// SymbolTable.java

import java.util.HashMap;

public class SymbolTable

{

    private HashMap<String, Integer> myMap;

    int nextEmpty;

    public SymbolTable(){

        myMap = new HashMap<>();

*// initialize existing symbols*

        myMap.put("SP", 0);

        myMap.put("LCL", 1);

        myMap.put("ARG", 2);

        myMap.put("THIS", 3);

        myMap.put("THAT", 4);

        myMap.put("SCREEN", 16384);

        myMap.put("KBD", 24576);

*// R0 - R15*

        for(int i = 0; i < 16; i++){

            myMap.put("R" + i, i);

        }

        nextEmpty = 16;

    }

    public void addEntry(String symbol, int index){

        myMap.put(symbol, index);

    }

    public boolean contains(String symbol){

        return myMap.containsKey(symbol);

    }

    public int GetAddress(String symbol){

*// fix label exists but not defined*

        if(!contains(symbol)){

            myMap.put(symbol, nextEmpty);

            nextEmpty++;

        }

        return myMap.get(symbol);

    }

}

// Parser.java

import java.io.FileNotFoundException;

import java.util.Scanner;

import java.io.File;

public class Parser

{

    Scanner s;

    String currentCommand;

    boolean reachedEnd;

    int lineNumber;

    public Parser(File file) throws FileNotFoundException{

        s = new Scanner(file);

        currentCommand = "";

        reachedEnd = false;

        lineNumber = 0;

    }

    public boolean hasMoreCommands(){

        if(s.hasNextLine()){

            return true;

        } else{

            if(reachedEnd){

                return false;

            }

            reachedEnd = true;

        }

        return true;

    }

    public void advance(){

        if(!s.hasNextLine()){

            reachedEnd = true;

            return;

        }

        currentCommand = s.nextLine();

*// skip lines with comments*

        if(currentCommand.equals("") || currentCommand.startsWith("//")){

            advance();

            return;

        }

*// remove comments*

        if(currentCommand.contains("//")){

            currentCommand = currentCommand.substring(0, currentCommand.indexOf("//"));

        }

*// remove whitespace*

        currentCommand = currentCommand.replaceAll("\\s", "");

        currentCommand = currentCommand.replaceAll("\t", "");

        if(!commandType().equals("L\_COMMAND")){

            lineNumber++;

        }

    }

    public String commandType(){

        if(currentCommand.contains("@")){

            return "A\_COMMAND";

        } else if(currentCommand.contains("(") && currentCommand.contains(")")){

            return "L\_COMMAND";

        } else if(currentCommand.contains("=") || currentCommand.contains(";")){

            return "C\_COMMAND";

        }

        return "INVALID";

    }

    public String symbol(){

        if(currentCommand.contains("@")){

            return currentCommand.substring(currentCommand.indexOf("@") + 1);

        } else if(currentCommand.contains("(") && currentCommand.contains(")")){

            return currentCommand.substring(

                currentCommand.indexOf("(") + 1,

                currentCommand.indexOf(")"));

        }

        return "";

    }

    public String getCommand(){

        return currentCommand;

    }

    public int getLineNumber(){

        return lineNumber;

    }

}

// Assembler.java

import java.io.FileNotFoundException;

import java.io.FileWriter;

import java.io.IOException;

import java.io.File;

import java.util.HashMap;

*// instruction decoding stuff*

*// i xx a cccccc ddd jjj*

*// 15 14-13, 12, 11-6, 5-3 2-0*

*// bit 15: 0 = use A , 1 = use C*

*// bit 13-14: idk*

*// bit 12: used for comp (in C type)*

*// bit 11-6: used for com (in C type)*

*// bit 5-3: destination (in C type)*

*// 3: control writeM*

*// 4: write ALU output to D or not*

*// 5: choose ALU output at the mux or no*

*// bit 2-0: jump (in C type)*

*// destination   d1 (bit 5)   d2 (bit 4)   d3 (bit 3)*

*// M                 x            x            1*

*// D                 x            1            x*

*// A                 1            x            x*

*// meaning  j1 (bit 2)  j2 (bit 1)  j3 (bit 0)*

*// GT           x             x            1*

*// EQ           x             1            x*

*// LT           1             x            x*

*// jump if one of these condition is fulfilled*

public class Assembler

{

    public static void main(String[] args) {

*// usage: java Assembler [filename]*

        HashMap<String,String> jumpMap = new HashMap<>();

        jumpMap.put("JGT", "001");

        jumpMap.put("JEQ", "010");

        jumpMap.put("JGE", "011");

        jumpMap.put("JLT", "100");

        jumpMap.put("JNE", "101");

        jumpMap.put("JLE", "110");

        jumpMap.put("JMP", "111");

