class pgm04 {

            public static String tagNumber() {
       Random r = new Random();
       int num1; double num2, num3; char alpha;
       String TAGNO = " ";
//     get alphabet
        for(int i = 0; i<3;i++)

       Random num2 = r.nextdouble();
       alpha=(char) (num2 * 27) + 65;
       TAGNO +=  alpha;

//     get numerals
       TAGNO += "-";
       for(int i = 0; i<3;i++)
  {
       num3 = r.nextdouble();
       num1 = (int)(num3 * 10);
       TAGNO += num1;
  }
       return TAGNO;
}
//     generate and return a license number in th format ("NNNNNNNNN").
   Public static String licenseNumber()
  {
       Random lic = new Random();
       double licnum2 ;int licnum1 ;
       String LICENSE1 = " ";
       for(int i = 0; i<10;i++)
{
       licnum1 = (int)((licnum2 = lic.nextDouble())*10);
       LICENSE1 += licnum1;
}
       return LICENSE1;
 }

//  this returns the cost of the tag based on the age of the vehicle.
  Public static double tagCost(int year)
```

```
{
      GregorianCalendar cal = new GregorianCalendar();
      int yr = cal.get(Calendar.YEAR);
      int COST = 0;
      int yrdiff = yr - year;
      if(yrdiff >= 0 && yrdiff <= 3)
        COST = 85;
      else if (yrdiff >= 4 && yrdiff <=7)
        COST = 75;
      else if (yrdiff >=8 && yrdiff <=11)
        COST = 55;
      else if (yrdiff >=12 && yrdiff <=15)
        COST = 35;
      else if (yrdiff >= 16)
        COST = 15;
      else;
      return COST;
}

// this code eliminates all trailing white spaces.
  Public static String minimizeWhiteSpace(StringTokenizer a)
  {
      String str1 = " ";
      while(a.hasMoreTokens())
  {
      str1 += a.nextToken() + " ";
}
      return str1;
}

// this code calculates the age of a licensee
  public static int calAge(String date)
{
      int AGE = 0;
      GregorianCalendar cal = new GregorianCalendar();
      int day1 = cal.get(Calendar.DATE);
      int mon = cal.get(Calendar.MONTH) + 1;
      int yr = cal.get(Calendar.YEAR);
      int mm = Integer.parseInt(date.substring(0,2)) ;
      int dd = Integer.parseInt(date.substring(3,5)) ;
      int yy = Integer.parseInt(date.substring(6));
      if(mon > mm)
        AGE = yr - yy;
      else if(month < mm)
        AGE = (yr - yy) - 1;
      else if (mon == mm)
   {
      if(day1 < dd)
        AGE = yr - yy;
      else
        AGE = (yr - yy) - 1;
}
      else;
      return AGE;
}

// this code will process the input data
   public static void main(String[] args) throws IOException
{
      String linea;
      int lic1 = 0, tag1 = 0;
    BufferedReader stdin = new BufferedReader
        (new InputStreamReader(System.in));
```

```
        linea = Stdin.readLine();
        while(linea != null && linea.length() != 0)
 {
        StringTokenizer t = new StringTokenizer(input);
        String t1 = "";
        t1 = t.nextToken();
        if(t1.equalsIgnoreCase("tag"))
  {
        String manu = t.nextToken();
        String tno = tagNumber();
        String carmk t.nextToken();
        int year = Integer.parseInt(t.nextToken());
        String namea = "";
        namea = minimizeWhiteSpace(t);
        double cost = tagCost(year);
         System.out.println("TAG : " + "$" + (int)cost + " " +tno + " "
        + " " + manu + " " tag1++));
// this counts the increment of the tags issued.
 }
        else
        if(t1.equalsIgnoreCase("license"))
  {
        String year;
        year = t.nextToken();
        String nameb = "";
        nameb = minimizeWhiteSpace(t);
        int age1 = calAge(year);
        String licno;
        licno = licenseNumber();
         System.out.println("LIC : $5" + " " +licno + " " + year +" " + age1
        + " -" + lic1++);
// counts the increment of licenses issued
 }
        else;
   // end of the while loop.
        System.out.println(" ");
        System.out.println("Number of tags issued : " + tag1);
        System.out.println("Number of licenses issued : " + lic1);

   } // method main
 }
 }  // end of class pgm04
```

APPENDIX E

Program from student S2 that was identified by JPD as being involved in plagiarism with the previous program of student S21 in Appendix D.

```
//=====================================================================
//
// Program: Candidate Program with high similarity with Target #1
//
// Author: S2
//=====================================================================

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

public class pgm04
{
public static String tagNumber()
{
Random r = new Random();
int num1;
float num2,num3;
char xyz;
String TAGNUM = " ";

for(int i = 0; i < 3 ;i++)
{
xyz = (char) (((num2=r.nextFloat())*27)+ 65);
TAGNUM +=xyz ;
}
TAGNUM += »-«  ;
for(int i = 0; i < 3;i++)
{
num1 =(int) ((num3=r.nextFloat())*10);
TAGNUM+=num1;
}
return TAGNUM;
}

public static String licenseNumber()
{
Random lc = new Random();
float lnum2;int lnum1;
String L1= " ";
for(int i = 0; i<10; i++)
{
lnum1 =(int) ((lnum2 =lc.nextFloat())*10);
L1 +=lnum1;
}
return L1;
}

public static float tagCost(int year)
{
```

```
GregorianCalendar cal = new GregorianCalendar();
int Year = cal.get(Calendar.YEAR);
int cost =0;
int yrdiff = Year - year;
if (yrdiff >= 0&& yrdiff <= 3)
cost =85;
else
if(yrdiff >= 4 && yrdiff<= 11)
cost = 75;
else
if (yrdiff >= 8 && yrdiff <= 15)
cost = 55;
if (yrdiff >= 12 && yrdiff <= 15) cost = 35;
else if(yrdiff >=16)
cost = 15;
else;
return cost;
}
public static int calcAge(String date)
{
int age = 0;

GregorianCalendar cal = new GregorianCalendar();
int day1 = cal.get(Calendar.DATE);
int month = cal.get(Calendar.MONTH) + 1;
int Year = cal.get(Calendar.YEAR);
int mm = Integer.parseInt(date.substring(0,2)) ;
int dd = Integer.parseInt(date.substring(3,5)) ;
int yy = Integer.parseInt(date.substring(6));
if (month > mm)
age = Year - yy;
if (month <  mm )
age = (Year - yy) - 1;
if(month == mm)
{
if(day1 < dd)
age = Year - yy;
else
age=(Year - yy) - 1;
}
else;
return age;
}
public static String minimizeWhiteSpace(StringTokenizer m)
{
String Str1 = " ";
while(m.hasMoreTokens())
{

Str1 += m.nextToken() + " ";
}
return Str1;
}


public static void main (String[] args) throws IOException
{
String linem;
int lc1 = 0, tag1 = 0;
InputStreamReader isr=new InputStreamReader (System.in);
BufferedReader Stdin = new BufferedReader (isr);

linem = Stdin.readLine();
```

```
while(linem != null && linem.length() != 0)
{
StringTokenizer tz = new StringTokenizer(linem);
String t1 = "";
t1= tz.nextToken();
if (t1.equalsIgnoreCase("tag"))
{
String mfc = tz.nextToken();
String tno = tagNumber();
String vmake = tz.nextToken();
int year = Integer.parseInt(tz.nextToken());
String namem = "";
namem = minimizeWhiteSpace(tz);
float cost = tagCost(year);

System.out.println("TAG : " + "$" + (int)cost + " " + tno + " " + mfc + " " +
namem);
tag1++;
}
else
if(t1.equalsIgnoreCase("license"))
{
String year;
year = tz.nextToken();
String namen = "";
namen = minimizeWhiteSpace(tz);
int age1 =calcAge(year);
String lno;

lno = licenseNumber();
System.out.println("LIC : $5" + " " + lno + " " + year +" " + age1 +"  --" +
namen);
lc1++;
}
else
System.out.println("wrong input");
linem = Stdin.readLine();
}

System.out.println(" ");
System.out.println("Number of tags issued : " + tag1);
System.out.println("Number of licenses issued : " + lc1);

}
}
```

APPENDIX F

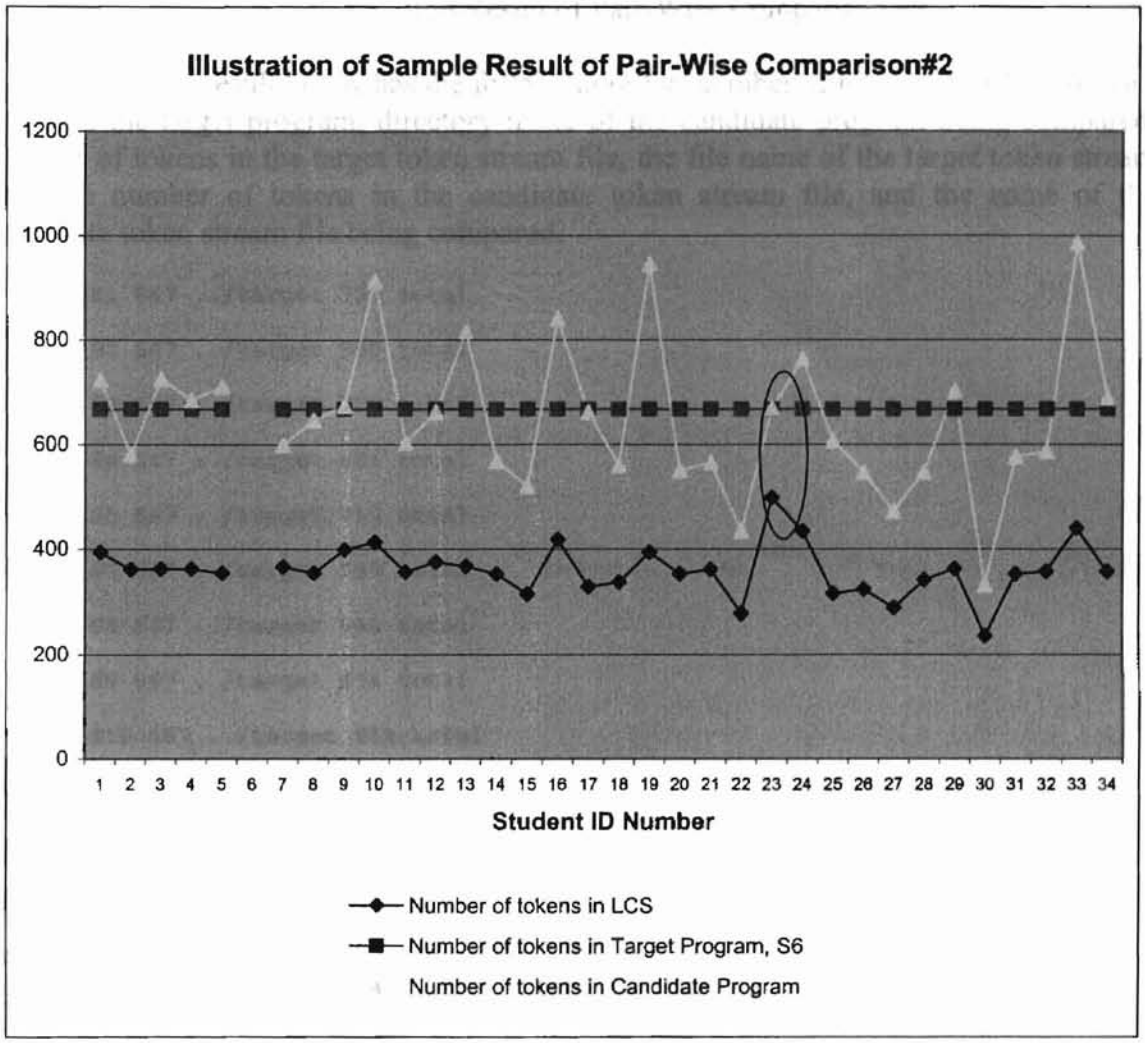
Illustration of Sample Result of Pair-Wise Comparison#2
1200
1000
800
600
400
200
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34
Student ID Number
Number of tokens in LCS
Number of tokens in Target Program, S6
Number of tokens in Candidate Program

APPENDIX G

Sample Execution Result of Pair-Wise Comparison #2

Each row of the result below has the following order: number of tokens in LCS, directory name of the target program, directory name of the candidate program being compared, number of tokens in the target token stream file, the file name of the target token stream file, the number of tokens in the candidate token stream file, and the name of the candidate token stream file being compared.

```
394 S6 S1 667 ../target 723 total

361 S6 S2 667 ../target 580 total

362 S6 S3 667 ../target 725 total

362 S6 S4 667 ../target 685 total

354 S6 S5 667 ../target 710 total

366 S6 S7 667 ../target 599 total

354 S6 S8 667 ../target 646 total

399 S6 S9 667 ../target 674 total

413 S6 S10 667 ../target 912 total

356 S6 S11 667 ../target 603 total

375 S6 S12 667 ../target 662 total

367 S6 S13 667 ../target 861 total

352 S6 S14 667 ../target 568 total

314 S6 S15 667 ../target 520 total

418 S6 S16 667 ../target 841 total

328 S6 S17 667 ../target 661 total

337 S6 S18 667 ../target 560 total

394 S6 S19 667 ../target 946 total

353 S6 S20 667 ../target 550 total

361 S6 S21 667 ../target 565 total

277 S6 S22 667 ../target 435 total
```

```
498 S6 S23 667 ../target 669 total

434 S6 S24 667 ../target 763 total

315 S6 S25 667 ../target 607 total

323 S6 S26 667 ../target 547 total

288 S6 S27 667 ../target 472 total

341 S6 S28 667 ../target 547 total

362 S6 S29 667 ../target 702 total

235 S6 S30 667 ../target 332 total

352 S6 S31 667 ../target 576 total

357 S6 S32 667 ../target 586 total

440 S6 S33 667 ../target 986 total

357 S6 S34 667 ../target 689 total
```

APPENDIX H

Program from student S6 that was identified by JPD as being involved in plagiarism with the program of student S23 which appears in Appendix I.

```
// ---------------------------------------------------------------------
//
// Program: Target Program #2
//
// Author: S6
//
// ---------------------------------------------------------------------

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

public class pgm04
{
   public static String licenseNumber ()
   {
   // Creates a method that calculates the 9 digit number of the license.
       Random r = new Random ();
       String lic = "";
       int count = 0;

   while (count < 9) {

       float f = r.nextFloat ();
       lic += (int)(f * 10);
       count ++;
       }
       return lic;
  }
   public static int tagCost (int year1)
   {
   // Creates a method that calculates the amount paid for the tag
   // depending on the age of the vehicle.
       int tcost = 0;
   GregorianCalendar cal = new GregorianCalendar ();
       int yr2 = cal.get (Calendar.YEAR);

   {if ((yr2 - 3) <= year1)
       tcost = 85;
   else if ((yr2 - 4) <= year1)
       tcost = 75;
   else if ((yr2 - 8) <= year1)
       tcost = 55;
   else if ((yr2 - 12) <= year1)
       tcost = 35;
   else if ((year1 + 16) <= yr2)
       tcost = 15;
       }
       return tcost;
```

```
}
public static String tagNumber ()
{
// Creates a method that calculates the first three letters and the
// last three numbers of the tag.
    Random r = new Random ();
    String tagA = "", tagB = "", tag = "";
    int count = 0;

while (count < 3) {

    float f = r.nextFloat ();
    char t = (char)((int)(f * 26) + 65);
    tagA += t;
    count ++;
    }
count = 0;
while (count < 3) {
    float f = r.nextFloat ();
    tagB += (int)(f * 10);
    count ++;
    }
tag = tagA + "-" + tagB;
    return tag;
}
public static String minimizeWhiteSpace (StringTokenizer t)
{
// Creates a method that minimizes the white space between each
// individual thing in each line of output.
    String sentence = "";

while (t.hasMoreTokens ()) {

    sentence += t.nextToken ();
    sentence = sentence + " ";
    }
    return sentence;
}
public static int calcAge (String year)
{
// Creates a method that calculates the birthdate and age of the
// driver.

GregorianCalendar cal = new GregorianCalendar ();

    int curmonth = cal.get (Calendar.MONTH), curdate = cal.get
    (Calendar.DATE), curyear = cal.get(Calendar.YEAR);

    String month = year.substring (0,2);
    String day = year.substring (3,5);
    String year3 = year.substring (6);

    double day2 = Double.parseDouble (day);
    double month2 = Double.parseDouble (month);
    double year5 = Double.parseDouble (year3);
    int day1 = (int)(day2);
    int month1 = (int)(month2);
    int year4 = (int)(year5);
    int age = curyear - year4;

if (month1 < curmonth) {
    age = curyear - year4;
if (month1 > curmonth)
```

```
    age = age - 1;
 if (month1 == curmonth && day1 < curdate)
    age = age - 1;
    }
 else;
     age = age;

 return age;
}
 public static void main (String [] args) throws IOException
 {
 // Creates a method that calls all actions performed in the other
 // methods.

    String manufacturer, style, year, year4 = "";
    String cost1 = "", sentence = "";
    int tissued = 0, lissued = 0, year1 = 0;
BufferedReader stdin = new BufferedReader (new InputStreamReader(System.in));
    sentence = stdin.readLine ();

 while (sentence != null) {
    StringTokenizer t = new StringTokenizer (sentence);

 if (t.countTokens () != 0) {
    String name = t.nextToken ();
 if (name.equalsIgnoreCase ("tag")) {
    manufacturer = t.nextToken ();
    style = t.nextToken ();
    year = t.nextToken ();
    double yr2 = Double.parseDouble (year);
    int yr1 = (int)(yr2);
    System.out.print ("TAG: ");

    int tcost = tagCost (year1);
    System.out.print ("$" + tcost);

    String tag = tagNumber ();
    System.out.print (" " + tag + " ");
    System.out.print (manufacturer + " " + style + " " + yr1 + " ");
    System.out.print ("--" + " " + t.nextToken () + " ");

    String middle = minimizeWhiteSpace (t);
    System.out.println (middle + " ");
    }
 if (name.equalsIgnoreCase ("license")) {
    year4 = t.nextToken ();

    int age = calcAge (year4);
    String lic = licenseNumber ();
    System.out.print ("LIC: $5 "+ lic + " " + year4 + " " + age +" ");
    System.out.print ("--" + " " + t.nextToken () + " ");

    String last = minimizeWhiteSpace (t);
    System.out.println (last);
    }
 if (name.equalsIgnoreCase ("tag")) {
    tissued++;
    }
 if (name.equalsIgnoreCase ("license")) {
    lissued++;
    }
 }
    sentence = stdin.readLine ();
```

```
    }
 if (tissued != 0) {
    System.out.println ("\nNumber of tags issued: " + tissued);
    }
 if (lissued != 0) {
    System.out.println ("Number of licenses issued: " + lissued);
    }
  }
}
```

APPENDIX I

Program from student S23 that was identified by JPD as being involved in plagiarism with the previous program of student S6 in Appendix H.

```
//*****************************************************************
//
// Program: Candidate Program with high similarity with Target #2
//
// Author : S23
//*****************************************************************

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

public class pgm04
{
	static Random r = new Random();  // allows for a new number to be put
into function

	public static int random_number(int size, int adjust){
		float f = r.nextFloat();
		return ((int)((f*size)+ adjust));
	}

	public static String tagNumber(){
		int chnumber;

		String plate="";
		for (int counter = 0; counter < 3; counter++){
			chnumber=(random_number(('Z' - 'A' +1), 'A'));
			plate = plate + (char)(chnumber);
		}
		plate = plate + "-";
		for (int counter = 0; counter < 3; counter++)
			plate = plate + random_number(10, 0);
		return plate;
	}

	public static String licenseNumber(){
		String license = "";
		for (int counter = 0; counter < 9; counter++)
			license = license + random_number(10, 0);
		return license;
	}

	public static float tagCost(int year){
		GregorianCalendar cal = new GregorianCalendar();

		int current_year = (cal.get(Calendar.YEAR));

		if (((current_year - year) >= 0) && ((current_year - year) <= 3))
			return(85);
```

```
            else if (((current_year - year) >= 4) && ((current_year - year) <=
7))
                  return(75);
            else if (((current_year - year) >= 8) && ((current_year - year) <=
11))
                  return(55);
            else if (((current_year - year) >= 12) && ((current_year - year)
<= 15))
                  return(35);
            else
                  return(15);
      }

      public static String minimizeWhitespace(StringTokenizer t){
            String token = "";
            while (t.hasMoreTokens())
           {
            token += t.nextToken();
            if (t.hasMoreTokens())
                        token += " ";
           }
          return token;
      }

      public static int calcAge(String date){
            GregorianCalendar cal = new GregorianCalendar();

            int current_year = (cal.get(Calendar.YEAR));
            int current_month = (cal.get(Calendar.MONTH));
            int current_date = (cal.get(Calendar.DATE));
            double ageYear = Double.parseDouble(date.substring(6));
            double ageMonth = Double.parseDouble(date.substring(0,2));
            double ageDate = Double.parseDouble(date.substring(3,5));
            int year = (int)ageYear;
            int month = (int)ageMonth;
            int dateInt = (int)ageDate;
            int age = current_year - year;

            if (current_month < month)
                  age--;
            else if ((current_month == month) && (current_date < dateInt))
                  age--;

            return age;
      }

      public static void main (String[] args) throws IOException
      {

            BufferedReader stdin = new BufferedReader (new InputStreamReader
(System.in));

           String inputLine, make, model, tag, lic;
            int year=0, tags=0, lics=0;

            inputLine = stdin.readLine();

            while (inputLine != null){
                  StringTokenizer token = new StringTokenizer(inputLine);
                  if (token.countTokens() !=  0){
                        lic = token.nextToken();
                        if (lic.equalsIgnoreCase("tag")){
                              model = token.nextToken ();
```

```
                                make = token.nextToken ();
                                year =
(int)Double.parseDouble(token.nextToken());
                               System.out.print ("TAG: ");
                               int cost = (int)tagCost(year);
                                  System.out.print ("$" + cost);
                                   tag = tagNumber();
                                System.out.print (" " + tag + " ");
                            System.out.print (model + " ");
                                System.out.print (make + " ");
                                 System.out.print (year + " ");
                             System.out.print ("-- " + token.nextToken () + " ");
                                 String name = minimizeWhitespace(token);
                                System.out.println (name + " ");
                              }
                              else
                            if (lic.equalsIgnoreCase ("license")){
                                String DOB = (token.nextToken());
                                int age = calcAge(DOB);
                              String licenseNum = licenseNumber();
                                 System.out.print ("LIC: $5 " + licenseNum + " " +
DOB + " " + age + " ");
                                System.out.print ("-- " + token.nextToken() + "
");
                                String name = minimizeWhitespace(token);
                            System.out.println (name);
                             }

                      if (lic.equalsIgnoreCase ("tag"))
                        tags++;
                    if (lic.equalsIgnoreCase ("license"))
                        lics++;

                    inputLine = stdin.readLine();
                      }
               }

               if (tags != 0)
                     System.out.println ("\nNumber of tags issued: " + tags);
               else
                      System.out.println ("\nNumber of tags issued cannot be
calculated.");

             if (lics != 0)
                     System.out.println ("Number of licenses issued: " + lics);
               else
                      System.out.println ("Number of licenses issued cannot be
calculated.");
       }
}
```

APPENDIX J

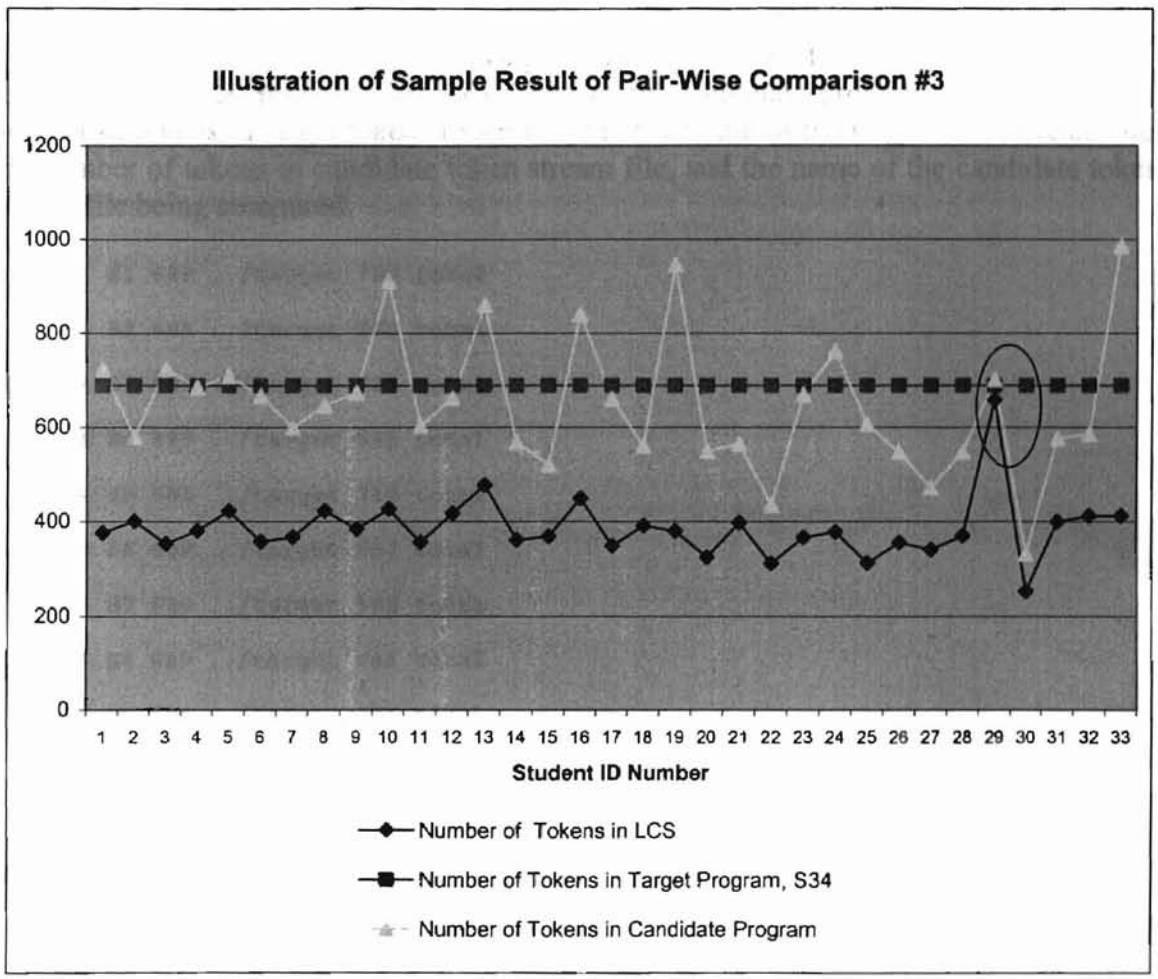
Illustration of Sample Result of Pair-Wise Comparison #3
1200
1000
800
600
400
200
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33
Student ID Number
Number of Tokens in LCS
Number of Tokens in Target Program, S34
Number of Tokens in Candidate Program

APPENDIX K

Sample Execution Result of Pair-Wise Comparison #3

Each row of the result has the following in order: number of tokens in LCS, directory name of the target program, directory name of the candidate program being compared, number of tokens in target token stream file, the file name of the target token stream file, the number of tokens in candidate token stream file, and the name of the candidate token stream file being compared.

```
376 S34 S1 689 ../target 723 total

401 S34 S2 689 ../target 580 total

353 S34 S3 689 ../target 725 total

381 S34 S4 689 ../target 685 total

423 S34 S5 689 ../target 710 total

357 S34 S6 689 ../target 667 total

367 S34 S7 689 ../target 599 total

423 S34 S8 689 ../target 646 total

384 S34 S9 689 ../target 674 total

426 S34 S10 689 ../target 912 total

356 S34 S11 689 ../target 603 total

417 S34 S12 689 ../target 662 total

477 S34 S13 689 ../target 861 total

361 S34 S14 689 ../target 568 total

368 S34 S15 689 ../target 520 total

449 S34 S16 689 ../target 841 total

348 S34 S17 689 ../target 661 total

391 S34 S18 689 ../target 560 total

380 S34 S19 689 ../target 946 total

324 S34 S20 689 ../target 550 total

397 S34 S21 689 ../target 565 total

311 S34 S22 689 ../target 435 total
```

```
366 S34 S23 689 ../target 669 total
378 S34 S24 689 ../target 763 total
312 S34 S25 689 ../target 607 total
355 S34 S26 689 ../target 547 total
340 S34 S27 689 ../target 472 total
370 S34 S28 689 ../target 547 total
658 S34 S29 689 ../target 702 total
252 S34 S30 689 ../target 332 total
399 S34 S31 689 ../target 576 total
412 S34 S32 689 ../target 586 total
412 S34 S33 689 ../target 986 total
```

APPENDIX L

Program from student S34 that was identified by JPD as being involved in plagiarism with the program of student S29 which appears in appendix M.

```
//=========================================================================
//
// Program: Target Program #3
//
// Author: S34
//
//=========================================================================

import java.io.*;
import java.util.*;
import java.text.*;
import java.math.*;

class pgm04  {

       public static void main (String[] args) throws IOException {
              BufferedReader stdin = new BufferedReader
                 (new InputStreamReader(System.in));

       GregorianCalendar calendar = new GregorianCalendar();

       if (!(stdin.ready())) {
          System.out.println ("NO input.");
          System.exit (0);}

       String input = stdin.readLine();
       StringTokenizer t = new StringTokenizer (input);
       String aa = "", bb="",cc="",  dd="", ee="", ff="",
       gg ="",ii ="";
       int tagC;
       int age=0 , a=0, b=0;

       while (t.hasMoreTokens()) {

       aa = t.nextToken();

       if (aa.equalsIgnoreCase("tag")) {
              bb = t.nextToken();
              cc = t.nextToken();
              dd = t.nextToken();
              gg = minimizeWhitespace(t);
              ee = tagNumber();
              tagC = (int)tagCost (Integer.parseInt (dd));
              System.out.println ( "TAG: $" + tagC + " " + ee +" " +
                           bb + " " + cc + " " + " -- " + gg);

             a++;
       }

       else if (aa.equalsIgnoreCase("license")) {
```

```
          ff = t.nextToken();
          gg = minimizeWhitespace(t);
          age = calcAge(ff);
          ii = licenseNumber();
          System.out.println("LIC: $5 " +ii + " " + ff + " " + age + " -- " + gg)
;
          b++;
          }

        if (!(stdin.ready())) break;

        input = stdin.readLine();
        t = new StringTokenizer(input);

        }
              System.out.println ("\nNumber of tags issued: " + a);
              System.out.println ("Number of licenses issued: " + b);

        }


//=====================================================================
// This method contain the way to generate random numbers.
//=====================================================================

        public static int dom (int aa, int bb)

        {
              Random k1 = new Random();

              int cc = bb - aa + 1;
              int i = k1.nextInt() % cc;

            if (i < 0 )
            {
              i = -i;
            }

              return aa + i;

        }


//=====================================================================
// This method contain the way to generate random letters.
//=====================================================================

        public static String randomstring ( int aa, int bb) {

        final int alphabet = 3;
        byte a[] = new byte [alphabet];

        for (int i = 0;
                i < alphabet;
                i++)

        {
         a[i] = (byte)dom(aa,bb);
        }

        return new String(a);
```

```
        }

//=====================================================================
//  This method contain the tag number
//=====================================================================

        public static String tagNumber() {

        DecimalFormat ff = new DecimalFormat ("000");

        final int aa = 0;
        final int bb = 999;
        int k1 = bb - aa + 1;
        Random tag_num = new Random();
        String alphabetString = randomstring('A' , 'Z');
        int tagNumber;
        String tagString = "";

        tagNumber = tag_num.nextInt() % k1;

        if (tagNumber < 0)
        {
           tagNumber = - tagNumber;
        }

        tagString = alphabetString + "-" +
String.valueOf(ff.format(tagNumber));

        return tagString;

        }



//=====================================================================
//  This method contain the license number
//=====================================================================

        public static String licenseNumber() {

        DecimalFormat ff = new DecimalFormat ("000000000");

        final long cc = 0;
        final long dd = 999999999;
        Random license = new Random();
        long k2 = dd - cc + 1;
        long number ;
        String licenseNumberString = "";

        number = license.nextLong() % k2;

        if ( number < 0)
        {
          number = - number;
        }

        licenseNumberString = String.valueOf(ff.format(number));

        return licenseNumberString;
        }

//=====================================================================
```

```
// This method contain the cost of the tag.
//=====================================================================

       public static float tagCost (int year)

       {

       GregorianCalendar calendar = new GregorianCalendar();
       int year1 = calendar.get(Calendar.YEAR);
       int number = year1 - year;
       float tag = (float)0.0;

       if ((number >= 0) && ( number <=3)) {
       tag = (float)85; }

       else if ((number >=4) && (number <= 7)) {
       tag = (float)75; }

       else if ((number >= 8) && (number <= 11)) {
       tag = (float)55; }

       else if ((number >= 12) && (number <= 15)) {
       tag = (float)35; }

       else if (number >= 16) {
       tag = (float)15; }

       return tag;

       }


//=====================================================================
// This method contain the format that how to retrieves all token from
//the StringTokenizer parameter, and format them into one string.
//=====================================================================

       public static String minimizeWhitespace (StringTokenizer t)

       {

       String num = " ";
       int tokenNumber = t.countTokens();

       while (tokenNumber > 1)
       {
        num += t.nextToken() + " ";
        tokenNumber --;
       }

       num += t.nextToken();

       return num;

       }

//=====================================================================
// This method contain the month, day and year.
//=====================================================================
       public static int calcAge(String date) {

       GregorianCalendar cal = new GregorianCalendar();
       StringTokenizer tokenizer  = new StringTokenizer (date, "/");
```

```
    String dateString, monthString, yearString;
    int mon1, date1, year1;

    monthString = tokenizer.nextToken();
    dateString = tokenizer.nextToken();
    yearString = tokenizer.nextToken();

    mon1 = cal.get(Calendar.MONTH) + 1;
    date1 = cal.get(Calendar.DATE);
    year1 = cal.get(Calendar.YEAR);

    if ((mon1 > Integer.parseInt (monthString)) ||
       ((mon1 == Integer.parseInt (monthString)) &&
        (date1 >= Integer.parseInt (dateString))))

    {
      return (year1 - Integer.parseInt (yearString));

    }
    else
    {
      return (year1 - Integer.parseInt(yearString) - 1);
    } }

}
```

APPENDIX M

Program from student S29 that was identified by JPD as being involved in plagiarism with the previous program of student S34 which appears in Appendix L.

```
//--------------------------------------------------------------------------
//
// Program: Candidate Program with high similarity with Target #3
//
// Author : S29
//
//--------------------------------------------------------------------------

import java.io.*;
import java.text.*;
import java.util.*;
import java.math.*;

public class pgm04 {

//--------------------------------------------------------------------------
//Program that will help the OSU tag agency issue license plates and driver's
//licenses.
//--------------------------------------------------------------------------


   public static void main(String args[]) throws IOException
   {
      //create new cal object.
      GregorianCalendar cal = new GregorianCalendar();

      //Initialization for following program.
      BufferedReader stdin = new BufferedReader
                 (new InputStreamReader(System.in));

      //if there is totally no input.
      if (!(stdin.ready()))  {
         System.out.println("No input.");
         System.exit (0);}//end if.

      String input = stdin.readLine();
      StringTokenizer t = new StringTokenizer (input);
      String fname="", mVeh="", vSty="", yVeh="", tagN="", dob="", name="",
      licN="";
      int tagC;
      int age=0, n=0, m=0;

      //Start doing processing in this loop.
      while (t.hasMoreTokens()) {  //Check whether the end of file.

            fname = t.nextToken();


            //If it is tag.
            if (fname.equalsIgnoreCase("tag")) {
```

```
            mVeh  = t.nextToken();
            vSty  = t.nextToken();
            yVeh  = t.nextToken();
            name  = minimizeWhitespace(t);
            tagN  = tagNumber();
            tagC  = (int)tagCost(Integer.parseInt(yVeh));
            System.out.println( "TAG: $"+tagC+" "+tagN+" "+mVeh+" "+vSty
                                +" "+yVeh+" -- "+name);
            n++;
            }//end if.

          //If it is license.
          else if (fname.equalsIgnoreCase("license")) {
            dob  = t.nextToken();
            name = minimizeWhitespace(t);
            age  = calcAge(dob);
            licN = licenseNumber();
            System.out.println( "LIC: $5 "+licN+" "+dob+" "+ age+
                                " -- "+name );
            m++;
            }//end else

            //Check if there still have input.
            if (!(stdin.ready())) break;

            //Read in next input.
            input = stdin.readLine();
            t = new StringTokenizer (input);
     }//end while

          //Print out total tags and lincenses issued.
          System.out.println();
          System.out.println("\nNumber of tags issued: " + n);
          System.out.println();
          System.out.println("Number of licenses issued: " + m);

   }//end main method

//-------------------------------------------------
//generate random number between a and b.
//-------------------------------------------------

public static int randomnumber (int a, int b)
{
        Random t = new Random();

        int num1 = b-a+1;
        int num2= t.nextInt()%num1;

        if (num2<0)
        {
          num2 = -num2;
        }
        return a + num2;
}

//-----------------------------------------------
//generate a random letter with 3 letters from a t
//-----------------------------------------------

public static String randomStr(int a,int b)
{
```

```
  final int Max = 3;

  byte A[] = new byte [Max];

  for (int num2=0; num2< Max; num2++)
  {
        A[num2] = (byte)randomnumber(a,b);
  }
  return new String (A);
}


//----------------------------------------
//generate and returns a tagnumber.
//----------------------------------------

public static String tagNumber()
{

   final int MIN = 0;
   final int MAX = 999;
   int i = MAX - MIN + 1;

   Random t = new Random();
   String lettersString = randomStr('A', 'Z');
   int tagnum;
   String tagstr = " ";

   tagnum = t.nextInt() % i;

        if (tagnum<0)
        {
          tagnum = -tagnum;
        }

        DecimalFormat fmt = new DecimalFormat ("000");

        tagstr = lettersString + "-" + String.valueOf (fmt.format(tagnum));

        return tagstr;
}


//-------------------------------------------------
//generate and return a license number.
//-------------------------------------------------

public static String licenseNumber()
{

  final long MIN_LIC = 0;
  final long MAX_LIC = 999999999;

  Random t = new Random();
  long num1 = MAX_LIC - MIN_LIC + 1;
  long license;
  String licenseStr= "";

  license = t.nextLong() % num1;

        if (license < 0)
        {
          license = -license;
```

```
        }

        DecimalFormat fmt = new DecimalFormat ("000000000");

        licenseStr = String.valueOf (fmt.format(license));


        return licenseStr;
}


//-----------------------------------------------------
//calculates and returns the tag cost.
//-----------------------------------------------------

public static float tagCost (int year)
{

  GregorianCalendar cal = new GregorianCalendar();
  int Cyear = cal.get(Calendar.YEAR);
  int Age = Cyear - year;
  float tagcost = (float) 0.0;


        if ((Age >= 0) && (Age <=3))
        {
          tagcost = (float) 85;
        }
        else if ((Age >= 4) && (Age <= 7))
        {
          tagcost = (float) 75;
        }
        else if ((Age >= 8) && (Age <= 11))
        {
          tagcost = (float)55;
        }
        else if ((Age >= 12) && (Age <= 15))
        {
          tagcost = (float) 35;
        }
        else if ((Age >=16))
        {
          tagcost = (float) 15;
        }

        return tagcost;
}


//-------------------------------------------------------------------------
//MinimizeWhitespace (StringTokenizer t).
//Retrieve all tokens from the StringTokenizer parameter, and format them into
//one string.
//-------------------------------------------------------------------------

public static String minimizeWhitespace (StringTokenizer t)
{

   String get = " ";
   int token = t.countTokens();

        while (token>1)
        {
```

```
            get += t.nextToken() +" ";
            token --;
          }

    get += t.nextToken();

    return get;
}


//-------------------------------------------------------------------------
//to calculate and return the age (in years) of the person born in that date.
//-------------------------------------------------------------------------

public static int calcAge (String date)
{
  GregorianCalendar cal = new GregorianCalendar ();
  StringTokenizer tokenizer = new StringTokenizer (date, "/");
  String monthstr, datestr, yearstr;
  int Cmonth, Cdate,Cyear;

  monthstr = tokenizer.nextToken();
  datestr = tokenizer.nextToken();
  yearstr = tokenizer.nextToken();

  Cmonth = cal.get (Calendar.MONTH) +1;
  Cdate = cal.get (Calendar.DATE);
  Cyear = cal.get (Calendar.YEAR);

        if ((Cmonth > Integer.parseInt (monthstr)) ||
            ((Cmonth == Integer.parseInt (monthstr)) &&
             (Cdate >= Integer.parseInt (datestr))))
        {
          return (Cyear - Integer.parseInt (yearstr));
        }

        else
        {
          return (Cyear - Integer.parseInt (yearstr) - 1);
             }
}
}
```

APPENDIX N

