Ben A. Schwennesen | (703) 258-9425 | bas65@duke.edu

Code Samples

JAVA

```
import java.awt.*;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Random;
import java.util.Scanner;
import java.util.TreeMap;
import java.util.TreeSet;
import java.util.concurrent.Semaphore;

import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;

/* PROGRAM TO TEST KNOWLEDGE OF LATIN VOCABULARY AND GRAMMATICAL CONSTRUCTIONS
 * LAST UPDATED: December 2016
 *
 * Owner: Ben Schwennesen (ben.schwennesen@live.com)
 *
 *  */

public class LatinGUI{

        private String in;
        private JFrame frame;
        private String ans;
        private JTextArea dialog;
        private JTextField resp;
        private String q;
```

```
private Boolean waiting;
private Semaphore semaphore;

protected static File latin = new File("data/latin_vocab.txt");
private static HashMap<String, String> terms = new HashMap<String, String>();
private static TreeMap<String, TreeMap<String, String>> map;
private static HashMap<String, String> type = new HashMap<String, String>();
private static HashMap<String, Integer> failMap = new HashMap<String, Integer>();
private static TreeSet<String> fails = new TreeSet<String>();
//private static String[][] specAdjs = new String[3][6];

public Boolean waiting(){
        return waiting;
}

public LatinGUI(){

    frame = new JFrame();
    semaphore = new Semaphore(0);

    frame.setTitle("Latin Dictionary");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    in = new String();
    ans = new String();

    Panel p1 = new Panel();
    p1.add(new Label("Enter translation:"), BorderLayout.WEST);

    resp = new JTextField(30);
    resp.setEditable(true);
    p1.add(resp, BorderLayout.AFTER_LINE_ENDS);

    //dialog = new JLabel();
    dialog = new JTextArea(40,64);
    dialog.setEditable(false);
    JScrollPane pane = new JScrollPane(dialog);
    Panel p2 = new Panel();
    p2.add(pane);
```

```
KeyListener act = new KeyListener(){

            @Override
            public void keyTyped(KeyEvent e) {}
            @Override
            public void keyPressed(KeyEvent e){}

            @Override
            public void keyReleased(KeyEvent e){
                    if(e.getKeyCode() == 10){
                            semaphore.release();
                            in = resp.getText();
                            if(in.length() > 1){
                                    setText();
                                    try {
                                            test(in);
                                    } catch (InterruptedException e1) {
                                            e1.printStackTrace();
                                    }
                            }
                    }
            }

            private void setText() {
                    // TODO Auto-generated method stub
                    dialog.setText(in.toString());
                    resp.setText("");
                    dialog.setText("");
            }



};
resp.addKeyListener(act);

frame.add(p1, BorderLayout.NORTH);
frame.add(p2, BorderLayout.SOUTH);
frame.setSize(800, 720);
frame.setVisible(true);
```

```
  }

public void sendText(String str){
        dialog.setText(str);
}

public void getNext(){
        List<String> keysAsArray = new ArrayList<String>(failMap.keySet());
        Random r = new Random();
        q = keysAsArray.get(r.nextInt(keysAsArray.size()));
        dialog.setText(q);
        ans = terms.get(q);
}

public boolean test(String entered) throws InterruptedException {

        in = "";
        if (ans.contains(entered)) {

                int tries = failMap.get(q);
                failMap.remove(q);

                dialog.setText(print(q) + "\n\n" + "Correct! Only took you "
                                + (tries + 1) + " attempt(s).\n\n" + q + " = " + ans + "\n\n"
                                + "Number of terms remaining: " + failMap.size());

                return true;
        }

        dialog.setText(print(q) + "\n\n" + "You failed!\n\n"+ q +" = "+ ans + "\n\n"
                        + "Number of terms remaining: " + failMap.size());
        fails.add(q);
        int temp = failMap.get(q);
        failMap.put(q, ++temp);
        return false;
}

private static String print(String q) {
        StringBuilder ret = new StringBuilder();
```

```
ret.append(q + "\n");
String categ = type.get(q);
switch (categ) {
        case "ADJ":
                ret.append("\nDECLENSIONS: \n\n");
                ret.append(adjs(q));
                break;
        case "SADJ":
                ret.append("\nDECLENSIONS: \n\n");
                ret.append(specAdjs(q));
                break;
        case "ADV":
                ret.append("\t Adverb");
                break;
        case "CONJ":
                ret.append("\t Conjunction");
                break;
        case "DB":
                ret.append("\t Double Usage");
                break;
        case "F":
                ret.append("\nDECLENSIONS: \n\n");
                ret.append(declensionsF(q));
                break;
        case "IDIOM":
                ret.append("\t Idiom");
                break;
        case "M":
                ret.append("\nDECLENSIONS: \n\n");
                ret.append(declensionsM(q));
                break;
        case "MFI":
                ret.append("\nDECLENSIONS: \n\n");
                ret.append(declensionsMFI(q));
                break;
        case "N":
                ret.append("\nDECLENSIONS: \n\n");
                ret.append(declensionsN(q));
                break;
```

```
                case "NI":
                        ret.append("\nDECLENSIONS: \n\n");
                        ret.append(declensionsNI(q));
                        break;
                case "N*":
                        ret.append("\t Indeclinable Noun");
                        break;
                case "PN":
                        ret.append(pronouns(q));
                        break;
                case "PREP + ABL":
                        ret.append("\t Preposition + Ablative");
                        break;
                case "PREP + ACC":
                        ret.append("\t Preposition + Accusative");
                        break;
                case "SFX":
                        ret.append("\t Suffix");
                        break;
                case "V":
                        ret.append("\nCONJUGATIONS: \n\n");
                        ret.append(verbs(q));
                        break;
        }
        return ret.toString();
}



private static String adjs(String q) {
        // qui? quae? quod?
        // nos / vos / etc. (POSSESSIVES)
        StringBuilder ret = new StringBuilder();
        String[] pParts = q.split(", ");
        String stem;
        if (pParts.length == 3) {
                if (pParts[0].equals("celer") || pParts[0].equals("ācer")
                                || pParts[0].equals("iūcundus")
                                || pParts[0].equals("longus")) {
```

```
        String[][] iMF = declensions.get(5);
        String[][] iN = declensions.get(6);
        stem = pParts[1].substring(0, pParts[1].length() - 2);
        ret.append("\t" + pParts[0] + " || " + pParts[1] + " || " + pParts[2]
                        + "\n");
        for (int c = 1; c < 4; c++) {
                ret.append("\t" + stem + iMF[0][c] + " || " + stem + iN[0][c]
                                + "\n");
        }
        ret.append("\t" + stem + iN[0][4] + " || " + stem + iN[0][4] + "\n");
        ret.append("\t" + stem + iMF[0][5] + " || " + stem + iN[0][5] + "\n\n");
        for (int c = 0; c < 6; c++) {
                ret.append("\t" + stem + iMF[0][c] + " || " + stem + iN[0][c]
                                + "\n");
        }
        return ret.toString();
}

// OF THREE ENDINGS
if(pParts[1].substring(pParts[1].length()-2) == "is"){
        stem = pParts[1].substring(0, pParts[1].length()-2);
        String[][] MFN = declensions.get(9);
        ret.append("\t"+pParts[0]+" || "+stem+MFN[2][0]+" || "+stem+MFN[4][0]+"\n");
        for(int c = 1; c < 6; c++){
                ret.append("\t" + stem + MFN[0][c] + " || " + stem
                                + MFN[2][c] + stem + MFN[4][c] + "\n");
        }
        ret.append("\n");
        for(int c = 0; c < 6; c++){
                ret.append("\t"+stem+MFN[1][c]+" || "+stem+MFN[3][c]
                                + " || " + stem + MFN[5][c]);
        }
}

stem = pParts[1].substring(0, pParts[1].length() - 1);

// regular adjs --> return (declensions EACH GENDER)

String[][] M = declensions.get(1);
```

```
		String[][] F = declensions.get(0);
		String[][] N = declensions.get(2);
		ret.append("\t" + pParts[0] + " || " + pParts[1] + " || " + pParts[2]
				+ "\n");
		for (int c = 1; c < 6; c++) {
			ret.append("\t" + stem + M[0][c] + " || " + stem + F[0][c] + " || "
					+ stem + N[0][c] + "\n");
		}
		ret.append("\n");
		for (int c = 0; c < 6; c++) {
			ret.append("\t" + stem + M[1][c] + " || " + stem + F[1][c] + " || "
					+ stem + N[1][c] + "\n");
		}
		return ret.toString();
	}
	String[][] iMF = declensions.get(5);
	String[][] iN = declensions.get(6);
	stem = pParts[1].substring(0, pParts[1].length() - 1);
	if (pParts.length == 2) {
		if (q.contains("gen.")) {
			stem = stem.substring(7);
			// of one ending
			ret.append("\t" + pParts[0] + " || " + pParts[0] + "\n");
			for (int c = 1; c < 4; c++) {
				ret.append("\t" + stem + iMF[0][c] + " || " + stem + iN[0][c]
						+ "\n");
			}
			ret.append("\t" + stem + iN[0][4] + " || " + stem + iN[0][4] + "\n");
			ret.append("\t" + stem + iMF[0][5] + " || " + stem + iN[0][5] + "\n\n");
			for (int c = 0; c < 6; c++) {
				ret.append("\t" + stem + iMF[1][c] + " || " + stem + iN[1][c]
						+ "\n");
			}
			return ret.toString();
		} else {
			// of two endings
			ret.append("\t" + pParts[0] + " || " + pParts[1] + "\n");
			for (int c = 1; c < 3; c++) {
				ret.append("\t" + stem + iMF[0][c] + " || " + stem + iN[0][c]
```

```
                                        + "\n");
                        }
                        ret.append("\t" + stem + iMF[0][3] + " || " + pParts[1] + "\n");
                        ret.append("\t" + stem + iN[0][4] + " || " + stem + iN[0][4] + "\n");
                        ret.append("\t" + stem + iMF[0][5] + " || " + stem + iN[0][5] + "\n\n");
                        for (int c = 0; c < 6; c++) {
                                ret.append("\t" + stem + iMF[1][c] + " || " + stem + iN[1][c]
                                        + "\n");
                        }
                        return ret.toString();
                }
        }
        return ret.toString();
}

private static String specAdjs(String q) {
        String[][] specAdjs = declensions.get(10);
        StringBuilder ret = new StringBuilder();
        String[] pParts = q.split(",");
        int len = pParts[0].length();
        String stem = pParts[0].substring(0, len - 2);
        for (int c = 0; c < 6; c++) {
                ret.append("\t" + stem + specAdjs[0][c] + " || " + stem + specAdjs[2][c]
                                + " || " + stem + specAdjs[4][c] + "\n");
        }
        ret.append("\n");
        for (int c = 0; c < 6; c++) {
                ret.append("\t" + stem + specAdjs[1][c] + " || " + stem + specAdjs[3][c]
                                + " || " + stem + specAdjs[5][c] + "\n");
        }
        return ret.toString();
}

private static String verbs(String s) {
        StringBuilder ret = new StringBuilder();
        String[] pParts = s.split(", ");
        if (pParts.length != 4) {
                ret.append("\tDEFECTIVE OR ABNORMAL");
                // audeō, audēre, ausum sum
```

```
        // coepī, coepisse, coeptum
        // possum, posse, potuī
        // ōdī, ōdisse, ōsum
        // discō, discere, didicī
        // salveō, salvere
        // videor, vidērī, vīsus sum
        // timeō, timēre, timuī
        return ret.toString();
}
for (String str : pParts) {
        str.trim();
}
String inf = pParts[1];
String uno = pParts[0];
String stem = inf.substring(0, inf.length() - 3);
inf = inf.substring(inf.length() - 3);
String[] act = verbForms.get(0);
boolean someError = true;
ret.append("\n");
ret.append("Active Present/Imperfect/Future \n\n");

switch (inf) {
        case "āre":
                someError = false;
                break;
        case "ēre":
                someError = false;
                act = verbForms.get(1);
                break;
        case "ere":
                someError = false;
                if (uno.charAt(uno.length() - 2) == 'i') {
                        // i-stem
                        act = verbForms.get(3);
                } else {
                        // not i-stem
                        act = verbForms.get(2);
                }
                break;
```

```
        case "īre":
                someError = false;
                act = verbForms.get(4);
                break;
        case "are":
                ret.append("dō || dās || dat || damus || datis || dant \n");
                ret.append("dabam || dabās || dabat ||dabāmus || dabātis || dabant \n");
                ret.append("dabō || dabis || dabit || dabimus || dabitis || dabunt");
                break;
        case "sse":
                ret.append("sum || es || est || sumus || estis || sunt \n");
                ret.append("eram || erās || erat ||erāmus || erātis || erant \n");
                ret.append("erō || eris || erit || erimus || eritis || erunt \n");
                break;
}
if (!someError) {
        for (int i = 0; i < act.length; i++) {
                ret.append(stem + act[i] + " || ");
                /*
                 * if ((i == 2 || i == 5 || i == 8 ||
                                i == 11 || i == 14 || i == 17) && i > 0) {
                        ret.append("\n\n");
                */
                if ((i== 5 || i == 11 || i == 17) && i>0)
                        ret.append("\n");
                //if ((i == 2 || i == 8 || i == 14) && i>0 )
                        //ret.append("\n\t");
                /*
                 * 0 = active present 1st sg.
                 * 1 =   -       -    2nd sg.
                 * 2 =   -       -    3rd sg.
                 * 3 =   -       -    1st pl.
                 * 4 =   -       -    2nd pl.
                 * 5 =   -       -    3rd pl.
                 * 6 = active imperfect 1st sg.
                 * 7 =   -       -      2nd sg.
                 * 8 =   -       -      3rd sg.
                 * 9 =   -       -      1st pl.
                 * 10 =  -       -      2nd pl.
```

```
                * 11 =  -       -      3rd pl.
                * 12 = active future 1st sg.
                * 13 =    -     -    2nd sg.
                * 14 =    -     -    3rd sg.
                * 15 =    -     -    1st pl.
                * 16 =    -     -    2nd pl.
                * 17 =    -     -    3rd pl.
                *
                */

        }
}
ret.append("\n");

String perf = pParts[2];
perf = perf.substring(0, perf.length() - 1);
String[] perfectSys = verbForms.get(5);
ret.append("Active Perfect/Pluperfect/Future perfect \n\n");

for (int i = 0; i < perfectSys.length; i++) {
        ret.append(perf + perfectSys[i] + " || ");
        if ((i== 5 || i == 11 || i == 17) && i>0)
                ret.append("\n");
        //if ((i == 2 || i == 8 || i == 14) && i>0 )
                //ret.append("\n\t");
        /*
         * 0 = active perfect 1st sg.
         * 1 =   -       -    2nd sg.
         * 2 =   -       -    3rd sg.
         * 3 =   -       -    1st pl.
         * 4 =   -       -    2nd pl.
         * 5 =   -       -    3rd pl.
         * 6 = active pluperfect 1st sg.
         * 7 =   -        -     2nd sg.
         * 8 =   -        -     3rd sg.
         * 9 =   -        -     1st pl.
         * 10 =  -        -     2nd pl.
         * 11 =  -        -     3rd pl.
         * 12 = active future perfect 1st sg.
```

```
         * 13 =    -     -       -    2nd sg.
         * 14 =    -     -       -    3rd sg.
         * 15 =    -     -       -    1st pl.
         * 16 =    -     -       -    2nd pl.
         * 17 =    -     -       -    3rd pl.
         *
         */
}
ret.append("\n");

ret.append("Passive Present/Imperfect/Future \n\n");

if (inf.equals("āre")) {
        String[] passPres = verbForms.get(6);
        for (int i = 0; i < passPres.length; i++) {
                ret.append(stem + passPres[i] + " || ");
                if ((i== 5 || i == 11 || i == 17) && i>0)
                        ret.append("\n");
                //if ((i == 2 || i == 8 || i == 14) && i>0 )
                        //ret.append("\n\t");
        }
} else if (inf.equals("ēre")) {
        String[] passPres = verbForms.get(7);
        for (int i = 0; i < passPres.length; i++) {
                ret.append(stem + passPres[i] + " || ");
                if ((i== 5 || i == 11 || i == 17) && i>0)
                        ret.append("\n");
                //if ((i == 2 || i == 8 || i == 14) && i>0 )
                        //ret.append("\n\t");
        }
        /*
         * 0 = passive present 1st sg.