        HashMap<String,String> compMap = new HashMap<>();

        compMap.put("0", "101010");

        compMap.put("1", "111111");

        compMap.put("-1", "111010");

        compMap.put("D", "001100");

        compMap.put("A", "110000");

        compMap.put("!D", "001101");

        compMap.put("!A", "110001");

        compMap.put("-D", "001111");

        compMap.put("-A", "110011");

        compMap.put("D+1", "011111");

        compMap.put("A+1", "110111");

        compMap.put("D-1", "001110");

        compMap.put("A-1", "110010");

        compMap.put("D+A", "000010");

        compMap.put("D-A", "010011");

        compMap.put("A-D", "000111");

        compMap.put("D&A", "000000");

        compMap.put("D|A", "010101");

        HashMap<String,String> compMap\_A = new HashMap<>();

        compMap\_A.put("0", "101010");

        compMap\_A.put("1", "111111");

        compMap\_A.put("-1", "111010");

        compMap\_A.put("D", "001100");

        compMap\_A.put("M", "110000");

        compMap\_A.put("!D", "001101");

        compMap\_A.put("!M", "110001");

        compMap\_A.put("-D", "001111");

        compMap\_A.put("-M", "110011");

        compMap\_A.put("D+1", "011111");

        compMap\_A.put("M+1", "110111");

        compMap\_A.put("D-1", "001110");

        compMap\_A.put("M-1", "110010");

        compMap\_A.put("D+M", "000010");

        compMap\_A.put("D-M", "010011");

        compMap\_A.put("M-D", "000111");

        compMap\_A.put("D&M", "000000");

        compMap\_A.put("D|M", "010101");

        File file = new File(args[0]);

        Parser parser = null;

        try{

            parser = new Parser(file);

        } catch(FileNotFoundException e){

            System.out.println(e);

            return;

        }

*// populate symboltable*

        SymbolTable symbolTable = new SymbolTable();

*// first pass: add new symbols to the table*

        while(parser.hasMoreCommands()){

            String type = parser.commandType();

*// parse A/L command*

            if(type.equals("L\_COMMAND")){

*// get symbol*

                String symbol = parser.symbol();

*// add new entries but leave numerical labels alone*

                if(!Character.isDigit(symbol.charAt(0))){

*// check if it already exists*

                    if(!symbolTable.contains(symbol)){

                        symbolTable.addEntry(symbol,parser.getLineNumber());

                    }

                }

            }

            parser.advance();

        }

*// return to the beginning of file*

        try{

            parser = new Parser(file);

        } catch(FileNotFoundException e){

            System.out.println(e);

            return;

        }

        FileWriter fw = null;

        try{

            fw = new FileWriter(new File(args[0].substring(0, args[0].indexOf(".")) + ".hack"), false);

        } catch(IOException e){

            System.out.println(e);

            return;

        }

        StringBuilder outString = new StringBuilder();

*// second pass: parse instructions*

        while(parser.hasMoreCommands()){

            String type = parser.commandType();

*// handle A-type*

            if(type.equals("A\_COMMAND")){

*// get symbol*

                String symbol = parser.symbol();

*// change numerical label to string*

                if(!Character.isDigit(symbol.charAt(0))){

                    symbol = Integer.toBinaryString(symbolTable.GetAddress(symbol));

                } else{

                    symbol = Integer.toBinaryString(Integer.parseInt(symbol));

                }

*// create A-type instruction*

                for(int i = 16; i > symbol.length(); i--){

                    outString.append("0");

                }

                outString.append(symbol);

                outString.append("\n");

            } else if(type.equals("C\_COMMAND")){ *// handle C-type*

                outString.append("111");

                String dbits = "000";

                String jbits = "000";

                String compBits;

                String command = parser.getCommand();

*// create J Bits*

                if(command.contains(";")){

                    jbits = jumpMap.get(command.substring(command.indexOf(";")+1));

                    command = command.substring(0, command.indexOf(";")); *// remove jump section*

                }

                if(command.contains("=")){

                    String destinationSegment = command.substring(0, command.indexOf("="));

                    if(destinationSegment.contains("A")){

                        dbits = "1" + dbits.substring(1);

                    }

                    if(destinationSegment.contains("D")){

                        dbits = dbits.substring(0, 1) + "1" + dbits.substring(2);

                    }

                    if(destinationSegment.contains("M")){

                        dbits = dbits.substring(0, 2) + "1";

                    }

                    String argSegment = command.substring(command.indexOf("=")+1);

                    if(argSegment.contains("A")){

                        outString.append("0");

                        compBits = compMap.get(argSegment);

                    } else{

                        if(argSegment.contains("M")){

                            outString.append("1");

                        } else{

                            outString.append("0");

                        }

                        compBits = compMap\_A.get(argSegment);

                    }

                } else{

                    if(command.contains("A")){

                        outString.append("0");

                        compBits = compMap.get(command);

                    } else{

                        if(command.contains("M")){

                            outString.append("1");

                        } else{

                            outString.append("0");

                        }

                        compBits = compMap\_A.get(command);

                    }

                    dbits = "000";

                }

                outString.append(compBits + dbits + jbits + "\n");

            }

            parser.advance();

        }

        try{

            fw.write(outString.toString());

            fw.close();

        } catch(IOException e){

            System.out.println(e);

            return;

        }

    }

}

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated