Using “MOSS” – Plagiarism Detection Software

UB Introduction Manual for Professors and TAs

December 19, 2008

Dr. Bina Ramamurthy (bina@cse.buffalo.edu)
Scott Settembre (ss424@cse.buffalo.edu)

Moss - Measure Of Software Similarity

This primer is designed to quickly familiarize Professors and Teacher Assistants with the "MOSS" Plagiarism Detection Software (PDS). It will be helpful to read this prior to giving instructions to students on how you wish them to submit their coding projects, since proper submission of assignments will speed the process considerably (and your TA's will thank you for it).

MOSS Server and Authors

You can find out more about the PDS called "MOSS" at: http://theory.stanford.edu/~aiken/moss/

However, to save you the time, MOSS was developed and incrementally enhanced since 1994. It uses several metrics outlined in the paper: http://theory.stanford.edu/~aiken/publications/papers/sigmod03.pdf

Applicability at UB

MOSS is effective in detecting similar structures in nearly every different type of language that we use here at the University at Buffalo, including C, C++, Java, **LISP**, Perl, Python, Scheme, **Matlab** and Prolog. As you can guess, it is not fooled by renaming variables, rearranging or functionalizing blocks of code, extracting code to different modules or files, or the placing of NOP or junk code between useful codes. MOSS can also *exclude* matching base files (code supplied by instructors or TAs) and it can *exclude* the consideration of code that is matched across many different submissions (code perhaps supplied by a coding book or website, like a sorting algorithm, that is not central to the assignment).

Effectiveness

In a practical use scenario here at UB (in a core class), MOSS discovered 20 students (out of 93) using from 1000-3800 lines of shared code *each*. In comparison, the TA's were only effective in ferreting out 2 of those 20 students prior to running the PDS. At first all 20 students denied involvement in cheating, but when faced with the PDS report and MOSS's code matching feature, they all confessed.

Results

A one month assignment from about 100 students can be submitted and assessed within a single minute. Results come in two forms, an overall class summary page and an offending pair code matching window. The report will remain on the MOSS server for 20 days. There is no limit to the number of files that can be submitted.

The main summary page (See Figure 1) pairs each student with the other student who has matching code. If more than one student is involved and has shared the same code, multiple line entries are made for each student, ranked by the number of lines of similar code.

The offending pair code matching window (See Figure 2) uses *color highlighting* to show matches from each block of code that is similar between each student. These windows display ALL files of a project concatenated together and allows a Professor or TA to easily click on a color and display side by side the matching code. (Figure 3 and 4 show additional matching techniques in action.)

[Note: In our experience, there are no 'coincidental' matches when the percentages are higher than 25%.]

How to Use

1. Register by reading and emailing appropriately from the bottom of the webpage:

   http://theory.stanford.edu/~aiken/moss/

   (feel free to use the script from Appendix A.)
2. In the email that will return to you almost immediately, you will need to cut and paste the perl script (being careful not to mess up the LF formatting – so best to do this on LINUX). [note: the email version of this script from the website has been discontinued, so you must use the perl script from the email]
3. You will not need to modify the script at all for use at UB.

MOSS works best to check all the files of a particular assignment. MOSS can be set to consider all files in a directory to be part of the same project. In our experimentation, we created a directory in each student's submit directory for each project. Then we ran MOSS on each section/student/project directory at the same time.

For example:

perl moss -l c -m 40 -d /submit/bina/*/*/project3/*.c

This will:

1. Run the script "moss" using perl
2. "-l c" = use the language c (use "-l Matlab" or "-l LISP", other options can be found in the script)
3. "-40 m" = ignore duplicate code if it appears in more than 40 students' projects
4. "-d" = consider all files in a directory as part of the same project
5. "/submit/bina/*/*/project3/*.c" = use all the "c" files in the directory "project3" that can be found in either the "cse421" or "cse521" directories (first *) of all "usernames" (second *). If you are a managing more than one class, you can change it to something like "/submit/profname/classname/*/project1/*.Matlab" which will just use the "project1" directories of all the students in the class "classname"

The perl script with then submit all the files (to the PDS server) from the path as described and return a webpage as a reference. Remember, this web page will only remain on the server for 20 days.

If you need assistance in setting up moss or a demonstration on how to apply it, please feel free to contact ss424@cse.buffalo.edu . We can give you tips on how to organize submissions as well as running and configuring the MOSS PDS for specific issues. (See Appendix A for the MOSS script)

How to Interpret Results

After many hours of use, and clicking on various matches, we have found that the primary metric to determine if cheating occurred is the lines of code matched. Just using percentages, we would find that students with heavily “padded” code or commented code would be able to lower their percentage in the listing and escape detection.

It would be up to each instructor to determine how many lines of code would constitute misconduct. The chance that 25% of an individual’s code is identical to another student without any code sharing is extremely low, so that is where we decided to place the cutoff point. After several hours of looking at code around that 25% borderline, I can easily say that in a project where code sharing is not permitted, you will find a lot of evidence of direct cheating. Anything 60% or above is an absolute indication of MAJOR code sharing, which for our projects constituted above 1000-3800 copied lines of code.

[Note: Cheating at this level excludes accidental similiarities, similiarities due to side-by-side individual but ‘collaborative’ programming (which is still not allowed), talking about the project with other students by voice/phone/conversation, and pure coincidence. Violators at high levels of detection will definitely be found to have clearly resorted to using **CPR** (cut/paste/rename/reassemble) on their project.]

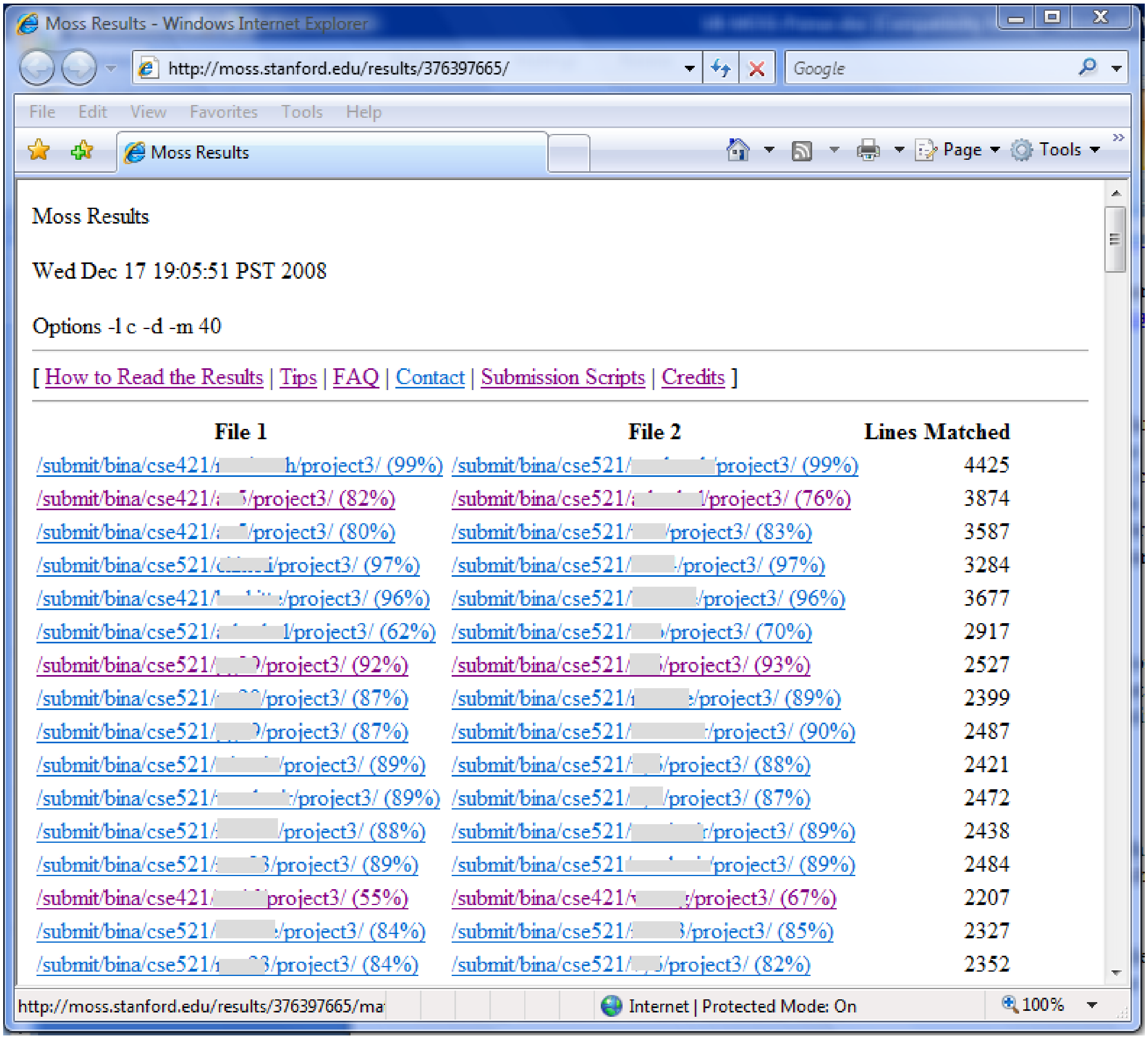
Why use PDS?

Code sharing is a direct violation of the code of conduct at UB. Code sharing in large class can be prevented easily by using MOSS. Good, honest students look across the computer labs at clusters of other students working and collaborating on projects and say to themselves, “Why doesn’t anyone do anything about this?” We can.

The assignments that we give as instructors are not only supposed to teach, but also to build confidence in a student. By allowing these cheaters to slip through the cracks by not safeguarding our grades, we rob them of the opportunity to know that they do not need to cheat, that they CAN indeed complete the assignment.

Cheaters hurt everyone, including themselves, our department, and our university’s good name. We can teach these errant students otherwise by making it known that our “threat” of using PDS in classes is no longer a bluff. It should be used regularly and used in all classes that have large amounts of students and significant coding.

Both your honest students and your TA’s will thank you for it!


Moss Results - Windows Internet Explorer
File
Edit
View
Favorites
Tools
Help
Moss Results
Page
Tools
Internet | Protected Mode: On
100%

http://moss.stanford.edu/results/376397665/

Moss Results

Wed Dec 17 19:05:51 PST 2008

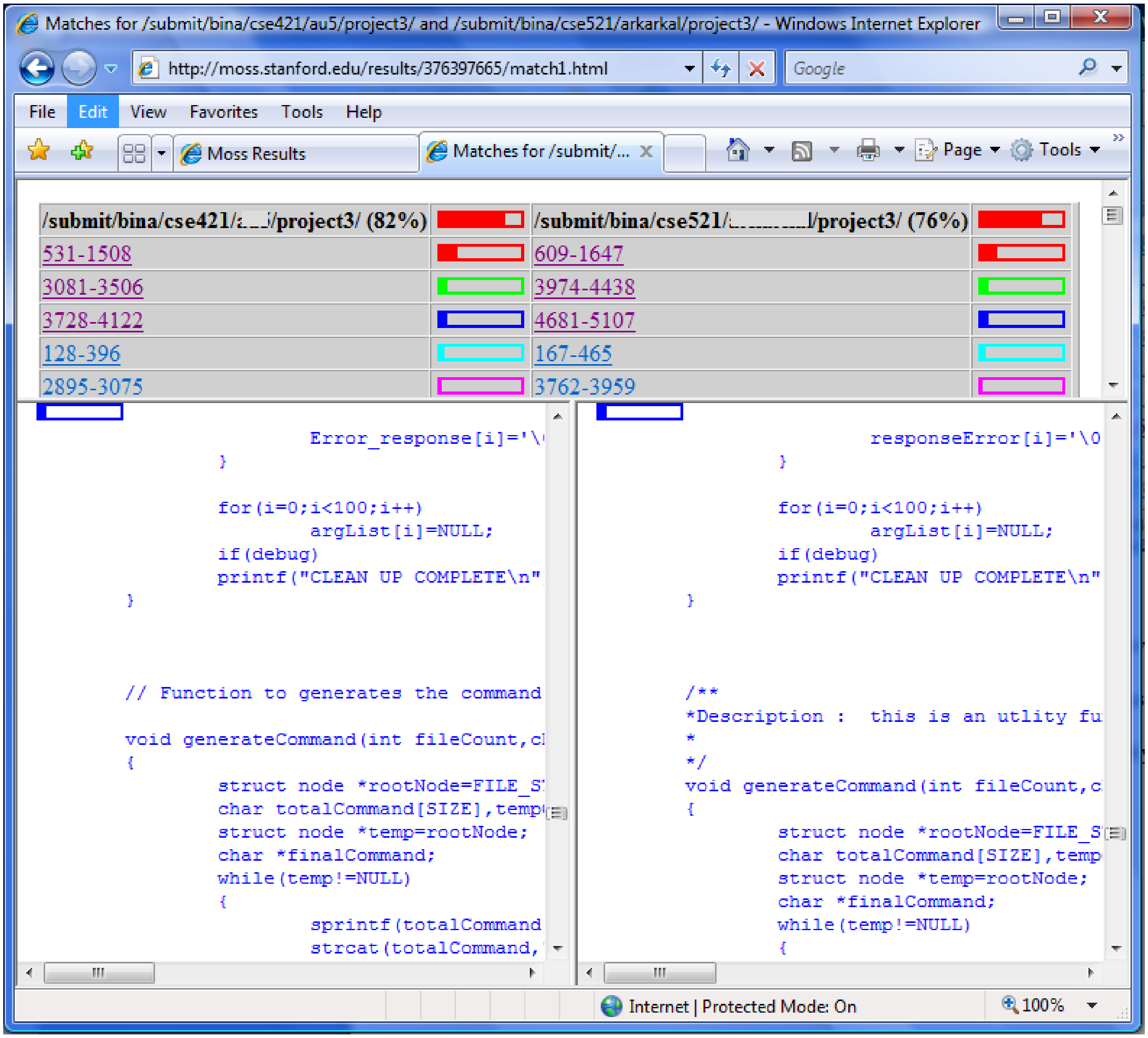
Options -l c -d -m 40

[How to Read the Results | Tips | FAQ | Contact | Submission Scripts | Credits]

File 1	File 2	Lines Matched
/submit/bina/cse421/[redacted]/project3/ (99%)	/submit/bina/cse521/[redacted]/project3/ (99%)	4425
/submit/bina/cse421/[redacted]/project3/ (82%)	/submit/bina/cse521/[redacted]/project3/ (76%)	3874
/submit/bina/cse421/[redacted]/project3/ (80%)	/submit/bina/cse521/[redacted]/project3/ (83%)	3587
/submit/bina/cse521/[redacted]/project3/ (97%)	/submit/bina/cse521/[redacted]/project3/ (97%)	3284
/submit/bina/cse421/[redacted]/project3/ (96%)	/submit/bina/cse521/[redacted]/project3/ (96%)	3677
/submit/bina/cse521/[redacted]/project3/ (62%)	/submit/bina/cse521/[redacted]/project3/ (70%)	2917
/submit/bina/cse521/[redacted]/project3/ (92%)	/submit/bina/cse521/[redacted]/project3/ (93%)	2527
/submit/bina/cse521/[redacted]/project3/ (87%)	/submit/bina/cse521/[redacted]/project3/ (89%)	2399
/submit/bina/cse521/[redacted]/project3/ (87%)	/submit/bina/cse521/[redacted]/project3/ (90%)	2487
/submit/bina/cse521/[redacted]/project3/ (89%)	/submit/bina/cse521/[redacted]/project3/ (88%)	2421
/submit/bina/cse521/[redacted]/project3/ (89%)	/submit/bina/cse521/[redacted]/project3/ (87%)	2472
/submit/bina/cse521/[redacted]/project3/ (88%)	/submit/bina/cse521/[redacted]/project3/ (89%)	2438
/submit/bina/cse521/[redacted]/project3/ (89%)	/submit/bina/cse521[redacted]/project3/ (89%)	2484
/submit/bina/cse421/[redacted]/project3/ (55%)	/submit/bina/cse421/[redacted]/project3/ (67%)	2207
/submit/bina/cse521/[redacted]/project3/ (84%)	/submit/bina/cse521/[redacted]/project3/ (85%)	2327
/submit/bina/cse521/[redacted]/project3/ (84%)	/submit/bina/cse521/[redacted]/project3/ (82%)	2352

http://moss.stanford.edu/results/376397665/ma

Figure 1. Main Summary Page – accessed by web browser. The top line shows a project that matched 99% since it was from a student that submitted the same code twice in two different directories. All other lines show percentages of matching in an approximated 4-5K line project. (Names are grayed out to protect the guilty…)

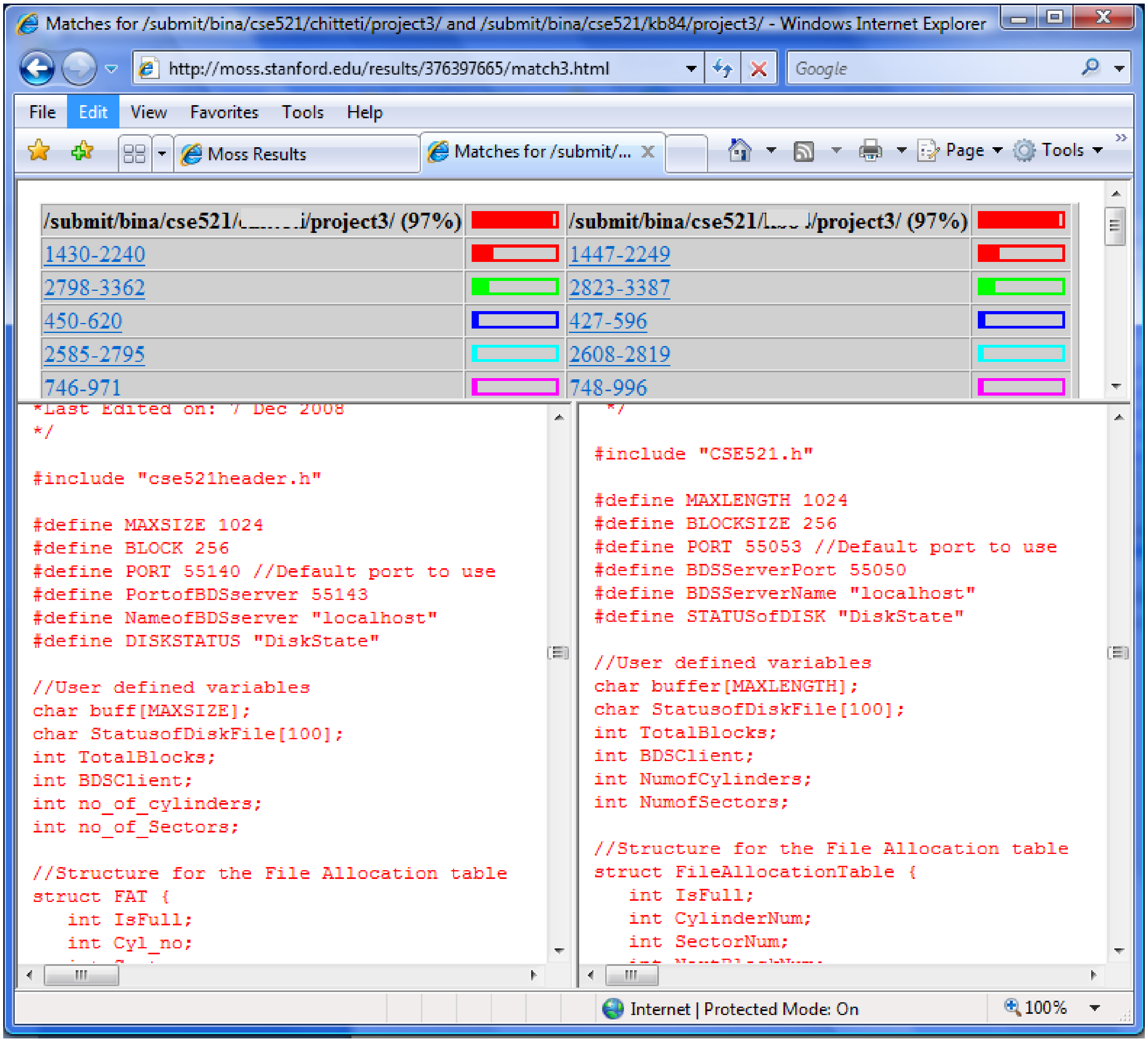

Matches for /submit/bina/cse421/au5/project3/ and /submit/bina/cse521/arkarkal/project3/ - Windows Internet Explorer
http://moss.stanford.edu/results/376397665/match1.html
Google
File
Edit
View
Favorites
Tools
Help
Moss Results
Matches for /submit/...
Page
Tools
Internet | Protected Mode: On
100%

/submit/bina/cse421/[redacted]/project3/ (82%)	🟥	/submit/bina/cse521/[redacted]/project3/ (76%)	🟥
531-1508	🟥	609-1647	🟥
3081-3506	🟩	3974-4438	🟩
3728-4122	🟦	4681-5107	🟦
128-396	🟦	167-465	🟦
2895-3075	🟪	3762-3959	🟪

```
                        Error_response[i]='\
                }

                for(i=0;i<100;i++)
                        argList[i]=NULL;
                if(debug)
                printf("CLEAN UP COMPLETE\n"
        }



        // Function to generates the command

        void generateCommand(int fileCount,cl
        {
                struct node *rootNode=FILE_S
                char totalCommand[SIZE],temp
                struct node *temp=rootNode;
                char *finalCommand;
                while(temp!=NULL)
                {
                        sprintf(totalCommand
                        strcat(totalCommand,'
```

```
                        responseError[i]='\0
                }

                for(i=0;i<100;i++)
                        argList[i]=NULL;
                if(debug)
                printf("CLEAN UP COMPLETE\n"
        }



        /**
        *Description :  this is an utlity fu
        *
        */
        void generateCommand(int fileCount,c
        {
                struct node *rootNode=FILE_S
                char totalCommand[SIZE],temp
                struct node *temp=rootNode;
                char *finalCommand;
                while(temp!=NULL)
                {
```

Figure 2. Offending pair code matching window. This shows one student who didn't even attempt to subvert a "diff" command! (They seem to not even call our bluff!) As one student who 'lended' the code out, wrote to me in an email – "3874 lines of code [donno what to say ; I coded so well :) || they copied so bad :( ]"

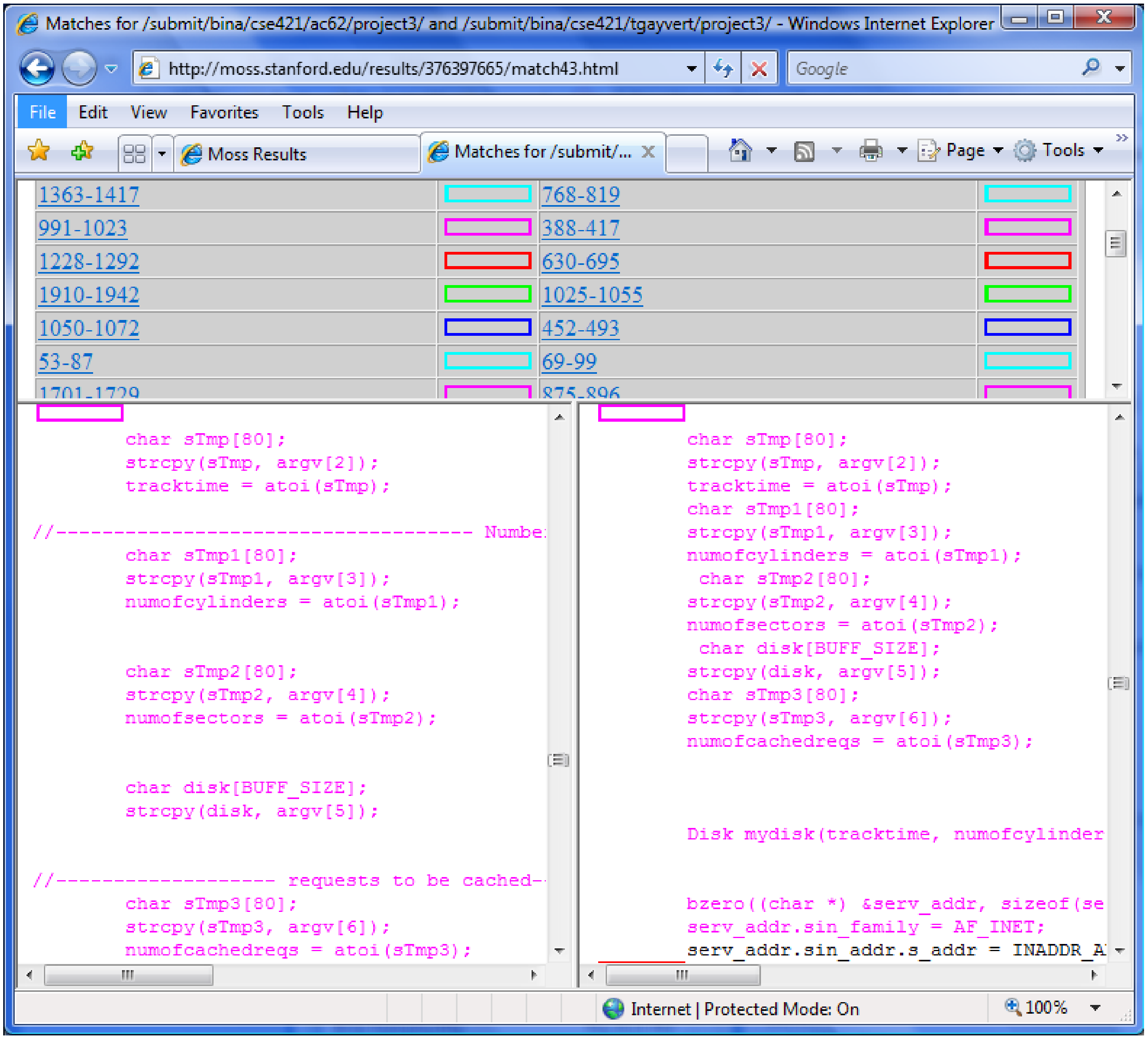

Matches for /submit/bina/cse521/chitteti/project3/ and /submit/bina/cse521/kb84/project3/ - Windows Internet Explorer
http://moss.stanford.edu/results/376397665/match3.html
Google
File Edit View Favorites Tools Help
Moss Results
Matches for /submit/...
Page Tools
Internet | Protected Mode: On
100%

/submit/bina/cse521/[redacted]/project3/ (97%)	🟥	/submit/bina/cse521/[redacted]/project3/ (97%)	🟥
1430-2240	🟥	1447-2249	🟥
2798-3362	🟩	2823-3387	🟩
450-620	🟦	427-596	🟦
2585-2795	🟦	2608-2819	🟦
746-971	🟪	748-996	🟪

```
*Last Edited on: 7 Dec 2008
*/

#include "cse521header.h"

#define MAXSIZE 1024
#define BLOCK 256
#define PORT 55140 //Default port to use
#define PortofBDSserver 55143
#define NameofBDSserver "localhost"
#define DISKSTATUS "DiskState"

//User defined variables
char buff[MAXSIZE];
char StatusofDiskFile[100];
int TotalBlocks;
int BDSClient;
int no_of_cylinders;
int no_of_Sectors;

//Structure for the File Allocation table
struct FAT {
   int IsFull;
   int Cyl_no;
```

```
  */

#include "CSE521.h"

#define MAXLENGTH 1024
#define BLOCKSIZE 256
#define PORT 55053 //Default port to use
#define BDSServerPort 55050
#define BDSServerName "localhost"
#define STATUSofDISK "DiskState"

//User defined variables
char buffer[MAXLENGTH];
char StatusofDiskFile[100];
int TotalBlocks;
int BDSClient;
int NumofCylinders;
int NumofSectors;

//Structure for the File Allocation table
struct FileAllocationTable {
   int IsFull;
   int CylinderNum;
   int SectorNum;
```

Figure 3. Here is a pair of students that actually both copied from ANOTHER student nearly their entire project. They spend several hours searching through the code and renaming each and every variable and structure field. When first questioned, one of the students claimed she didn't cut and paste or use anyone else's code, she just "collaborated" on the computer next to them. If that was true, then we should have them represent UB in the next *Computer Olympics* for *Synchronized Typing*!


Matches for /submit/bina/cse421/ac62/project3/ and /submit/bina/cse421/tgayvert/project3/ - Windows Internet Explorer
http://moss.stanford.edu/results/376397665/match43.html
Google
File
Edit
View
Favorites
Tools
Help
Moss Results
Matches for /submit/...
Page
Tools
Internet | Protected Mode: On
100%

1363-1417	🟦	768-819	🟦
991-1023	🟪	388-417	🟪
1228-1292	🟥	630-695	🟥
1910-1942	🟩	1025-1055	🟩
1050-1072	🟦	452-493	🟦
53-87	🟦	69-99	🟦
1701-1729	🟪	875-896	🟪

```
        char sTmp[80];
        strcpy(sTmp, argv[2]);
        tracktime = atoi(sTmp);

//-------------------------------------- Numbe

        char sTmp1[80];
        strcpy(sTmp1, argv[3]);
        numofcylinders = atoi(sTmp1);


        char sTmp2[80];
        strcpy(sTmp2, argv[4]);
        numofsectors = atoi(sTmp2);


        char disk[BUFF_SIZE];
        strcpy(disk, argv[5]);


//------------------- requests to be cached-
        char sTmp3[80];
        strcpy(sTmp3, argv[6]);
        numofcachedreqs = atoi(sTmp3);
```

```
        char sTmp[80];
        strcpy(sTmp, argv[2]);
        tracktime = atoi(sTmp);
        char sTmp1[80];
        strcpy(sTmp1, argv[3]);
        numofcylinders = atoi(sTmp1);
         char sTmp2[80];
        strcpy(sTmp2, argv[4]);
        numofsectors = atoi(sTmp2);
         char disk[BUFF_SIZE];
        strcpy(disk, argv[5]);
        char sTmp3[80];
        strcpy(sTmp3, argv[6]);
        numofcachedreqs = atoi(sTmp3);



        Disk mydisk(tracktime, numofcylinder


        bzero((char *) &serv_addr, sizeof(se
        serv_addr.sin_family = AF_INET;
        serv_addr.sin_addr.s_addr = INADDR_A
```

Figure 4. This shows a match for a code block that had different formatting, comments inserted, and even new lines of useful code added. I would never have been able to catch something like this; it's like an NP-Complete problem for TAs, code matches might take a long time to find, but it is much quicker to verify. With MOSS we need only verify!

Appendix A – MOSS script

Here is a MOSS script for you to use or peruse, do not put the script or this document in place where students can read it (violates copyright by the author). Instructions for use are contained within the comments of the script or you can consult our case study within the main document. To use, place the “moss” script anywhere and use the following command, carefully replacing the <NAME> parameters with directory names, or extensions, or a wildcard “*”:

perl moss -l c -m 40 -d /submit/<instructorname>/<classname>/*/*.<extension>

For example: perl moss -l c -m 40 -d /submit/bina/*/*/project3/*.c

(NOTE: Some lines wrap in this PDF file, but they would have to be fixed to be contained on the same line)
============================== moss script ==================================

```
#!/usr/bin/perl
#
# Some CSE421/521 Examples
# =====================
# perl moss -l c -m 40 -d /submit/bina/*/*/project3/*.c
# perl moss -l c -m 40 -d /submit/bina/cse521/*/project3/*.c
#
# Useful unzip commands and remove paths and place files in directory:
# ==================================================
# unzip -j nameofzip.zip -d project1
#
# Useful untar commands, untar and remove paths and place files in an already created directory:
# ====================================================================
# mkdir project1
# tar -xvf nameoftar.tar --strip=0 -C project1
#
# For tar.gz, use this instead:
# ===================
# tar -xvfz nameoftar.tar -C project1
#
# Useful renaming commands:
# =====================
# rename .cc .c *.cc                #renames all.cc files in a directory to .c
# rename .cpp .c *.cpp              #renames all .cpp files in a directory to .c
#
# MAIN SCRIPT
# =============================================
# Please read all the comments down to the line that says "STOP".
# These comments are divided into three sections:
#
#    1. usage instructions
#    2. installation instructions
#    3. standard copyright
```

```
#
# Feel free to share this script with other instructors of programming
# classes, but please do not place the script in a publicly accessible
# place.  Comments, questions, and bug reports should be sent to
# moss-request@moss.stanford.edu.
#
# IMPORTANT: This script is known to work on Unix and on Windows using Cygwin.
# It is not known to work on other ways of using Perl under Windows.  If the
# script does not work for you under Windows, you can try the email-based
# version for Windows (available on the Moss home page).
#

#
#  Section 1. Usage instructions
#
#  moss [-l language] [-d] [-b basefile1] ... [-b basefilen] [-m #] [-c "string"] file1 file2 file3 ...
#
# The -l option specifies the source language of the tested programs.
# Moss supports many different languages; see the variable "languages" below for the
# full list.
#
# Example: Compare the lisp programs foo.lisp and bar.lisp:
#
#    moss -l lisp foo.lisp bar.lisp
#
#
# The -d option specifies that submissions are by directory, not by file.
# That is, files in a directory are taken to be part of the same program,
# and reported matches are organized accordingly by directory.
#
# Example: Compare the programs foo and bar, which consist of .c and .h
# files in the directories foo and bar respectively.
#
#    moss -d foo/*.c foo/*.h bar/*.c bar/*.h
#
# Example: Each program consists of the *.c and *.h files in a directory under
# the directory "assignment1."
#
#    moss -d assignment1/*/*.h assignment1/*/*.c
#
#
# The -b option names a "base file".  Moss normally reports all code
# that matches in pairs of files.  When a base file is supplied,
# program code that also appears in the base file is not counted in matches.
# A typical base file will include, for example, the instructor-supplied
# code for an assignment.  Multiple -b options are allowed.  You should
# use a base file if it is convenient; base files improve results, but
# are not usually necessary for obtaining useful information.
#
# IMPORTANT: Unlike previous versions of moss, the -b option *always*
# takes a single filename, even if the -d option is also used.
#
# Examples:
#
#  Submit all of the C++ files in the current directory, using skeleton.cc
#  as the base file:
#
#    moss -l cc -b skeleton.cc *.cc
#
#  Submit all of the ML programs in directories asn1.96/* and asn1.97/*, where
#  asn1.97/instructor/example.ml and asn1.96/instructor/example.ml contain the base files.
#
#    moss -l ml -b asn1.97/instructor/example.ml -b asn1.96/instructor/example.ml -d asn1.97/*/*.ml
asn1.96/*/*.ml
#
# The -m option sets the maximum number of times a given passage may appear
# before it is ignored.  A passage of code that appears in many programs
# is probably legitimate sharing and not the result of plagiarism.  With -m N,
# any passage appearing in more than N programs is treated as if it appeared in
# a base file (i.e., it is never reported).  Option -m can be used to control
# moss' sensitivity.  With -m 2, moss reports only passages that appear
# in exactly two programs.  If one expects many very similar solutions
# (e.g., the short first assignments typical of introductory programming
# courses) then using -m 3 or -m 4 is a good way to eliminate all but
# truly unusual matches between programs while still being able to detect
# 3-way or 4-way plagiarism.  With -m 1000000 (or any very
# large number), moss reports all matches, no matter how often they appear.
# The -m setting is most useful for large assignments where one also a base file
# expected to hold all legitimately shared code.  The default for -m is 10.
#
# Examples:
#
#   moss -l pascal -m 2 *.pascal
#   moss -l cc -m 1000000 -b mycode.cc asn1/*.cc
#
#
# The -c option supplies a comment string that is attached to the generated
# report.  This option facilitates matching queries submitted with replies
# received, especially when several queries are submitted at once.
#
# Example:
#
#   moss -l scheme -c "Scheme programs" *.sch
#
# The -n option determines the number of matching files to show in the results.
# The default is 250.
#
# Example:
#   moss -c java -n 200 *.java
# The -x option sends queries to the current experimental version of the server.
# The experimental server has the most recent Moss features and is also usually
# less stable (read: may have more bugs).
#
# Example:
#
#   moss -x -l ml *.ml
#



#
# Section 2.  Installation instructions.
#
# You may need to change the very first line of this script
# if perl is not in /usr/bin on your system.  Just replace /usr/bin
# with the pathname of the directory where perl resides.
#

#
#  3. Standard Copyright
#
#Copyright (c) 1997 The Regents of the University of California.