         * 1 =    -       -    2nd sg.
         * 2 =    -       -    3rd sg.
         * 3 =    -       -    1st pl.
         * 4 =    -       -    2nd pl.
         * 5 =    -       -    3rd pl.
         * 6 = passive imperfect 1st sg.
         * 7 =    -       -      2nd sg.
```

```
         * 8 =    -       -      3rd sg.
         * 9 =    -       -      1st pl.
         * 10 =   -       -      2nd pl.
         * 11 =   -       -      3rd pl.
         * 12 = passive future 1st sg.
         * 13 =    -      -     2nd sg.
         * 14 =    -      -     3rd sg.
         * 15 =    -      -     1st pl.
         * 16 =    -      -     2nd pl.
         * 17 =    -      -     3rd pl.
         *
         */
}

ret.append("\n");
ret.append("Passive Perfect/Pluperfect/Future perfect \n\n");

stem = pParts[3];
stem = stem.substring(0, stem.length() - 2);
String ppStem = pParts[3];
ppStem = ppStem.substring(0, ppStem.length() - 2);
String[] passPerf = verbForms.get(8);

for (int i = 0; i < passPerf.length; i++) {
        ret.append(stem + passPerf[i] + " || ");
        if ((i== 5 || i == 11 || i == 17) && i>0)
                ret.append("\n");
        if ((i == 2 || i == 8 || i == 14) && i>0 )
                ret.append("\n\t");
        /*
         * 0 = passive perfect 1st sg.
         * 1 =   -       -    2nd sg.
         * 2 =   -       -    3rd sg.
         * 3 =   -       -    1st pl.
         * 4 =   -       -    2nd pl.
         * 5 =   -       -    3rd pl.
         * 6 = passive pluperfect 1st sg.
         * 7 =   -        -     2nd sg.
         * 8 =   -        -     3rd sg.
```

```
                * 9 =   -        -      1st pl.
                * 10 =  -        -      2nd pl.
                * 11 =  -        -      3rd pl.
                * 12 = passive future perfect 1st sg.
                * 13 =    -     -       -    2nd sg.
                * 14 =    -     -       -    3rd sg.
                * 15 =    -     -       -    1st pl.
                * 16 =    -     -       -    2nd pl.
                * 17 =    -     -       -    3rd pl.
                *
                */
        }

        return ret.toString();
}

private static String pronouns(String s) {
        StringBuilder str = new StringBuilder();
        switch (s) {
                // hic, haec, hoc
                case "hic, haec, hoc":
                        str.append("\thic || haec || hoc\n");
                        str.append("\t -- huius -- \n");
                        str.append("\t --  huic -- \n");
                        str.append("\thunc || hanc || hoc\n");
                        str.append("\thōc || hāc || hōc\n\n");

                        str.append("\thī || hae || haec\n");
                        str.append("\thōrum || hārum || hōrum\n");
                        str.append("\t -- hīs -- \n");
                        str.append("\thōs || hās || haec\n");
                        str.append("\t -- hīs -- \n");
                        break;

                        // ille, illa, illud & iste, ista, istud
                        // & ipse, ipsa, ipsum
                case "iste, ista, istud":
                        str.append("\tiste || ista || istud\n");
                        str.append("\t -- istīus -- \n");
```

```
        str.append("\t --  istī -- \n");
        str.append("\tistum || istam || istum\n");
        str.append("\tistō || istā || istō\n\n");
                        // Vocative not used
        str.append("\tistī || istae || iste\n");
        str.append("\tistōrum || istārum || istōrum\n");
        str.append("\t -- istīs -- \n");
        str.append("\tistōs || istās || ista\n");
        str.append("\t -- istīs -- \n");
                        // Vocative not used
        break;

case "ille, illa, illud":
        str.append("\tille || illa || illud\n");
        str.append("\t -- illīus -- \n");
        str.append("\t --  illī -- \n");
        str.append("\tillum || illam || illum\n");
        str.append("\tillō || illā || illō\n\n");

        str.append("\tillī || illae || illa\n");
        str.append("\tillōrum || illārum || illōrum\n");
        str.append("\t -- illīs -- \n");
        str.append("\tillōs || illās || illa\n");
        str.append("\t -- illīs -- \n");
        break;

case "ipse, ipsa, ipsum":
        str.append("\tipse || ipsa || ipsud\n");
        str.append("\t -- ipsīus -- \n");
        str.append("\t --  ipsī -- \n");
        str.append("\tipsum || ipsam || ipsum\n");
        str.append("\tipsō || ipsā || ipsō\n\n");

        str.append("\tipsī || ipsae || ipsa\n");
        str.append("\tipsōrum || ipsārum || ipsōrum\n");
        str.append("\t -- ipsīs -- \n");
        str.append("\tipsōs || ipsās || ipsa\n");
        str.append("\t -- ipsīs -- \n");
        break;
```

```
// ego, mei
case "ego, meī":
        str.append("\t (ego) \n");
        str.append("\t meī \n");
        str.append("\t mihi \n");
        str.append("\t mē \n");
        str.append("\t mē \n\n");

        str.append("\t (nōs) \n");
        str.append("\t nostrum/nostrī \n");
        str.append("\t nōbīs \n");
        str.append("\t nōs \n");
        str.append("\t nōbīs \n");
        str.append("\t *** Parenthesis: if _reflexive_, no nominative case");
        break;

// tu, tui
case "tū, tuī":
        str.append("\t tū \n");
        str.append("\t tuī \n");
        str.append("\t tibi \n");
        str.append("\t tē \n");
        str.append("\t tē \n\n");

        str.append("\t vōs \n");
        str.append("\t vestrum/vestrī \n");
        str.append("\t vōbīs \n");
        str.append("\t vōs \n");
        str.append("\t vōbīs \n");
        break;

// sui
case "suī":
        str.append("\t -- \n");
        str.append("\tsuī\n");
        str.append("\tsibi\n");
        str.append("\tsē\n");
        str.append("\tsē\n\n");
```

```
        str.append("\t -- \n");
        str.append("\tsuī\n");
        str.append("\tsibi\n");
        str.append("\tsē\n");
        str.append("\tsē");
        break;

// is, ea, id
case "is, ea, id":
        str.append("\tis || ea || id\n");
        str.append("\t -- eius -- \n");
        str.append("\t --  eī -- \n");
        str.append("\teum || eam || id\n");
        str.append("\teō || eā || eō\n\n");

        str.append("\teī/iī || eae || ea\n");
        str.append("\teōrum || eārum || eōrum\n");
        str.append("\t -- eīs -- \n");
        str.append("\teōs || eās || ea\n");
        str.append("\t -- eīs -- \n");
        break;

        // idem, eadem, idem
case "īdem, eadem, idem":
        str.append("\tīdem || eadem || idem\n");
        str.append("\t -- eiusdem -- \n");
        str.append("\t --  eīdem -- \n");
        str.append("\teundem || eandem || idem\n");
        str.append("\teōdem || eādem || eōdem\n\n");

        str.append("\teīdem || eaedem || eadem\n");
        str.append("\teōrundem || eārundem || eōrundem\n");
        str.append("\t -- eīsdem -- \n");
        str.append("\teōsdem || eāsdem || eadem\n");
        str.append("\t -- eīsdem -- ");
        break;

        // quid (just quid)
```

```
case "quid":
        str.append("\n");
        break;

        // quis?, quid?
case "quis?, quid?":
        str.append("\tquis || quid\n");
        str.append("\t-- cuius --\n");
        str.append("\t -- cui -- \n");
        str.append("\tquem || quid\n");
        str.append("\t -- quō -- \n\n");

        str.append("\tquī || quae || quae\n");
        str.append("\tquōrum || quārum || quōrum\n");
        str.append("\t -- quibus -- \n");
        str.append("\tquōs || quās || quōs\n");
        str.append("\t -- quibus -- \n");
        break;

        // quisque, quidque
case "quisque, quidque (gen. cuiusque; dat. cuique)":
        str.append("\tquisque || quidque\n");
        str.append("\t-- cuiusque --\n");
        str.append("\t -- cuique -- \n");
        str.append("\tquemque || quidque\n");
        str.append("\t -- quōque -- \n\n");

        str.append("\tquīque || quaeque || quaeque\n");
        str.append("\tquōrumque || quārumque || quōrumque\n");
        str.append("\t -- quibusque -- \n");
        str.append("\tquōsque || quāsque || quōsque\n");
        str.append("\t -- quibusque -- \n");
        break;

        // qui, quae, quod
case "quī, quae, quod":
        str.append("\tquī || quae || quod\n");
        str.append("\t -- cuius -- \n");
        str.append("\t  -- cui --  \n");
```

```
                        str.append("\tquem || quam || quod\n");
                        str.append("\tquō  || quā  ||  quō\n\n");

                        str.append("\tquī || quae || quae\n");
                        str.append("\tquōrum || quārum || quōrum\n");
                        str.append("\t -- quibus -- \n");
                        str.append("\tquōs || quās || quōs\n");
                        str.append("\t -- quibus -- \n");
                        break;
                }

                return str.toString();
        }

        private static String declensionsM(String s) {
                StringBuilder str = new StringBuilder();
                String[] entries = s.split(", ");
                String stem;
                int len = entries[1].length();
                char genEnd = entries[1].charAt(len - 1);
                switch (genEnd) {
                        case 'e': // PAINS 1st
                                stem = entries[1].substring(0, entries[1].length() - 2);
                                String[][] ends = declensions.get(0);
                                /*
                                   for (int c = 0; c < 5; c++) {
                                          str.append("\t" + stem + ends[0][c] + " || " + stem + ends[1][c]
                                                        + "\n");
                                       str.append("\t"+stem+ends[0][c]);
                                }
                                */
                                for(int c = 0; c < 6; c++) str.append("\t"+stem+ends[0][c]+"\n");
                                str.append("\n");
                                for(int c = 0; c < 6; c++) str.append("\t"+stem+ends[1][c]+"\n");
                                break;
                        case 'ī': // 2nd declension
                                // -us vs. -er
                                stem = entries[1].substring(0, entries[1].length() - 1);
                                int l = entries[0].length();
```

```
		if (entries[0].charAt(l - 1) == 's') {
			// -us
			String[][] ends2 = declensions.get(1);
			/*
			for (int c = 0; c < 5; c++) {
				str.append("\t" + stem + ends2[0][c] + " || " + stem
						+ ends2[1][c] + "\n");
			}
			*/
			for(int c = 0; c < 6; c++) str.append("\t"+stem+ends2[0][c]+"\n");
			str.append("\n");
			for(int c = 0; c < 6; c++) str.append("\t"+stem+ends2[1][c]+"\n");
		} else {
			// -er
			String[][] ends3 = declensions.get(1);
			str.append("\t" + entries[0] + " || " + stem + ends3[1][0] + "\n");
			/*
			for (int c = 1; c < 5; c++) {
				str.append("\t" + stem + ends3[0][c] + " || " + stem
						+ ends3[1][c] + "\n");
			}
			*/
			str.append("\t"+entries[0]+"\n");
			for(int c = 1; c < 5; c++) str.append("\t"+stem+ends3[0][c]+"\n");
			str.append("\t"+entries[0]+"\n\n");
			for(int c = 0; c < 6; c++) str.append("\t"+stem+ends3[1][c]+"\n");
		}
		break;
	case 's':
		stem = entries[1].substring(0, entries[1].length() - 2);
		if (entries[1].charAt(len - 2) == 'i') {
			// 3rd declension
			String[][] ends4 = declensions.get(3);
			/*
			str.append("\t" + entries[0] + " || " + stem + ends4[1][0] + "\n");
			for (int c = 1; c < 5; c++) {
				str.append("\t" + stem + ends4[0][c] + " || " + stem
						+ ends4[1][c] + "\n");
			}
```

```
                        */

                        str.append("\t"+entries[0]+"\n");
                        for(int c = 1; c < 6; c++) str.append("\t"+stem+ends4[0][c]+"\n");
                        str.append("\n");
                        for(int c = 0; c < 6; c++) str.append("\t"+stem+ends4[1][c]+"\n");

                } else if (entries[1].charAt(len - 2) == 'ū') {
                        // 4th declension
                        String[][] ends4 = declensions.get(7);
                        /*
                        for (int c = 0; c < 5; c++) {
                                str.append("\t" + stem + ends4[0][c] + " || " + stem
                                                + ends4[1][c] + "\n");
                        }
                        */

                        for(int c = 0; c < 6; c++) str.append("\t"+stem+ends4[0][c]+"\n");
                        str.append("\n");
                        for(int c = 0; c < 6; c++) str.append("\t"+stem+ends4[1][c]+"\n");

                } else {
                        str.append("HMMMMMMM");
                        // vīs, vīs
                        // nēmō, nūllīus, nēminī, nēminem, nūllā
                        // loca
                }
                break;
        }
        return str.toString();
}

private static String declensionsF(String s) {
        StringBuilder str = new StringBuilder();
        String[] entries = s.split(", ");
        String stem;
        int len = entries[1].length();
        char genEnd = entries[1].charAt(len - 1);
        switch (genEnd) {
```

```
case 'e': // -ae
        stem = entries[1].substring(0, entries[1].length() - 2);
        String[][] ends = declensions.get(0);
        /*
        for (int c = 0; c < 5; c++) {
                str.append("\t" + stem + ends[0][c] + " || " + stem + ends[1][c]
                                + "\n");
        }
        */
        for(int c = 0; c < 6; c++) str.append("\t"+stem+ends[0][c]+"\n");
        str.append("\n");
        for(int c = 0; c < 6; c++) str.append("\t"+stem+ends[1][c]+"\n");
        break;
case 's':
        stem = entries[1].substring(0, entries[1].length() - 2);
        if (entries[1].charAt(len - 2) == 'i') {
                // 3rd declension
                String[][] ends4 = declensions.get(3);
                /*
                for (int c = 1; c < 5; c++) {
                        str.append("\t" + stem + ends4[0][c] + " || " + stem
                                        + ends4[1][c] + "\n");
                }
                */

                str.append("\t"+entries[0]+"\n");
                for(int c = 1; c < 6; c++) str.append("\t"+stem+ends4[0][c]+"\n");
                str.append("\n");
                for(int c = 0; c < 6; c++) str.append("\t"+stem+ends4[1][c]+"\n");

        } else if (entries[1].charAt(len - 2) == 'ū') {
                // 4th declension
                String[][] ends4 = declensions.get(7);
                /*
                for (int c = 0; c < 5; c++) {
                        str.append("\t" + stem + ends4[0][c] + " || " + stem
                                        + ends4[1][c] + "\n");
                }
                */
```

```
                        for(int c = 0; c < 6; c++) str.append("\t"+stem+ends4[0][c]+"\n");
                        str.append("\n");
                        for(int c = 0; c < 6; c++) str.append("\t"+stem+ends4[1][c]+"\n");

                } else {
                        str.append("HMMMMMMM");
                }
                break;
        }
        return str.toString();
}

private static String declensionsN(String s) {
        StringBuilder str = new StringBuilder();
        String[] entries = s.split(", ");
        String stem;
        int len = entries[1].length();
        char genEnd = entries[1].charAt(len - 1);
        switch (genEnd) {
                case 'ī': // 2nd declension
                        stem = entries[1].substring(0, len - 1);
                        String[][] ends = declensions.get(2);
                        for (int c = 0; c < 6; c++) {
                                //str.append("\t" + stem + ends[0][c] + " || " + stem + ends[1][c]
                                                //+ "\n");
                                str.append("\t" + stem + ends[0][c] + "\n");
                        }
                        str.append("\n");
                        for (int c = 0; c < 6; c++){
                                str.append("\t" + stem + ends[1][c] + "\n");
                        }
                        break;
                case 's':
                        stem = entries[1].substring(0, len - 2);
                        if (entries[1].charAt(len - 2) == 'i') {
                                // 3rd declension
                                String[][] ends4 = declensions.get(4);
                                str.append("\t" + entries[0] + "\n");
```

```
                        //str.append("\t"+ entries[0] +"\n");
                        for(int c = 1; c < 5; c++){
                                if(c == 3) str.append("\t"+ entries[0] +"\n");
                                else str.append("\t"+stem+ends4[0][c]+"\n");
                        }
                        str.append("\t"+entries[0]+"\n\n");
                        for(int c = 0; c < 6; c++) str.append("\t"+stem+ends4[1][c]+"\n");
                } else if (entries[1].charAt(len - 2) == 'ū') {
                        // 4th declension
                        String[][] ends4 = declensions.get(7);
                        /*
                        for (int c = 0; c < 5; c++) {
                                str.append("\t" + stem + ends4[0][c] + " || " + stem
                                                + ends4[1][c] + "\n");
                        }
                        */
                        for(int c = 0; c < 6; c++) str.append("\t"+stem+ends4[0][c]+"\n");
                        str.append("\n");
                        for(int c = 0; c < 6; c++) str.append("\t"+stem+ends4[1][c]+"\n");
                } else {
                        str.append("HMMMMMMM");
                }
                break;
        }
        return str.toString();
}

private static String declensionsMFI(String q) {
        StringBuilder str = new StringBuilder();
        String[] entries = q.split(", ");
        String stem;
        stem = entries[1].substring(0, entries[1].length() - 2);
        String[][] ends = declensions.get(5);
        /*
        str.append("\t" + entries[0] + " || " + stem + ends[1][0] + "\n");
        for (int c = 1; c < 5; c++) {
                str.append("\t" + stem + ends[0][c] + " || " + stem + ends[1][c] + "\n");
        }
        */
```

```
        str.append("\t"+entries[0]+"\n");
        for(int c = 1; c < 6; c++) str.append("\t"+stem+ends[0][c]+"\n");
        str.append("\n");
        for(int c = 0; c < 6; c++) str.append("\t"+stem+ends[1][c]+"\n");

        return str.toString();
}

private static String declensionsNI(String q) {
        StringBuilder str = new StringBuilder();
        String[] entries = q.split(", ");
        String stem;
        stem = entries[1].substring(0, entries[1].length() - 2);
        String[][] ends = declensions.get(6);
        /*
        for (int c = 1; c < 5; c++) {
                if (c == 3) {
                        str.append("\t" + entries[0] + " || " + stem + ends[1][3] + "\n");
                } else {
                        str.append("\t" + stem + ends[0][c] + " || " + stem + ends[1][c]
                                        + "\n");
                }
        }
        */

        str.append("\t"+entries[0]+"\n");
        for(int c = 1; c < 6; c++){
                if(c == 3) str.append("\t"+entries[0]+"\n");
                else str.append("\t"+stem+ends[0][c]+"\n");
        }
        str.append("\n");
        for(int c = 0; c < 6; c++) str.append("\t"+stem+ends[1][c]+"\n");

        return str.toString();
}

public static TreeMap<String, TreeMap<String, String>> makeMap(Scanner in) {
        makeVerbs();
```

```
        makeDeclensions();
        map = new TreeMap<String, TreeMap<String, String>>();
        while (in.hasNextLine()) {
                String temp = in.nextLine();
                String[] comps = temp.split(" : ");
                comps[0].trim();
                comps[1].trim();
                comps[2].trim();
                if (comps.length < 2)
                        break;
                if (!map.containsKey(comps[0])) {
                        map.put(comps[0], new TreeMap<String, String>());
                }
                type.put(comps[1], comps[0]);
                TreeMap<String, String> theMap = map.get(comps[0]);
                theMap.put(comps[1], comps[2]);
                terms.put(comps[1], comps[2]);
                failMap.put(comps[1], 0);
        }
        return map;
}

private static List<String[][]> declensions = new ArrayList<String[][]>();

private static void makeDeclensions() {
        String[][] dec1 = new String[2][6];
        dec1[0][0] = "a";
        dec1[0][1] = "ae";
        dec1[0][2] = "ae";
        dec1[0][3] = "am";
        dec1[0][4] = "ā";
        dec1[0][5] = "a";
        dec1[1][0] = "ae";
        dec1[1][1] = "ārum";
        dec1[1][2] = "īs";
        dec1[1][3] = "ās";
        dec1[1][4] = "īs";
        dec1[1][5] = "ae";
        declensions.add(dec1); // 0
```

```
String[][] dec2M = new String[2][7]; // SPECIAL
dec2M[0][0] = "us";
dec2M[0][1] = "ī";
dec2M[0][2] = "ō";
dec2M[0][3] = "um";
dec2M[0][4] = "ō";
dec2M[0][5] = "e";
dec2M[0][6] = "e"; // SPECIAL
dec2M[1][0] = "ī";
dec2M[1][1] = "ōrum";
dec2M[1][2] = "īs";
dec2M[1][3] = "ōs";
dec2M[1][4] = "īs";
dec2M[1][5] = "ī";
declensions.add(dec2M); // 1

String[][] dec2N = new String[2][6];
dec2N[0][0] = "um";
dec2N[0][1] = "ī";
dec2N[0][2] = "ō";
dec2N[0][3] = "um";
dec2N[0][4] = "ō";
dec2N[0][5] = "um";
dec2N[1][0] = "a";
dec2N[1][1] = "ōrum";
dec2N[1][2] = "īs";
dec2N[1][3] = "a";
dec2N[1][4] = "īs";
dec2N[1][5] = "a";
declensions.add(dec2N); // 2

String[][] dec3MF = new String[2][6];
dec3MF[0][0] = "";
dec3MF[0][1] = "is";
dec3MF[0][2] = "ī";
dec3MF[0][3] = "em";
dec3MF[0][4] = "e";
dec3MF[0][5] = ""; // SAME AS NOMINATIVE
```

```
dec3MF[1][0] = "ēs";
dec3MF[1][1] = "um";
dec3MF[1][2] = "ibus";
dec3MF[1][3] = "ēs";
dec3MF[1][4] = "ibus";
dec3MF[1][5] = "ēs";
declensions.add(dec3MF); // 3

String[][] dec3N = new String[2][6];
dec3N[0][0] = "";
dec3N[0][1] = "is";
dec3N[0][2] = "ī";
dec3N[0][3] = "";
dec3N[0][4] = "e";
dec3N[0][5] = "";
dec3N[1][0] = "a";
dec3N[1][1] = "um";
dec3N[1][2] = "ibus";
dec3N[1][3] = "a";
dec3N[1][4] = "ibus";
dec3N[1][5] = "a";
declensions.add(dec3N); // 4

String[][] dec3MFI = new String[2][6];
dec3MFI[0][0] = "is";
dec3MFI[0][1] = "is";
dec3MFI[0][2] = "ī";
dec3MFI[0][3] = "em";
dec3MFI[0][4] = "e";
dec3MFI[0][5] = "is";
dec3MFI[1][0] = "ēs";
dec3MFI[1][1] = "ium";
dec3MFI[1][2] = "ibus";
dec3MFI[1][3] = "ēs";
dec3MFI[1][4] = "ibus";
dec3MFI[1][5] = "ēs";
declensions.add(dec3MFI); // 5

String[][] dec3NI = new String[2][6];
```

```
dec3NI[0][0] = "e";
dec3NI[0][1] = "is";
dec3NI[0][2] = "ī";
dec3NI[0][3] = "e";
dec3NI[0][4] = "ī";
dec3NI[0][5] = "e";
dec3NI[1][0] = "ia";
dec3NI[1][1] = "ium";
dec3NI[1][2] = "ibus";
dec3NI[1][3] = "ia";
dec3NI[1][4] = "ibus";
dec3NI[1][5] = "ia";
declensions.add(dec3NI); // 6

String[][] dec4MF = new String[2][6];
dec4MF[0][0] = "us";
dec4MF[0][1] = "ūs";
dec4MF[0][2] = "uī";
dec4MF[0][3] = "um";
dec4MF[0][4] = "ū";
dec4MF[0][5] = "us";
dec4MF[1][0] = "ūs";
dec4MF[1][1] = "uum";
dec4MF[1][2] = "ibus";
dec4MF[1][3] = "ūs";
dec4MF[1][4] = "ibus";
dec4MF[1][5] = "ūs";
declensions.add(dec4MF); // 7

String[][] dec4N = new String[2][6];
dec4N[0][0] = "ū";
dec4N[0][1] = "ūs";
dec4N[0][2] = "ū";
dec4N[0][3] = "ū";
dec4N[0][4] = "ū";
dec4N[0][5] = "ū";
dec4N[1][0] = "ua";
dec4N[1][1] = "uum";
dec4N[1][2] = "ibus";
```

```
dec4N[1][3] = "ua";

dec4N[1][4] = "ibus";

dec4N[1][5] = "ua";

declensions.add(dec4N); // 8


String[][] dec3END = new String[6][6];

// MASC

dec3END[0][0] = ""; // SPECIAL

dec3END[0][1] = "is";

dec3END[0][2] = "ī";

dec3END[0][3] = "em";

dec3END[0][4] = "ī";

dec3END[0][5] = "er";

dec3END[1][0] = "ēs";

dec3END[1][1] = "ium";

dec3END[1][2] = "ibus";

dec3END[1][3] = "ēs";

dec3END[1][4] = "ibus";

dec3END[1][5] = "ēs";

// FEM

dec3END[2][0] = "er";

dec3END[2][1] = "is";

dec3END[2][2] = "ī";

dec3END[2][3] = "em";

dec3END[2][4] = "ī";

dec3END[2][5] = "er";

dec3END[3][0] = "ēs";

dec3END[3][1] = "ium";

dec3END[3][2] = "ibus";

dec3END[3][3] = "ēs";

dec3END[3][4] = "ibus";

dec3END[3][5] = "ēs";

// NEUT

dec3END[4][0] = "e";

dec3END[4][1] = "is";

dec3END[4][2] = "ī";

dec3END[4][3] = "e";

dec3END[4][4] = "ī";

dec3END[4][5] = "e";
```

```
dec3END[5][0] = "ia";
dec3END[5][1] = "ium";
dec3END[5][2] = "ibus";
dec3END[5][3] = "ia";
dec3END[5][4] = "ibus";
dec3END[5][5] = "ia";
declensions.add(dec3END); // 9

String[][] specAdjs = new String[6][6];
specAdjs[0][0] = "us";
specAdjs[0][1] = "īus";
specAdjs[0][2] = "ī";
specAdjs[0][3] = "um";
specAdjs[0][4] = "ō";
specAdjs[0][5] = "us";
specAdjs[1][0] = "ī";
specAdjs[1][1] = "ōrum";
specAdjs[1][2] = "īs";
specAdjs[1][3] = "ōs";
specAdjs[1][4] = "īs";
specAdjs[1][5] = "ī";

specAdjs[2][0] = "a";
specAdjs[2][1] = "īus";
specAdjs[2][2] = "ī";
specAdjs[2][3] = "am";
specAdjs[2][4] = "ā";
specAdjs[2][5] = "a";
specAdjs[3][0] = "ae";
specAdjs[3][1] = "ārum";
specAdjs[3][2] = "īs";
specAdjs[3][3] = "ās";
specAdjs[3][4] = "īs";
specAdjs[3][5] = "ae";


specAdjs[4][0] = "um";
specAdjs[4][1] = "īus";
specAdjs[4][2] = "ī";
```

```
        specAdjs[4][3] = "um";
        specAdjs[4][4] = "ō";
        specAdjs[4][5] = "um";
        specAdjs[5][0] = "a";
        specAdjs[5][1] = "ōrum";
        specAdjs[5][2] = "īs";
        specAdjs[5][3] = "a";
        specAdjs[5][4] = "īs";
        specAdjs[5][5] = "a";
        declensions.add(specAdjs); // 10
}

private static List<String[]> verbForms = new ArrayList<String[]>();

// List stores the types of conjugations
// 0-4 : 1st, 2nd, 3rd, 3rd i-stem, 4th
// 5 : Active Perfect
// 6 : Passive Present
// 7 : Passive Perfect

private static void makeVerbs() {
        String[] first = new String[18];
        first[0] = "ō";
        first[1] = "ās";
        first[2] = "at";
        first[3] = "āmus";
        first[4] = "ātis";
        first[5] = "ant";
        first[6] = "ābam";
        first[7] = "ābās";
        first[8] = "ābat";
        first[9] = "ābāmus";
        first[10] = "ābātis";
        first[11] = "ābant";
        first[12] = "ābō";
        first[13] = "ābis";
        first[14] = "ābit";
        first[15] = "ābimus";
        first[16] = "ābitis";
```

```
first[17] = "ābunt";
verbForms.add(first);
String[] second = new String[18];
second[0] = "eō";
second[1] = "ēs";
second[2] = "et";
second[3] = "ēmus";
second[4] = "ētis";
second[5] = "ent";
second[6] = "ēbam";
second[7] = "ēbās";
second[8] = "ēbat";
second[9] = "ēbāmus";
second[10] = "ēbātis";
second[11] = "ēbant";
second[12] = "ēbō";
second[13] = "ēbis";
second[14] = "ēbit";
second[15] = "ēbimus";
second[16] = "ēbitis";
second[17] = "ēbunt";
verbForms.add(second);
String[] third = new String[18];
third[0] = "ō";
third[1] = "is";
third[2] = "it";
third[3] = "imus";
third[4] = "itis";
third[5] = "unt";
third[6] = "ēbam";
third[7] = "ēbās";
third[8] = "ēbat";
third[9] = "ēbāmus";
third[10] = "ēbātis";
third[11] = "ēbant";
third[12] = "am";
third[13] = "ēs";
third[14] = "et";
third[15] = "ēmus";
```

```
third[16] = "ētis";

third[17] = "ent";

verbForms.add(third);

String[] third_i = new String[18];

third_i[0] = "iō";

third_i[1] = "is";

third_i[2] = "it";

third_i[3] = "imus";

third_i[4] = "itis";

third_i[5] = "iunt";

third_i[6] = "iēbam";

third_i[7] = "iēbās";

