import java.io.File;  
import java.util.ArrayList;  
import java.util.HashMap;  
  
public class VMtranslator {  
  
 public static ArrayList<File> getVMFiles(File dir){  
 File[] files = dir.listFiles();  
  
 ArrayList<File> result = new ArrayList<File>();  
  
 for (File f:files){  
 if (f.getName().endsWith(".vm")){  
 result.add(f);  
 }  
 }  
  
 return result;  
 }  
  
 public static void main(String[] args) {  
  
 if (args.length != 1){  
 System.out.println("Usage:java VMtranslator [filename|directory]");  
 } else {  
 String fileInName = args[0];  
  
 File fileIn = new File(fileInName);  
  
 String fileOutPath = "";  
  
 File fileOut;  
  
 CodeWriter writer;  
  
 ArrayList<File> vmFiles = new ArrayList<File>();  
  
 if (fileIn.isFile()) {  
 String path = fileIn.getAbsolutePath();  
  
 if (!Parser.getExt(path).equals(".vm")) {  
 throw new IllegalArgumentException(".vm files only.");  
 }  
  
 vmFiles.add(fileIn);  
  
 fileOutPath = fileIn.getAbsolutePath().substring(0, fileIn.getAbsolutePath().lastIndexOf(".")) + ".asm";  
 } else if (fileIn.isDirectory()) {  
 vmFiles = getVMFiles(fileIn);  
  
 if (vmFiles.size() == 0) {  
 throw new IllegalArgumentException("No vm file in this directory");  
 }  
  
 fileOutPath = fileIn.getAbsolutePath() + "/" + fileIn.getName() + ".asm";  
 }  
  
 fileOut = new File(fileOutPath);  
 writer = new CodeWriter(fileOut);  
  
 writer.writeInit();  
  
 for (File f : vmFiles) {  
 writer.setFileName(f);  
  
 Parser parser = new Parser(f);  
  
 int type = -1;  
  
 while (parser.hasMoreCommands()) {  
 parser.advance();  
  
 type = parser.commandType();  
  
 if (type == Parser.ARITHMETIC) {  
 writer.writeArithmetic(parser.arg1());  
 } else if (type == Parser.POP || type == Parser.PUSH) {  
 writer.writePushPop(type, parser.arg1(), parser.arg2());  
 } else if (type == Parser.LABEL) {  
 writer.writeLabel(parser.arg1());  
 } else if (type == Parser.GOTO) {  
 writer.writeGoto(parser.arg1());  
 } else if (type == Parser.IF) {  
 writer.writeIf(parser.arg1());  
 } else if (type == Parser.RETURN) {  
 writer.writeReturn();  
 } else if (type == Parser.FUNCTION) {  
 writer.writeFunction(parser.arg1(),parser.arg2());  
 } else if (type == Parser.CALL) {  
 writer.writeCall(parser.arg1(),parser.arg2());  
 }  
 }  
 }  
  
 writer.close();  
 System.out.println("File created : " + fileOutPath);  
 }  
 }  
  
}

import java.io.File;  
import java.io.FileNotFoundException;  
import java.util.ArrayList;  
import java.util.IllegalFormatException;  
import java.util.Scanner;  
import java.util.regex.Matcher;  
import java.util.regex.Pattern;  
  
public class Parser {  
 private Scanner cmds;  
 private String currentCmd;  
 public static final int ARITHMETIC = 0;  
 public static final int PUSH = 1;  
 public static final int POP = 2;  
 public static final int LABEL = 3;  
 public static final int GOTO = 4;  
 public static final int IF = 5;  
 public static final int FUNCTION = 6;  
 public static final int RETURN = 7;  
 public static final int CALL = 8;  
 public static final ArrayList<String> arithmeticCmds = new ArrayList<String>();  
 private int argType;  
 private String argument1;  
 private int argument2;  
  
 static {  
 arithmeticCmds.add("add");arithmeticCmds.add("sub");arithmeticCmds.add("neg");arithmeticCmds.add("eq");arithmeticCmds.add("gt");  
 arithmeticCmds.add("lt");arithmeticCmds.add("and");arithmeticCmds.add("or");arithmeticCmds.add("not");  
 }  
  
 public Parser(File fileIn) {  
 argType = -1;  
 argument1 = "";  
 argument2 = -1;  
  
 try {  
 cmds = new Scanner(fileIn);  
  
 String preprocessed = "";  
 String line = "";  
  
 while(cmds.hasNext()){  
 line = noComments(cmds.nextLine()).trim();  
  
 if (line.length() > 0) {  
 preprocessed += line + "\n";  
 }  
 }  
  
 cmds = new Scanner(preprocessed.trim());  
 } catch (FileNotFoundException e) {  
 System.out.println("File not found!");  
 }  
 }  
  
 public boolean hasMoreCommands(){  
 return cmds.hasNextLine();  
 }  
  
 public void advance(){  
 currentCmd = cmds.nextLine();  
 argument1 = "";//initialize arg1  
 argument2 = -1;//initialize arg2  
  
 String[] segs = currentCmd.split(" ");  
  
 if (segs.length > 3){  
 throw new IllegalArgumentException("Too much arguments!");  
 }  
  
 if (arithmeticCmds.contains(segs[0])){  
 argType = ARITHMETIC;  
 argument1 = segs[0];  
  
 } else if (segs[0].equals("return")) {  
 argType = RETURN;  
 argument1 = segs[0];  
  
 } else {  
 argument1 = segs[1];  
  
 if(segs[0].equals("push")){  
 argType = PUSH;  
 } else if(segs[0].equals("pop")){  
 argType = POP;  
 } else if(segs[0].equals("label")){  
 argType = LABEL;  
 } else if(segs[0].equals("if-goto")){  
 argType = IF;  
 } else if (segs[0].equals("goto")){  
 argType = GOTO;  
 } else if (segs[0].equals("function")){  
 argType = FUNCTION;  
 } else if (segs[0].equals("call")){  
 argType = CALL;  
 } else {  
 throw new IllegalArgumentException("Unknown Command Type!");  
 }  
  
 if (argType == PUSH || argType == POP || argType == FUNCTION || argType == CALL){  
 try {  
 argument2 = Integer.parseInt(segs[2]);  
 } catch (Exception e) {  
 throw new IllegalArgumentException("Argument2 is not an integer!");  
 }  
 }  
 }  
 }  
  
 public int commandType(){  
 if (argType != -1) {  
 return argType;  
 } else {  
 throw new IllegalStateException("No command!");  
 }  
 }  
  
 public String arg1(){  
 if (commandType() != RETURN){  
 return argument1;  
 } else {  
 throw new IllegalStateException("Can not get arg1 from a RETURN type command!");  
 }  
 }  
  
 public int arg2(){  
 if (commandType() == PUSH || commandType() == POP || commandType() == FUNCTION || commandType() == CALL){  
 return argument2;  
 } else {  
 throw new IllegalStateException("Can not get arg2!");  
 }  
 }  
  
 public static String noComments(String strIn){  
 int position = strIn.indexOf("//");  
 if (position != -1){  
 strIn = strIn.substring(0, position);  
 }  
  
 return strIn;  
 }  
  
 public static String noSpaces(String strIn){  
 String result = "";  
 if (strIn.length() != 0){  
 String[] segs = strIn.split(" ");  
  
 for (String s: segs){  
 result += s;  
 }  
 }  
  
 return result;  
 }  
  
 public static String getExt(String fileName){  
 int index = fileName.lastIndexOf('.');  
 if (index != -1){  
 return fileName.substring(index);  
 } else {  
 return "";  
 }  
 }  
}

import java.io.File;  
import java.io.FileNotFoundException;  
import java.io.PrintWriter;  
import java.util.regex.Matcher;  
import java.util.regex.Pattern;  
  
public class CodeWriter {  
  
 private int arthJumpFlag;  
 private PrintWriter outPrinter;  
 private static final Pattern labelReg = Pattern.compile("^[^0-9][0-9A-Za-z\\\_\\:\\.\\$]+");  
 private static int labelCnt = 0;  
 private static String fileName = "";  
  
 public CodeWriter(File fileOut) {  
 try {  
 fileName = fileOut.getName();  
 outPrinter = new PrintWriter(fileOut);  
 arthJumpFlag = 0;  
  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public void setFileName(File fileOut){  
 fileName = fileOut.getName();  
 }  
  