#All rights reserved.
#
#Permission to use, copy, modify, and distribute this software for any
#purpose, without fee, and without written agreement is hereby granted,
#provided that the above copyright notice and the following two
#paragraphs appear in all copies of this software.
#
#IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR
#DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT
#OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF
#CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
#
#THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES,
#INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
#AND FITNESS FOR A PARTICULAR PURPOSE.  THE SOFTWARE PROVIDED HEREUNDER IS
#ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATION TO
#PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.
#
#
# STOP.  It should not be necessary to change anything below this line
# to use the script.
#
use IO::Socket;

#
# As of the date this script was written, the following languages were supported.  This script will work with
# languages added later however.  Check the moss website for the full list of supported languages.
#
```

```
@languages = ("c", "cc", "java", "ml", "pascal", "ada", "lisp", "scheme", "haskell", "fortran", "ascii", "vhdl",
"perl", "matlab", "python", "mips", "prolog", "spice", "vb", "csharp", "modula2", "a8086", "javascript", "plsql");

$server = 'moss.stanford.edu';
$port = '7690';
$noreq = "Request not sent.";
$usage = "usage: moss [-x] [-l language] [-d] [-b basefile1] ... [-b basefilen] [-m #] [-c \"string\"] file1 file2 file3
...";

#
# The userid is used to authenticate your queries to the server; don't change it!
#
$userid=194542047;

#
# Process the command line options.  This is done in a non-standard
# way to allow multiple -b's.
#
$opt_l = "c";   # default language is c
$opt_m = 10;
$opt_d = 0;
$opt_x = 0;
$opt_c = "";
$opt_n = 250;
$bindex = 0;    # this becomes non-zero if we have any base files

while (@ARGV && ($_ = $ARGV[0]) =~ /^-(.)(.*)/) {
    ($first,$rest) = ($1,$2);

    shift(@ARGV);
    if ($first eq "d") {
            $opt_d = 1;
            next;
    }
    if ($first eq "b") {
            if($rest eq '') {
                die "No argument for option -b.\n" unless @ARGV;
                $rest = shift(@ARGV);
            }
            $opt_b[$bindex++] = $rest;
            next;
    }
    if ($first eq "l") {
            if ($rest eq '') {
                die "No argument for option -l.\n" unless @ARGV;
                $rest = shift(@ARGV);
            }
            $opt_l = $rest;
            next;
    }
    if ($first eq "m") {
            if($rest eq '') {
                die "No argument for option -m.\n" unless @ARGV;
                $rest = shift(@ARGV);
            }
            $opt_m = $rest;
            next;
    }
    if ($first eq "c") {
            if($rest eq '') {
                die "No argument for option -c.\n" unless @ARGV;
                $rest = shift(@ARGV);
            }
            $opt_c = $rest;
            next;
    }
    if ($first eq "n") {
            if($rest eq '') {
                die "No argument for option -n.\n" unless @ARGV;
                $rest = shift(@ARGV);
            }
            $opt_n = $rest;
            next;
    }
    if ($first eq "x") {
            $opt_x = 1;
            next;
    }
    #
    # Override the name of the server.  This is used for testing this script.
    #
    if ($first eq "s") {
            $server = shift(@ARGV);
            next;
    }
    #
    # Override the port.  This is used for testing this script.
    #
    if ($first eq "p") {
            $port = shift(@ARGV);
            next;
    }
    die "Unrecognized option -$first.  $usage\n";
}

#
# Check a bunch of things first to ensure that the
# script will be able to run to completion.
#
```

```
#
# Make sure all the argument files exist and are readable.
#
print "Checking files . . . \n";
$i = 0;
while($i < $bindex)
{
   die "Base file $opt_b[$i] does not exist. $noreq\n" unless -e "$opt_b[$i]";
   die "Base file $opt_b[$i] is not readable. $noreq\n" unless -r "$opt_b[$i]";
   die "Base file $opt_b is not a text file. $noreq\n" unless -T "$opt_b[$i]";
   $i++;
}
foreach $file (@ARGV)
{
   die "File $file does not exist. $noreq\n" unless -e "$file";
   die "File $file is not readable. $noreq\n" unless -r "$file";
   die "File $file is not a text file. $noreq\n" unless -T "$file";
}

if ("@ARGV" eq '') {
   die "No files submitted.\n $usage";
}
print "OK\n";

#
# Now the real processing begins.
#


$sock = new IO::Socket::INET (
                          PeerAddr => $server,
                          PeerPort => $port,
                          Proto => 'tcp',
                         );
die "Could not connect to server $server: $!\n" unless $sock;
$sock->autoflush(1);

sub read_from_server {
   $msg = <$sock>;
   print $msg;
}

sub upload_file {
   local ($file, $id, $lang) = @_;
#
# The stat function does not seem to give correct filesizes on windows, so
# we compute the size here via brute force.
#
   open(F,$file);
   $size = 0;
```

```
    while (<F>) {
            $size += length($_);
    }
    close(F);

    print "Uploading $file ...";
    print $sock "file $id $lang $size $file\n";
    open(F,$file);
    while (<F>) {
            print $sock $_;
    }
    close(F);
    print "done.\n";
}


print $sock "moss $userid\n";      # authenticate user
print $sock "directory $opt_d\n";
print $sock "X $opt_x\n";
print $sock "maxmatches $opt_m\n";
print $sock "show $opt_n\n";

#
# confirm that we have a supported languages
#
print $sock "language $opt_l\n";
$msg = <$sock>;
chop($msg);
if ($msg eq "no") {
    print $sock "end\n";
    die "Unrecognized language $opt_l.";
}


# upload any base files
$i = 0;
while($i < $bindex) {
    &upload_file($opt_b[$i++],0,$opt_l);
}

$setid = 1;
foreach $file (@ARGV) {
    &upload_file($file,$setid++,$opt_l);
}

print $sock "query 0 $opt_c\n";
print "Query submitted.  Waiting for the server's response.\n";
&read_from_server();
print $sock "end\n";
close($sock);
```