third_i[8] = "iēbat";

third_i[9] = "iēbāmus";

third_i[10] = "iēbātis";

third_i[11] = "iēbant";

third_i[12] = "iam";

third_i[13] = "iēs";

third_i[14] = "iet";

third_i[15] = "iēmus";

third_i[16] = "iētis";

third_i[17] = "ient";

verbForms.add(third_i);

String[] fourth = new String[18];

fourth[0] = "iō";

fourth[1] = "īs";

fourth[2] = "it";

fourth[3] = "īmus";

fourth[4] = "ītis";

fourth[5] = "iunt";

fourth[6] = "iēbam";

fourth[7] = "iēbās";

fourth[8] = "iēbat";

fourth[9] = "iēbāmus";

fourth[10] = "iēbātis";

fourth[11] = "iēbant";

fourth[12] = "iam";

fourth[13] = "iēs";

fourth[14] = "iet";
```

```
fourth[15] = "iēmus";

fourth[16] = "iētis";

fourth[17] = "ient";

verbForms.add(fourth);

// 5 : Active Perfect

String[] actPerf = new String[18];

actPerf[0] = "ī";

actPerf[1] = "istī";

actPerf[2] = "it";

actPerf[3] = "imus";

actPerf[4] = "istis";

actPerf[5] = "ērunt";

actPerf[6] = "eram";

actPerf[7] = "erās";

actPerf[8] = "erat";

actPerf[9] = "erāmus";

actPerf[10] = "erātis";

actPerf[11] = "erant";

actPerf[12] = "erō";

actPerf[13] = "eris";

actPerf[14] = "erit";

actPerf[15] = "erimus";

actPerf[16] = "eritis";

actPerf[17] = "erint";

verbForms.add(actPerf);

// 6 : Passive Present 1st

String[] pasPres1 = new String[18];

pasPres1[0] = "or";

pasPres1[1] = "āris";

pasPres1[2] = "ātur";

pasPres1[3] = "āmur";

pasPres1[4] = "āminī";

pasPres1[5] = "antur";

pasPres1[6] = "ābar";

pasPres1[7] = "ābāris";

pasPres1[8] = "ābātur";

pasPres1[9] = "ābāmur";

pasPres1[10] = "ābāminī";

pasPres1[11] = "ābantur";
```

```
pasPres1[12] = "ābor";
pasPres1[13] = "āberis";
pasPres1[14] = "ābitur";
pasPres1[15] = "ābimur";
pasPres1[16] = "ābiminī";
pasPres1[17] = "ābuntur";
verbForms.add(pasPres1);
// 7 : Passive Present 2nd
String[] pasPres2 = new String[18];
pasPres2[0] = "eor";
pasPres2[1] = "ēris";
pasPres2[2] = "ētur";
pasPres2[3] = "ēmur";
pasPres2[4] = "ēminī";
pasPres2[5] = "entur";
pasPres2[6] = "ēbar";
pasPres2[7] = "ēbāris";
pasPres2[8] = "ēbātur";
pasPres2[9] = "ēbāmur";
pasPres2[10] = "ēbāminī";
pasPres2[11] = "ēbantur";
pasPres2[12] = "ēbor";
pasPres2[13] = "ēberis";
pasPres2[14] = "ēbitur";
pasPres2[15] = "ēbimur";
pasPres2[16] = "ēbiminī";
pasPres2[17] = "ēbuntur";
verbForms.add(pasPres2);
// 8 : Passive Perfect
String[] pasPerf = new String[18];
pasPerf[0] = "us, -a, -um sum";
pasPerf[1] = "us, -a, -um es";
pasPerf[2] = "us, -a, -um est";
pasPerf[3] = "ī, -ae, -a sumus";
pasPerf[4] = "ī, -ae, -a estis";
pasPerf[5] = "ī, -ae, -a sunt";
pasPerf[6] = "us, -a, -um eram";
pasPerf[7] = "us, -a, -um erās";
pasPerf[8] = "us, -a, -um erat";
```

```
        pasPerf[9] = "ī, -ae, -a erāmus";
        pasPerf[10] = "ī, -ae, -a erātis";
        pasPerf[11] = "ī, -ae, -a erant";
        pasPerf[12] = "us, -a, -um erō";
        pasPerf[13] = "us, -a, -um eris";
        pasPerf[14] = "us, -a, -um erit";
        pasPerf[15] = "ī, -ae, -a erimus";
        pasPerf[16] = "ī, -ae, -a eritis";
        pasPerf[17] = "ī, -ae, -a erunt";
        verbForms.add(pasPerf);
}

@SuppressWarnings("unused")
private static void printMap() {
        for (String key : map.keySet()) {
                System.out.println("Category: " + key);
                TreeMap<String, String> tempMap = map.get(key);
                for (String term : tempMap.keySet()) {
                        System.out.print("\t" + term);
                        System.out.println("  --> " + tempMap.get(term) + "\n");
                }
        }
}



public static void main(String[] args) throws IOException, InterruptedException {
        Scanner inFile = new Scanner(latin);
        makeMap(inFile);
        //printMap();
        LatinGUI GUI = new LatinGUI();
        while (!failMap.isEmpty()) {
                GUI.getNext();
                GUI.semaphore.acquire(2);
        }
        String reviewset = new String();
        for (String str : fails) {
                reviewset += "\t" + str;
                reviewset +="  --> " + terms.get(str) + "\n";
        }
```

```
            GUI.sendText("Finally! I thought you'd just quit. \n Nevertheless,"
                            + "you win. Review these: " + reviewset);
        }
}
```

PYTHON

```
''' SIMPLE PROGRAM TO GENERATE A RANDOM MATRIX AND COMPUTE DETERMINANT
      Motivation was to quickly make practice problems for
      linear algebra, primarily for computing determinants '''

from random import *
import numpy
def printf(a, *args):
    print(a, " ", end='')

''' GENERATE THE MATRIX
      m: specify number of rows (defaults to random in [3,7)
      forceSq: make the matrix square (default true)         '''
def randomMatrix(m = randrange(3,7), forceSq = True):
    n = random()
    if n < 0.5 or forceSq: n = m
    else: n = randrange(3,7)
    l = []
    printf("[")
    if(m <= 4):
        for i in range(m):
            subl = []
            for j in range(n):
                if(i == 0 and j == 0):
                    printf(1)
                    subl+=[1]
                else:
                    r = randrange(-6,7)
                    subl += [r]
                    printf(r)
            else:
                if(i == m-1):
                    l += [subl]
                    break
```

```
                else: printf(";")
                l += [subl]
    else:
        for i in range(m):
            subl = []
            for j in range(n):
                if(i == 0 and j == 0):
                    printf(1)
                    subl+=[1]
                else:
                    r = randrange(-4,4)
                    subl += [r]
                    printf(r)
            else:
                if(i == m-1):
                    l += [subl]
                    break
                else: printf(";")
                l += [subl]
    printf("]")
    return l

''' CHECK IF HAND CALCULATION OF DETERMINANT IS CORRECT '''
def testDet(mat):
    d = numpy.linalg.det(mat)
    print('\n')
    val = input("Enter calculated value: ")
    if abs(float(d) - float(val)) < 1:
        print(mat, d, "success", sep = "\n")
        return True
    else:
        print("Actual: ", d)
        return False
```

C

```
/* program to store stats about basketball players (homework assignment) */
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
/* structure to store stats */
struct player {
    char* name;
    float ppg;
    float mpg;
    float ppm;
    struct player*next;
};
/* function definitions */
void freeList(struct player*head);
void sortList(struct player*head);
void exchg(struct player*a,struct player*b);
void printList(struct player*head);

int main(int argc, char*argv[]){
    /* ensure we have the proper number of arguments */
    if(argc != 2){
        printf("Please supply a .txt following the binary exe when executing; exiting.
\n");
        return 0;
    }
    /* open the file */
    FILE*stats = fopen(argv[1], "r");
    char c;
    /* check if given bad file path */
    if(stats == 0){
        printf("ERROR: failed to open file.\n");
    }
    /* check if first character is EOF */
```

```
else if((c=fgetc(stats))==EOF){
    printf("STAT FILE IS EMPTY\n");
    fclose(stats);
}
/* file has data if we've arrived here */
else {
    /* allocate mem for head ptr and initialize all fields */
    struct player*head = (struct player*) malloc( sizeof( struct player ));
    head->ppg=0.0;
    head->mpg=0.0;
    head->ppm=0.0;
    head->name=(char*)malloc(255);
    head->next=NULL;
    /* declare a transition pointer */
    struct player*trans= head;
    /* get the first name, where we stored the first
     character already when checking for EOF */
    trans->name[0]=c;
    int i = 1;
    while( (c=fgetc(stats)) != '\n'){
        trans->name[i] = c;
        i++;
    }
    /* terminate the first name */
    trans->name[i]='\0';
    do{
        /* remove newlines from fgets */
        for (int i = 0; i<strlen(trans->name); i++){
            if( trans->name[i] == '\n' ){
                trans->name[i]='\0';
                break;
            }
        }
        /* break if we read DONE */
```

```
        if(strcmp(trans->name,"DONE")==0){ break; }
        /* scan in the float values */
        char temp[255];
        fgets(temp,sizeof(temp),stats);
        sscanf(temp, "%f",&trans->ppg);
        fgets(temp,sizeof(temp),stats);
        sscanf(temp, "%f",&trans->mpg);
        /* calculate points-per-minute */
        if(trans->ppg != 0.0 && trans->mpg != 0){
            trans->ppm= (float)trans->ppg/trans->mpg;
        } else {
            trans->ppm=0.0;
        }
        /* allocate mem for next player, initialize fields */
        trans->next = (struct player*) malloc(sizeof(struct player));
        trans=trans->next;
        trans->ppg=0.0;
        trans->mpg=0.0;
        trans->ppm=0.0;
        trans->name=(char*)malloc(255);
        trans->next=NULL;
    }  while( fgets(trans->name,256,stats ));
    /* after breaking, initialize a value for the last node's name
     (so that valgrind quits its yapping) */

    trans->name[0]='\0';
    /* close out the input file */
    fclose(stats);
    /* pass start of head into sorting, printing, freeing functs */
    sortList(&head[0]);
    printList(&head[0]);
    freeList(&head[0]);
}
/* EXIT_SUCCESS */
```

```
    return 0;
}
/* use bubble sort to order players by pt-per-min */
void sortList(struct player *head){
    /* use int as indicator */
    int change;
    /* declare temporary player */
    struct player *t;
    /* declare comparison ptr (init to NULL) */
    struct player *fin = NULL;
    /* ensure we have something to do */
    if(head==NULL){return;}
    do{
        /* init int to zero and temp ptr to head */
        change = 0;
        t=head;
        /* check for needed exchanges while not at end */
        while(t->next != fin){
            if(t->ppm < t->next->ppm){
                exchg(t,t->next);
                change = 1;
            }
            t=t->next;
        }
        /* update fin */
        fin=t;
    }  /* do until change remains zero through the loop */
    while(change);
}
/* funct to exchange the values in two ptrs (called in bubble sort) */
void exchg(struct player *a, struct player *b){
    /* swap the numbers (simple) */
    float tppg = a->ppg;
    float tmpg = a->mpg;
```

```
    float tppm = a->ppm;
    a->ppg = b->ppg;
    a->mpg = b->mpg;
    a->ppm = b->ppm;
    b->ppg = tppg;
    b->mpg = tmpg;
    b->ppm = tppm;
    /* swap the strings */
    char*tn = a->name;
    a->name = b->name;
    b->name = tn;
}
/* deallocate memory */
void freeList(struct player *head){
    struct player *temp;
    while(head!=NULL){

        temp = head;
        head = head->next;
        free(temp->name);
        free(temp);
    }
    free(head);
}

/* print list, stopping at empty name/null player */
void printList(struct player *head){
    struct player *temp = head;
    while(temp!=NULL && temp->name[0]!='\0'){
        printf("%s %f\n", temp->name, temp->ppm);
        temp=temp->next;
    }
}
```

MIPS ASSEMBLY (Implementation of C program above)

```
    .text
    .align 2            # included for good habit's sake
    .globl main
main:                   # Use to allocate space for first player
    li $a0,12           # allocate block from heap for player node
    li $v0,9
    syscall
    move $s0,$v0        # store the block in save-reg
    move $s3,$s0        # store the start address
in:
    li $v0,4            # prompt for name
    la $a0,npr
    syscall

    li $v0,9            # Allocate memory for string
    li $a0,64
    syscall

    move $a0,$v0        # get name from console
    li $a1,64           # seems safe to assume max of 64 char name
    li $v0,8
    syscall

    la $a1,done         # load address for comparison to DONE
    move $t3,$a0        # copy the name since we're about to iterate
strcomp:
    lbu $t0,0($a0)      # load corresponding chars from the words
    lbu $t1,0($a1)
    bne $t0,$t1,midIn   # go get more input if the chars arent equal
    beqz $t0,reset      # if we reach null, strings match and done with input
    addi $a0,$a0,1      # increment string pointers
    addi $a1,$a1,1
    j strcomp           # loop
```

```
midIn:
    sw $t3,0($s0)       # if we're back, store the name in node

    li $v0,4            # prompt for ppg
    la $a0,ppr
    syscall
    li $v0,6            # get ppg
    syscall
    mov.s $f1,$f0       # store ppg in $f1

    li $v0,4            # prompt for mpg
    la $a0,mpr
    syscall
    li $v0,6            # get mpg
    syscall

    # IF EITHER IS ZERO NEED TO HAVE ZERO IN PPM
    #!  >> If numerator is zero result will be zero anyways
    #!  >> If denominator is zero, must account for it
    mtc1 $zero,$f4
    c.eq.s $f4,$f0
    bc1t denomZ

    div.s $f3,$f1,$f0   # calc ppm via float division
finishIn:
    s.s $f3,4($s0)      # store result of division

    li $a0,20           # allocate 32-byte block from heap for player node
    li $v0,9
    syscall             # block will be returned in $v0
    sw $v0,8($s0)       # point to the new block
    lw $s0,8($s0)       # advance
    j in                # keep going
denomZ:
```

```
    mov.s $f3,$f4       # move zero into register for ppm
    j finishIn          # go back after the division instruction
reset:
    beq $s3,$s0,exit    # check if DONE was entered first
    move $s4,$s0        # store end address
    move $s0,$s3        # return to first node
    j sort
loop:
    lw $s0,8($s0)       # advance player 1
sort:
    lw $s1,8($s0)       # load player 2 node

    beq $s1,$s4,out     # if second player at end address we're done

    lw $t0,0($s0)       # load player 1
    lw $t1,0($s1)       # load player 2

    l.s $f1,4($s0)      # load floats for comparison
    l.s $f2,4($s1)
    c.lt.s $f1,$f2      # set condition flag to true if $f1<$f2
                            # so false--> sorted (want decending)
    bc1f loop
    # SWAP NEEDED IF WE ARRIVE HERE
    l.s $f1,4($s0)      # swap ppm
    l.s $f2,4($s1)
    s.s $f1,4($s1)
    s.s $f2,4($s0)

    sw $t0,0($s1)       # swap names
    sw $t1,0($s0)
    # NEED TO GO BACK OVER ONCE WE'VE SWAPPED
    move $s0,$s3        # so return to first node
    j sort              # ... and take it from the top
out:
```

```
    move $s0,$s3        # return to first node (now max)
printL:
    li $a1,10           # load newline ascii constant (for comparison)
    lw $t0,0($s0)       # load name into $t0
printN:
    lbu $a0,0($t0)
    beq $a0,$a1,printF  # print chars until '\n'
    li $v0,11
    syscall
    addi $t0,$t0,1      # iterate
    j printN            # keep printing name
printF:
    li $a0,32           # print a space
    li $v0,11
    syscall

    li $v0,2            # print ppm
    l.s $f12,4($s0)
    syscall

    li $v0,4            # print new line
    la $a0,nl
    syscall

    lw $s0,8($s0)       # advance node
    beq $s4,$s0,exit    # check if we're at end address
    j printL            # continue printing
exit:
    li $v0,10           # exit syscall
    syscall
    .data
buf: .space 64
npr: .asciiz "Enter player's last name:"
ppr: .asciiz "Enter player's points per game:"
```

```
mpr: .asciiz "Enter player's minutes per game:"

done:.asciiz "DONE\n"

nl:  .asciiz "\n"
```

MATLAB

```
function [cV,cE]=ComputeCutGraph(M,T)

% Based on Algorithm 3 in Gu's Computational Conformal Geometry (pp. 189)
% Owner: Ben Schwennesen (ben.schwennesen@live.com)
% Part of mesh processing research project for Duke Dept. of Mathematics

% *** Note: Development stopped since suitable open-source implementation
% was found.

% Find cut graph given mesh M and T, the full version of minimal
%   spanning tree (MST)

% Cut graph C is all edges not in T (must be s.c.)
%  Compute cut graph G of mesh
  % a. compute dual mesh ~M of M
    % i. each face corresponds to unique vertex
    % ii. each vertex corresponds to unique face
    % iii. each edge adjacent to faces f1,f2 in M corresponds to an edge ~e
        %   ~e = [~v(f1),~v(f2)]
  % b. generate minimum spanning tree ~T of vertices of ~M
  % c. G={e|~e not in ~T}

nV = M.nV;
if isempty(M.E)
    [~,E]=M.ComputeAdjacencyMatrix;
else
E=M.E;
[I,J] = find(E);
cV = sparse(nV,1);
cE = sparse(nV,2);
for m = 1:length(I)
  i = I(m); j = J(m);
  if (xor(E(i,j),E(j,i)) && T(i,j) == 0 && T(j,i) == 0)
```

```
    loc = find(C(i,:) == 0);
    if ~isempty(loc)
      cE(i,loc(1)) = j;
    end
    loc = find(C(j,:) == 0);
    if ~isempty(loc)
      cE(j,loc(1)) = i;
    end
    cV(i) = 1;
    cV(j) = 1;
  end
end
cV = find(cV);
end
end
```