```
%{
//=============================================================================*
//     Program: Scanner                                                        *
//     Author: Dongchi Wang                                                    *
//     Advisor: Dr. Mansur H. Samadzadeh                                       *
//     Date: February 2001                                                     *
//     Programming Languages and tools: Flex and C                             *
//-----------------------------------------------------------------------------*
//     The source code for JPD consists of two parts: Scanner and TSM          *
// (Token Stream Matcher).                                                     *
//     Scanner is to recognize all meaningful tokens in a Java program.        *
// The input to the Scanner is a Java program without syntax errors. The       *
// output from the Scanner is a stream of tokens delimited by a white          *
// space. Tokens recognizable to the scanner include Java keywords,            *
// meaningful punctuation marks, constant numbers, and operators. Scanner      *
// is implemented using Flex and C.                                            *
//     The code for Scanner has three phases.                                  *
//     Phase I: also called Fpass meaning the first pass is to recognize       *
// names of classes defined in the program and names of methods defined        *
// in the classes.                                                             *
//     Phase II: also called Spass meaning the second pass is to tokenize      *
// method body and put the token stream as output into a file under the        *
// name of the method for each method.                                         *
//     Phase III: also called Tpass meaning the third pass is to collect       *
// and sequence the token stream for the Java program based on the             *
// calling order, and output the result into a file named "total".             *
//     TSM is to commit the pair-wise comparison for two token streams,        *
// each is output from the Scanner. The algorithm implemented here is          *
// LCS(Longest Common Stream). Some subtle changes have been done to fit       *
// in this project.                                                            *
//                                                                             *
//=============================================================================*
//=============================================================================*
//     Program: Scanner Phase I - Fpass                                        *
//     Author: Dongchi Wang                                                    *
//     Advisor: Dr. Mansur H. Samadzadeh                                       *
//     Date: February 2001                                                     *
//     Programming Languages and tools: Flex and C                             *
//-----------------------------------------------------------------------------*
//                                                                             *
//     This phase is to recognize names of classes defined in the program      *
// and names of all methods defined in the classes.                            *
//     A data structure, ClassStruc, has been defined in this program to       *
// hold class information which includes all method names defined in the       *
// class and the counters for left and right curly brakets. If there are       *
// more than one class defined in the Java program, class info will be         *
// organized into a linked list. Then, all recognized method names will        *
// be inserted to Phase II - Spass, a predefined program, and save the         *
// file Spass as a customized file, SPcust, which is to tokenize each          *
// method body and write the token stream of each method to a file by the      *
// name of the method. Phase I - Fpass also insert the recognized method       *
// names into Phase III - Tpass, another predefined program, and save          *
// Tpass as a customized file, TPcust.                                         *
//=============================================================================*
```

```
#include <unistd.h>
#include <stropts.h>
#include <stdio.h>
#include <memory.h>
#include <stddef.h>
typedef
struct MethodsStruc{              /* hold the data member for a method */
       char name[15];             /* method name */
       char *tokenStr[2000];
}MethodsStruc;

/* the following structure is used to contain a class name and all */
/* method names defined in the class */

typedef struct ClassStruc{
  char name[15];                  /* class name */
  MethodsStruc *methods[15];      /* method name array */
  struct ClassStruc * nextClass; /* pointer to next class */
  int LeftCurly;                  /* counter for left curly braket */
  int methodInd;                  /* the index of current method being processed
                                  */
}ClassStruc;

/*~~~ Globle variables ~~~*/

ClassStruc *classHead = NULL;     /* head of the class linklist */
ClassStruc *classTail = NULL;     /* tail of the class linklist */
ClassStruc * temp = NULL;         /* temporary pointer to the current class */
                                  /* being processed */

%}
%pointer
%%
"/*"    {     /* this block is to skip all comments inside */
              /* and */

        register int c;
        for( ; ; )
        {
              while((c = input()) != '*' && c != EOF)
                     ;            //do nothing
              if( c == '*' )
              {
                     while(( c = input()) == '*' )
                            ;
                     if( c == '/' )
                            break; //found the end of the comments
              }
              if( c == EOF )
              {
                     printf( "comment doesn't end correctly.\n");
                     break;
              }
         }
        }
"//"    {     /* eat up comment line following "//" */
              char c;
              while((c = input()) != '\n')
                     ;
         }
class  {

         char * tempname = (char *)malloc(15); //temperary name holder
```

```
  int i = 0;                /* index for the charactor array above */
  char c;                   /* current char being processed */
                            /* current class being processed */
  temp = (ClassStruc *)malloc(sizeof(ClassStruc));
  if(temp == NULL)
  {
       printf("Memory allocation Error.\n");
       exit(1);
  }
  strcpy(tempname,"");      /* clear the garbage of the place holder */
  temp->LeftCurly=0;        /* initial the left curly brakets counter */
temp->methodInd = 0;        /* initial the method index currently */
                            /* being processed */
c = input();          //get the first char
while((c == ' ')||(c == '\t')) //skip the first wild space char
       c = input();
while((c != ' ')&&(c != '\t')&&(c != '{')&&(c !='\0')&&(c!='\n'))
{
       tempname[i] = c;     /* get the name of the class */
       i++;                 /* increment the char index */
       c = input();         /* get the next meaningful char */
}
tempname[i] = '\0';         /* end the name string by null char */
unput(c);                   /* put the '{' back if applicable */

  /* --- to link the class structures ---*/

if(classHead == NULL)
{
       strcpy(temp->name, tempname);     /* set up the head and tail */
                                         /* pointer */
       classTail = classHead = temp;
       classTail->nextClass = NULL;
 }
 else
 {
       strcpy(temp->name, tempname);
       classTail->nextClass = temp;
       classTail = temp;
       classTail->nextClass = NULL;
 }
}
[a-zA-Z _\t]+[ \t]*[(][ _\ta-z[:punct:][:alnum:]]*[)][[:alpha:]* \t\n]*[{]    {

/* this block is to locate the method body in a class */

int len=0;
char c;
char *tempstr = malloc(15);
char *temp2, *temp3;
int j = temp->methodInd;    /* remember the next method index to be */
                            /* processed */
temp->LeftCurly++;          /* left curly bracket counter is a signof */
                            /* method body */
if(temp->LeftCurly == 2)    /* find a method body */
{
       temp->methods[j] = (MethodsStruc *)malloc(sizeof(MethodsStruc));
       strcpy(temp->methods[j]->name, "");
       tempstr=(char *)strtok(yytext, "(");/* trunk the string before()*/
       temp2=(char *)strtok(tempstr, " \t"); /* get the first token */

       while(temp2 != NULL)
       {
```

```
                temp3 = temp2;      /* remember the previous token */
                temp2=(char *)strtok(NULL, " \t");
            }

        /* the token right before () would be the name of the method */

            strcpy(temp->methods[j]->name, temp3);
            temp->methodInd++;
        }
    }
[{]             {     temp->LeftCurly++;  }
[}]             {
                      temp->LeftCurly--;
                }
[[:alpha:]]+[_[:alnum:]]*
[[:digit:]]
[[:punct:]]
[\n]
%%
main(int argc, char ** argv)
{
    int i;
    ClassStruc * tempcl;        /* holder for the temporary class */
    FILE *SecPass;              /*file pointer to the template second pass */
    FILE *Customerized;         /*file pointer to the customized second pass
                                */
    FILE *ThirdIn, *ThirdOut;
    char *inputS = malloc(80); /*hold a line of input */
    char *theKey = malloc(20); /*hold the first token of the line above */
        strcpy(inputS, ""); /*clear those strings above */
    strcpy(theKey, "");

    if(argc != 2)
    {     printf("Can't find input file, now use standard input\n");
          yyin = stdin;
    }
    else
          yyin = fopen(argv[1], "r");
    yylex();
    printf("\n");

    /*================================================================
       insert the definition of the set of programmer defined
       methods in the predefined program, Spass(Second Pass) and
       Tpass(Third Pass)
     ================================================================*/

    if((SecPass = fopen("../../Spass", "r"))== NULL)
    {
          printf("Unable to open input file, Spass. \n");
          exit(1);
    }
    if((Customerized = fopen("../../SPcust", "w+"))==NULL)
    {
          printf("Unable to open input file, SPcust. \n");
          exit(1);
    }
    if((ThirdIn = fopen("../../Tpass", "r")) == NULL)
    {
          printf("Unable to open input file, Tpass. \n");
          exit(1);
    }
    if((ThirdOut = fopen("../../TPcust", "w+"))==NULL)
```

```
        {
                printf("Unable to open input file, TPcust. \n");
                exit(1);
        }

        /*============================================================
           copy each line of the file, Spass, and insert the method
           names in the form of method1|method2|method3|...|methodn
           at the end of the line starting by Methods
          ============================================================*/

        while(fgets(inputS, 80, SecPass)!=NULL)
        {
           sscanf(inputS, "%s", theKey);
           if(strcmp(theKey, "Methods")==0)
           {
               fputs("Methods ", Customerized);
               for(tempcl=classHead; tempcl != NULL; tempcl = tempcl->nextClass)
               {
                       i = 0;
                       if(tempcl->nextClass==NULL)
                       {
                               while(tempcl->methods[++i]!=NULL)
                                       fprintf(Customerized, "%s|", tempcl-
>methods[i-1]->name);
                               fprintf(Customerized, "%s", tempcl->methods[i-1]-
>name);
                               break;
                       }
                       while(tempcl->methods[i]!=NULL)
                               fprintf(Customerized, "%s|", tempcl->methods[i++]-
>name);
               }
               fprintf(Customerized, "\n");
            }
            else
               fprintf(Customerized, "%s", inputS );
        }
        fclose(SecPass);
        fclose(Customerized);
        strcpy(inputS, "");
        strcpy(theKey, "");
        while(fgets(inputS, 80, ThirdIn)!=NULL)
        {
           sscanf(inputS, "%s", theKey);
           if(strcmp(theKey, "Methods")==0)
           {
               fputs("Methods ", ThirdOut);
               for(tempcl=classHead; tempcl != NULL; tempcl = tempcl->nextClass)
               {
                       i = 0;
                       if(tempcl->nextClass==NULL)
                       {
                               while(tempcl->methods[++i]!=NULL)
                               fprintf(ThirdOut, "%s|", tempcl->methods[i-1]->name);
                               fprintf(ThirdOut, "%s", tempcl->methods[i-1]->name);
                               break;
                       }
                       while(tempcl->methods[i]!=NULL)
                               fprintf(ThirdOut, "%s|", tempcl->methods[i++]->name);

               }
               fprintf(ThirdOut, "\n");
```

```
            }
            else
            fprintf(ThirdOut, "%s", inputS );
        }
        fclose(ThirdIn);
        fclose(ThirdOut);
        printf("\n");
}
```

APPENDIX O

```
%{
//============================================================================*
//     Program: Scanner Phase II - Spass                                      *
//     Author: Dongchi Wang                                                   *
//     Advisor: Dr. Mansur H. Samadzadeh                                      *
//     Date: February 2001                                                    *
//     Programming Languages and Tools: Flex and C                            *
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*
//                                                                            *
//     Phase II - Spass                                                       *
//     This program is to tokenize each method body and write the token stream *
// as output into a file under the name of the method. It processes methods   *
// one after another.                                                         *
//     This is just a template predefined for this process since each program *
// has methods with different names so that the match-pattern for each program*
// varies. At the end of the processing of phase I - Fpass, a file by the name*
// of SPcust(customized second pass) will be opened and written by copying    *
// Spass line by line except for inserting a sequence of method names         *
// delimited by | to the right of the line of the specification, Methods.     *
//                                                                            *
//============================================================================*

#include <stdio.h>
#include <memory.h>
#include <stddef.h>
//~~~globle variables~~~
FILE *fp;     /* file pointer to the current method-token-stream file */
int Left;     /* counter for '{', a sign of the end of a method body */
%}
%pointer
Methods       /* method names go here once Fpass finishes running */
%%
"/*"   {
              /* This block is to skip comments */

       register int c;
              for( ; ; )
              {
                     while((c = input()) != '*' && c != EOF)
                            ;                    /* skip */
                     if( c == '*' )
                     {
                            while(( c = input()) == '*' )
                                          ;
                            if( c == '/' )
                                   break;        /* found the end of comments */
                     }
                     if( c == EOF )
                     {
                            printf( "comment doesn't end correctly.\n");
                            break;
                     }
              }
       }
```

```
"//"          {
                                /* skip the // style comments */

                      char c;
                      while((c = input()) != '\n')
                            ;
              }
{Methods}+[ \t]*{()[ \t[:punct:][:alnum:]]*[)][ \t\n]*[[:alpha:]+[ \t\n]*]*[{]
       {

              /* this block is to tokenize method body */
              char *methodName;    /* hold the method name */

              /* recognize the name of the method */
              methodName = (char *)strtok(yytext, " \t(");

              /* open a file to write by the name of the method */
              fp = fopen(methodName, "w+");

              /* initialize the counter of { to 1 */
              Left = 1;
       }

[{]           {      Left++;              /* incremented the counter of {  */
       }
[}]    {
              Left--;                     /* decremented the counter of {  */

              /* in the case of exit from the method body, close the file
              pointer */
              if(Left == 0)
              {
                      fclose(fp);
                      fp = NULL;
              }
       }
while|for   {
              /* substitute "while" and "for" control keywords with the same
              token  */
              if(fp != NULL)
              fprintf(fp, "While_For ");
           }
if|else[ \t]%if|case       {
              /* substitute these keywords with the same token  */
                      if(fp != NULL)
                      fprintf(fp, "If_Else_Switch_Case ");
                }
else|default  {
              /* substitute "else" and "default" keywords with the same token
              */
                      if(fp != NULL)
                      fprintf(fp, "Else_Default ");
              }

do|switch|return|continue|new|throws|throw|break|finally|final|try|catch  {
              /* keep all of these keywords */
              if(fp != NULL)
                      fprintf(fp, "%s ", yytext);
              }
{Methods}[ \t]*[()[ \t[:punct:][:alnum:]]*[)]          {

              /* this block is to recognize method calls no matter if it is */
              /* a library method or programmer-defined method, this process */
```

```
                /* keep them all in order to expend them in the third pass   */

        char *methodName;
        char *marray[5];     /* array of method names */
        char *dot;
        char *dot1;          /* the back part of a function call after the dot */
        char *mark;
        int i = 0;           /* counter for the current nested method */
        methodName = (char *)strtok(yytext, "(");
        if(fp != NULL)
        {
                /* in the case of nested methods, */
                /* push them into a stack then pop them into the token stream */

                mark = methodName;
                while(methodName != NULL)
                {
                        marray[i] = (char *)malloc(20);
                        strcpy(marray[i], methodName);
                        methodName = (char *)strtok(NULL, "(");
                        if((mark = (char *)strchr(methodName, ')'))!=NULL)
                                break;
                        else if((mark = (char *)strchr(methodName, '.'))!=NULL)
                                methodName = mark;
                        i++;
                }
                while(i!=-1)
                        fprintf(fp, "%s ", marray[i--]);
        }
                        }
[.][ \t_[:alpha:]]+[(]      {
                        char *methodName;
                        methodName = (char *)strtok(yytext, " \t(");
                        if(fp != NULL)
                        fprintf(fp, "%s ( ", methodName);
                        }
[[:alpha:]][[:alnum:]_]*
[[:digit:]]   {
                        if(fp != NULL)
                        fprintf(fp, "%s ", yytext);
                }
[[:punct:]]   {
                        if(fp != NULL)
                        fprintf(fp, "%s ", yytext);
                }
[\n]
%%
main(int argc, char ** argv)
{
        if(argc != 2)
        {       printf("Can't find inputfile.\n");
                yyin = stdin;
        }
        else
                yyin = fopen(argv[1], "r");
        yylex();
        printf("\n");
}
```

APPENDIX P

```
%{
//=============================================================================*
//     Program: Scanner Phase III – Tpass                                      *
//     Author: Dongchi Wang                                                    *
//     Advisor: Dr. Mansur H. Samadzadeh                                       *
//     Date: February 2001                                                     *
//     Programming Languages and Tools: Flex and C                             *
//-----------------------------------------------------------------------------*
//                                                                             *
//     This program is to put all tokens in a Java program together based on   *
// the calling order.                                                          *
//     This is just a template predefined for this process since each program *
// has methods with different names and the match-pattern for each program     *
// varies. At the end of the processing of phase I - Fpass, a file by the      *
// name of TPcust(customized third pass) will be opened and written by         *
// copying Tpass line by line except for inserting a sequence of method names *
// delimited by | to the right of the line of the specification, Methods.      *
// Starting from the "main" method, TPcust process each token. When it meet a  *
// programmer-defined method call, it expands it in place.                     *
//                                                                             *
//=============================================================================*

#include <stdio.h>
#include <memory.h>
#include <stddef.h>
FILE *fp;     /* pointer to the current method-token-stream file */
FILE *totalfp;
%}
%pointer
Methods       /* method names go here once Fpass finishes running */
%%
{Methods}     {
              /* whenever meeting a programmer-defined method      */
              /* the file by the name of the method will be opened */
              /* and the token stream in the file will be copied   */
              /* into the  file main.                              */

                    char *buffer = malloc(80);
                    fp = fopen(yytext, "r");
                    strcpy(buffer, "");
                    while(fgets(buffer, 80, fp)!= NULL)
                          fputs(buffer, totalfp);
                    fclose(fp);
              }
              {
                    fputs(yytext, totalfp);
              }
%%
main(int argc, char ** argv)
{
      if(argc != 2)
      {     printf("Can't find inputfile.\n");
            yyin = stdin;
      }
```