 public void writeArithmetic(String command){  
 if (command.equals("add")){  
 outPrinter.print(arithmeticTemplate1() + "M=M+D\n");  
 } else if (command.equals("sub")){  
 outPrinter.print(arithmeticTemplate1() + "M=M-D\n");  
 } else if (command.equals("and")){  
 outPrinter.print(arithmeticTemplate1() + "M=M&D\n");  
 } else if (command.equals("or")){  
 outPrinter.print(arithmeticTemplate1() + "M=M|D\n");  
 } else if (command.equals("gt")){  
 outPrinter.print(arithmeticTemplate2("JLE"));//not <=  
 arthJumpFlag++;  
 } else if (command.equals("lt")){  
 outPrinter.print(arithmeticTemplate2("JGE"));//not >=  
 arthJumpFlag++;  
 } else if (command.equals("eq")){  
 outPrinter.print(arithmeticTemplate2("JNE"));//not <>  
 arthJumpFlag++;  
 } else if (command.equals("not")){  
 outPrinter.print("@SP\nA=M-1\nM=!M\n");  
 } else if (command.equals("neg")){  
 outPrinter.print("D=0\n@SP\nA=M-1\nM=D-M\n");  
 } else {  
 throw new IllegalArgumentException("Call writeArithmetic() for a non-arithmetic command");  
 }  
 }  
  
 public void writePushPop(int command, String segment, int index){  
 if (command == Parser.PUSH){  
 if (segment.equals("constant")){  
 outPrinter.print("@" + index + "\n" + "D=A\n@SP\nA=M\nM=D\n@SP\nM=M+1\n");  
 } else if (segment.equals("local")){  
 outPrinter.print(pushTemplate1("LCL",index,false));  
 } else if (segment.equals("argument")){  
 outPrinter.print(pushTemplate1("ARG",index,false));  
 } else if (segment.equals("this")){  
 outPrinter.print(pushTemplate1("THIS",index,false));  
 } else if (segment.equals("that")){  
 outPrinter.print(pushTemplate1("THAT",index,false));  
 } else if (segment.equals("temp")){  
 outPrinter.print(pushTemplate1("R5", index + 5,false));  
 } else if (segment.equals("pointer") && index == 0){  
 outPrinter.print(pushTemplate1("THIS",index,true));  
 } else if (segment.equals("pointer") && index == 1){  
 outPrinter.print(pushTemplate1("THAT",index,true));  
 } else if (segment.equals("static")){  
 outPrinter.print("@" + fileName + index + "\n" + "D=M\n@SP\nA=M\nM=D\n@SP\nM=M+1\n");  
 }  
  
 } else if(command == Parser.POP){  
 if (segment.equals("local")){  
 outPrinter.print(popTemplate1("LCL",index,false));  
 } else if (segment.equals("argument")){  
 outPrinter.print(popTemplate1("ARG",index,false));  
 } else if (segment.equals("this")){  
 outPrinter.print(popTemplate1("THIS",index,false));  
 } else if (segment.equals("that")){  
 outPrinter.print(popTemplate1("THAT",index,false));  
 } else if (segment.equals("temp")){  
 outPrinter.print(popTemplate1("R5", index + 5,false));  
 } else if (segment.equals("pointer") && index == 0){  
 outPrinter.print(popTemplate1("THIS",index,true));  
 } else if (segment.equals("pointer") && index == 1){  
 outPrinter.print(popTemplate1("THAT",index,true));  
 } else if (segment.equals("static")){  
 outPrinter.print("@" + fileName + index + "\nD=A\n@R13\nM=D\n@SP\nAM=M-1\nD=M\n@R13\nA=M\nM=D\n");  
 }  
  
 } else {  
 throw new IllegalArgumentException("Call writePushPop() for a non-pushpop command");  
 }  
 }  
  
 public void writeLabel(String label){  
 Matcher m = labelReg.matcher(label);  
 if (m.find()){  
 outPrinter.print("(" + label +")\n");  
 } else {  
 throw new IllegalArgumentException("Wrong label format!");  
 }  
 }  
  
 public void writeGoto(String label){  
 Matcher m = labelReg.matcher(label);  
 if (m.find()){  
 outPrinter.print("@" + label +"\n0;JMP\n");  
 } else {  
 throw new IllegalArgumentException("Wrong label format!");  
 }  
 }  
  
 public void writeIf(String label){  
 Matcher m = labelReg.matcher(label);  
 if (m.find()){  
 outPrinter.print(arithmeticTemplate1() + "@" + label +"\nD;JNE\n");  
 } else {  
 throw new IllegalArgumentException("Wrong label format!");  
 }  
 }  
  
 public void writeInit(){  
 outPrinter.print("@256\n" +  
 "D=A\n" +  
 "@SP\n" +  
 "M=D\n"  
 );  
 writeCall("Sys.init",0);  
 }  
  
 public void writeCall(String functionName, int numArgs){  
 String newLabel = "RETURN\_LABEL" + (labelCnt++);  
  
 outPrinter.print("@" + newLabel + "\n" + "D=A\n@SP\nA=M\nM=D\n@SP\nM=M+1\n");  
 outPrinter.print(pushTemplate1("LCL",0,true));  
 outPrinter.print(pushTemplate1("ARG",0,true));  
 outPrinter.print(pushTemplate1("THIS",0,true));  
 outPrinter.print(pushTemplate1("THAT",0,true));  
  
 outPrinter.print("@SP\n" +  
 "D=M\n" +  
 "@5\n" +  
 "D=D-A\n" +  
 "@" + numArgs + "\n" +  
 "D=D-A\n" +  
 "@ARG\n" +  
 "M=D\n" +  
 "@SP\n" +  
 "D=M\n" +  
 "@LCL\n" +  
 "M=D\n" +  
 "@" + functionName + "\n" +  
 "0;JMP\n" +  
 "(" + newLabel + ")\n"  
 );  
 }  
  
 public void writeReturn(){  
 outPrinter.print(returnTemplate());  
 }  
  
 public void writeFunction(String functionName, int numLocals){  
 outPrinter.print("(" + functionName +")\n");  
 for (int i = 0; i < numLocals; i++){  
 writePushPop(Parser.PUSH,"constant",0);  
 }  
 }  
  
 public String preFrameTemplate(String position){  
 return "@R11\n" +  
 "D=M-1\n" +  
 "AM=D\n" +  
 "D=M\n" +  
 "@" + position + "\n" +  
 "M=D\n";  
 }  
  
 public String returnTemplate(){  
 return "@LCL\n" +  
 "D=M\n" +  
 "@R11\n" +  
 "M=D\n" +  
 "@5\n" +  
 "A=D-A\n" +  
 "D=M\n" +  
 "@R12\n" +  
 "M=D\n" +  
 popTemplate1("ARG",0,false) +  
 "@ARG\n" +  
 "D=M\n" +  
 "@SP\n" +  
 "M=D+1\n" +  
 preFrameTemplate("THAT") +  
 preFrameTemplate("THIS") +  
 preFrameTemplate("ARG") +  
 preFrameTemplate("LCL") +  
 "@R12\n" +  
 "A=M\n" +  
 "0;JMP\n";  
 }  
  
 public void close(){  
 outPrinter.close();  
 }  
  
 private String arithmeticTemplate1(){  
 return "@SP\n" +  
 "AM=M-1\n" +  
 "D=M\n" +  
 "A=A-1\n";  
 }  
  
 private String arithmeticTemplate2(String type){  
 return "@SP\n" +  
 "AM=M-1\n" +  
 "D=M\n" +  
 "A=A-1\n" +  
 "D=M-D\n" +  
 "@FALSE" + arthJumpFlag + "\n" +  
 "D;" + type + "\n" +  
 "@SP\n" +  
 "A=M-1\n" +  
 "M=-1\n" +  
 "@CONTINUE" + arthJumpFlag + "\n" +  
 "0;JMP\n" +  
 "(FALSE" + arthJumpFlag + ")\n" +  
 "@SP\n" +  
 "A=M-1\n" +  
 "M=0\n" +  
 "(CONTINUE" + arthJumpFlag + ")\n";  
  
 }  
  
 private String pushTemplate1(String segment, int index, boolean isDirect){  
 String noPointerCode = (isDirect)? "" : "@" + index + "\n" + "A=D+A\nD=M\n";  
  
 return "@" + segment + "\n" +  
 "D=M\n" +  
 noPointerCode +  
 "@SP\n" +  
 "A=M\n" +  
 "M=D\n" +  
 "@SP\n" +  
 "M=M+1\n";  
 }  
  
 private String popTemplate1(String segment, int index, boolean isDirect){  
 String noPointerCode = (isDirect)? "D=A\n" : "D=M\n@" + index + "\nD=D+A\n";  
  
 return "@" + segment + "\n" +  
 noPointerCode +  
 "@R13\n" +  
 "M=D\n" +  
 "@SP\n" +  
 "AM=M-1\n" +  
 "D=M\n" +  
 "@R13\n" +  
 "A=M\n" +  
 "M=D\n";  
 }  
}