```
        else
                yyin = fopen(argv[1], "r");

/* a file, total, has been specified to contain all token streams */

        if((totalfp = fopen("total", "wr+"))==NULL)
        {
                printf("Can't open the file, total\n");
                exit(1);
        }
        yylex();
        fclose(totalfp);
        printf("\n");
}
```

APPENDIX Q

```
//=========================================================================*
//     Program: TSM - Token Stream Matcher                                 *
//     Author: Dongchi Wang                                                *
//     Advisor: Dr. Mansur H. Samadzadeh                                   *
//     Date: February 2001                                                 *
//     Programming Language: C                                             *
//-------------------------------------------------------------------------*
//                                                                         *
//     This program is to implement the longest common stream (LCS)        *
// algorithm to make pair-wise token stream matching.                      *
//     It takes two token stream file as input and outputs the number of   *
// tokens in the common longest string of both file.                       *
//     The traditional LCS algorithm has been used with a little change    *
// to fit in our need that is adding a integer variable count to keep      *
// track of the number of tokens in LCS for the final results.             *
//     The longest token stream this program can handle has a limit of     *
// 1000. In case of longer streams to be compared, a change of the         *
// dimension in a global variable Table needs to be made.                  *
//                                                                         *
//=========================================================================*

#include <stdio.h>
/* data structure for an element in tables b and c    */
typedef
struct CBcell {
       char arrow;
       int number;
}Cell;
int count = 0;              /* counter for the number of tokens in LCS     */
Cell Table[1000][1000];     /* This number can be customerized             */
                            /* print the common longest token stream       */
int Print_lcs(Cell t[][], char *s[], int i, int j);
main(int argc, char * argv[])
{
       char fstr[1000];     /* arrays to hold the token streams in order to*/
       char fstr2[1000];    /* find the LCS among the two token stream files */
       FILE f1;             /* file pointers to the token stream files     */
       FILE *f2;
       int i=0;             /* index */
       int j=0;
       int xi = 0;
       int yj = 0;
       char *buf1;
       char *buf2;
       if((f1 = fopen(argv[1], "r"))==NULL)
       {      printf("can't open the target file %s\n", argv[1]);
              Exit(1); }
       buf1 = (char *)malloc(80);
       buf2 = (char *)malloc(80);
       strcpy(buf1," ");
       strcpy(buf2, " ");
       while(i!=1000)
```

```
        {

                j = 0;
                /* initiate the number of common tokens as 0 in Table */
                while(j!=1000)
                {
                        Table[i][j].number = 0;
                        j++;
                }
                i++;
        }
        i =  1;
        j = 1;

        /* open the two token stream files that need to be compared */
        /* and copy them into two token arrays accordingly          */

        if((f2 = fopen(argv[2], "r"))==NULL)
        {       printf("Can't open the candidate file %s\n", argv[2]);
                Exit(1); }
        for(fscanf(f2, "%s", buf1); strcmp(buf1, "")!=0; fscanf(f2, "%s", buf1))
        {
                fstr2[i] = (char *)malloc(20);
                strcpy(fstr2[i++],buf1);
                strcpy(buf1,"");
        }
        xi = i-1;
        i=1;
        for(fscanf(f1, "%s", buf2); strcmp(buf2, "")!=0; fscanf(f1, "%s", buf2))
        {
                fstr[i] = (char *)malloc(20);
                strcpy(fstr[i++], buf2);
                strcpy(buf2,"");
        }
        yj = i-1;
        i=1;
        /* to build up the Table which keep track */
        /* of the number of the common tokens     */
        while(i<=xi)
        {
                j=1;
                while(j<=yj)
                {       if(strcmp(fstr2[i],fstr[j])==0)
                        {       Table[i][j].number = Table[i-1][j-1].number+1;
                                Table[i][j].arrow = '\\';
                        }
                        else if(Table[i-1][j].number >= Table[i][j-1].number)
                        {       Table[i][j].number = Table[i-1][j].number;
                                Table[i][j].arrow = '|';
                        }
                        else
                        {       Table[i][j].number = Table[i][j-1].number;
                                Table[i][j].arrow = '-';
                        }
                        j++;
                }
                fclose(f2);
                i++;
        }
        Print_lcs(Table, fstr2, xi, yj);
        printf("%d", count);
}
/* recursive function to output the LCS among the two token streams */
```

```
int Print_lcs(Cell t[][1000], char *s[], int i, int j)
{
	if(( i==0)||(j == 0))
		return;
	if(t[i][j].arrow == '\\')
	{
		Print_lcs(t, s, i-1, j-1);
		count++;
	}
	else if(t[i][j].arrow == '|')
		Print_lcs(t, s, i-1, j);
	else
		Print_lcs(t, s, i, j-1);
}
```

APPENDIX R

```
//=============================================================================*
//     ShellScript1                                                            *
//     Author: Dongchi Wang                                                    *
//     Advisor: Dr. Mansur H. Samadzadeh                                       *
//     Date: February 2001                                                     *
//     Programming Language: Unix shell script                                 *
//-----------------------------------------------------------------------------*
//                                                                             *
//     This shell script is to go into each student directory to run the       *
// three passes of JPD Scanner. The final output is going to be put into       *
// a file called total which is a token stream of a Java program.              *
// lex.yy.c is a program generated by flex when flex run each pass. It is      *
// overwritten every time any pass of the scanner runs.                        *
// The Java program being tested here is by the name of pgm04.java, you        *
// may substitute it by any other Java program.                                *
//                                                                             *
//=============================================================================*


#/bin/csh
cd /e/wdongch/thesis/folders
foreach directory(*)
        cd $directory
                    flex ../../Fpass
                    gcc -g lex.yy.c -o First -lfl
                    First pgm04.java
                    flex ../../SPcust
                    gcc -g lex.yy.c -o Second -lfl
                    Second pgm04.java
                    flex ../../TPcust
                    gcc -g lex.yy.c -o Third -lfl
                    Third main
        cd ..
  end
cd ..
```

APPENDIX S

```
//=========================================================================*
//     ShellScript2                                                        *
//     Author: Dongchi Wang                                                *
//     Advisor: Dr. Mansur H. Samadzadeh                                   *
//     Date: February 2001                                                 *
//     Programming Language: Unix shell script                             *
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*
//                                                                         *
//     This shell script is to go into each student directory to copy the  *
// token stream file total generated by JPD scanner to a file by the name  *
// of target which is going to be compared to each file total from         *
// different student directory for similarity by program TSM (Token Stream*
// matcher). The number of tokens in LCS (Longest Common Stream) is going  *
// to be output followed by the target student directory and the candidate*
// student directory. Then, the number of tokens in file target and the    *
// number of tokens in file total is also going to be output. The detailed*
// output are in APPENDIX C, G AND K.                                      *
//                                                                         *
//=========================================================================*

#/bin/csh
#include<string.h>
#include<stdio.h>
cd /e/wdongch/thesis/folders
foreach directory(*)
             if($directory != "target") then
        cd $directory
        cp total ../target

        cd ..
        endif
        foreach dire(*)
             if ( $dire != $directory ) then
                    if( $dire != "target") then
                    cd $dire
                    ../../TSM ../target total
                    echo $directory
                    echo $dire
                    wc -w ../target
                    wc -w total
                    printf "\n"
                    cd ..
                    endif
             endif
        end
end
cd ..
```

VITA

Dongchi Wang 2

Candidate for the Degree of

Master of Science

Thesis: TOWARD PLAGIARISM DETECTION IN JAVA PROGRAMS

Major Field: Computer Science

Biographical:

Personal Data: Born in Shenyang, P.R. of China, February 19, 1968, daughter of Taishun Wang and Junjiang Chi.

Education: Received Bachelor of Arts from Liaoning University in July 1990. Obtained Law degree from Law and Politics University of China in July 1992 , Beijing. Completed the requirements for Master of Science in Computer Science at the Computer Science Department at Oklahoma State University in May 2002.

Experience: Employed as a CIS (Computer and Information Services) mainframe operator from spring 1998 to fall 1999. Employed as a Teaching Assistant from January 2000 to December 2000.