```
public class parser {
    static int A_COMMAND = 0;
    static int C_COMMAND = 1;
    static int L_COMMAND = 2;

    private In file;
    private String currentCommand;

    //find file name
    public parser(String filename)
    {
        file = new In(filename);
    }
    //check for more lines
    public boolean hasMoreCommands()
    {
        if(file.hasNextLine())
        {
            return true;
        }
        else
        {
            file.close();
            return false;
        }
    }
    public void advance()
    {
        //continue to next line and get rid of spaces and comments
        String line;
        do{line = file.readLine().trim();}
        while(line.equals("") || line.substring(0,2).equals("//"));
        {
            String[] parts = line.split("//");
        }
        currentCommand = parts[0];
        currentCommand = currentCommand.replace(" ", "");
    }

    //check type of command
    public int commandType()
    {
        char iValue = currentCommand.charAt(0);
        if(iValue == '@')
        {
            return A_COMMAND;
        }
        else if(iValue == '(')
        {
            return L_COMMAND;
        }
```

```
    else
    {
        return C_COMMAND;
    }
}

//c-command values
//comp
public String comp()
{
    String comp = null;
    if(this.commandType() == C_COMMAND)
    {
        if(currentCommand.indexOf("=") != -1)
        {//get commande right of equals sign
            comp = currentCommand.split("=")[1];
        }
        else if(currentCommand.indexOf(";") != -1)
        {
            comp = currentCommand.split(";")[0];
        }
        else { }
    }
    return comp;
}
//dest
 public String dest()
 {
    String dest = null;
    if(this.commandType() == C_COMMAND && currentCommand.indexOf("=") != -1)
    {
        dest = currentCommand.split("=")[0];
    }
    return dest;
}
//jump
 public String jump()
 {
    String jump = null;
    if(this.commandType() == C_COMMAND)
    {
        if(currentCommand.indexOf(";") != -1)
        {
            jump = currentCommand.split(";")[1];
        }
        else { }
    }
    return jump;
}
//symbols
 public String symbol()
```

```
    {
        String symbol = null;
        if(this.commandType() == A_COMMAND )
        {
            symbol = currentCommand.substring(1, currentCommand.length());
        }
        else if(this.commandType() == L_COMMAND)
        {
            symbol = currentCommand.substring(1, currentCommand.length() - 1);
        }
        else { }
        return symbol;
    }
}
```

```java
public class code
{
    //translate hack to binary for comp
    public String comp(String comphack) {
        String compbinary = null;
        if(comphack == null)
        {
            return null;
        }
        else if(comphack.equals("0"))
        {
            compbinary = "0101010";
        }
        else if(comphack.equals("1"))
        {
            compbinary = "0111111";
        }
        else if(comphack.equals("-1"))
        {
            compbinary = "0111010";
        }
        else if(comphack.equals("D"))
        {
            compbinary = "0001100";
        }
        else if(comphack.equals("A"))
        {
            compbinary = "0110000";
        }
        else if(comphack.equals("!D"))
        {
            compbinary = "0001101";
        }
        else if(comphack.equals("!A"))
        {
            compbinary = "0110001";
        }
        else if(comphack.equals("-D"))
        {
            compbinary = "0001111";
        }
        else if(comphack.equals("-A"))
        {
            compbinary = "0110011";
        }
        else if(comphack.equals("D+1"))
        {
            compbinary = "0011111";
        }
        else if(comphack.equals("A+1"))
        {
```

```
    compbinary = "0110111";
}
else if(comphack.equals("D-1"))
{
    compbinary = "0001110";
}
else if(comphack.equals("A-1"))
{
    compbinary = "0110010";
}
else if(comphack.equals("D+A"))
{
    compbinary = "0000010";
}
else if(comphack.equals("D-A"))
{
    compbinary = "0010011";
}
else if(comphack.equals("A-D"))
{
    compbinary = "0000111";
}
else if(comphack.equals("D&A"))
{
    compbinary = "0000000";
}
else if(comphack.equals("D|A"))
{
    compbinary = "0010101";
}
else if(comphack.equals("M"))
{
    compbinary = "1110000";
}
else if(comphack.equals("!M"))
{
    compbinary = "1110001";
}
else if(comphack.equals("-M"))
{
    compbinary = "1110011";
}
else if(comphack.equals("M+1"))
{
    compbinary = "1110111";
}
else if(comphack.equals("M-1"))
{
    compbinary = "1110010";
}
else if(comphack.equals("D+M"))
```

```
    {
        compbinary = "1000010";
    }
    else if(comphack.equals("D-M"))
    {
        compbinary = "1010011";
    }
    else if(comphack.equals("M-D"))
    {
        compbinary = "1000111";
    }
    else if(comphack.equals("D&M"))
    {
        compbinary = "1000000";
    }
    else if(comphack.equals("D|M"))
    {
        compbinary = "1010101";
    }
    else{ }
    return compbinary;
}
//translate hack to binary for dest
public String dest(String desthack) {
    String destbinary = null;
    if(desthack == null)
    {
        destbinary = "000";
    }
    else if(desthack.equals("M"))
    {
        destbinary = "001";
    }
    else if(desthack.equals("D"))
    {
        destbinary = "010";
    }
    else if(desthack.equals("MD"))
    {
        destbinary = "011";
    }
    else if(desthack.equals("A"))
    {
        destbinary = "100";
    }
    else if(desthack.equals("AM"))
    {
        destbinary = "101";
    }
    else if(desthack.equals("AD"))
    {
```

```
            destbinary = "110";
        }
        else if(desthack.equals("AMD"))
        {
            destbinary = "111";
        }
        else{ }
        return destbinary;
    }
    //translate hack to binary for jump
    public String jump(String jumphack) {
        String jumpbinary;
        if(jumphack == null)
        {
            jumpbinary = "000";
        }
        else if(jumphack.equals("JGT"))
        {
            jumpbinary = "001";
        }
        else if(jumphack.equals("JEQ"))
        {
            jumpbinary = "010";
        }
        else if(jumphack.equals("JGE"))
        {
            jumpbinary = "011";
        }
        else if(jumphack.equals("JLT"))
        {
            jumpbinary = "100";
        }
        else if(jumphack.equals("JNE"))
        {
            jumpbinary = "101";
        }
        else if(jumphack.equals("JLE"))
        {
            jumpbinary = "110";
        }
        else if(jumphack.equals("JMP"))
        {
            jumpbinary = "111";
        }
        else
        {
            jumpbinary = "000";
        }
        return jumpbinary;
    }
}
```

```java
import java.util.Hashtable;

public class Symbols {
    private Hashtable<String, Integer> symboltable;
//translate hack to binary for symbols
    public Symbols() {
        symboltable = new Hashtable<String, Integer>();
        symboltable.put("SP", 0);
        symboltable.put("LCL", 1);
        symboltable.put("ARG", 2);
        symboltable.put("THIS", 3);
        symboltable.put("THAT", 4);
        symboltable.put("SCREEN", 16384);
        symboltable.put("KBD", 24576);
        symboltable.put("R0", 0);
        symboltable.put("R1", 1);
        symboltable.put("R2", 2);
        symboltable.put("R3", 3);
        symboltable.put("R4", 4);
        symboltable.put("R5", 5);
        symboltable.put("R6", 6);
        symboltable.put("R7", 7);
        symboltable.put("R8", 8);
        symboltable.put("R9", 9);
        symboltable.put("R10", 10);
        symboltable.put("R11", 11);
        symboltable.put("R12", 12);
        symboltable.put("R13", 13);
        symboltable.put("R14", 14);
        symboltable.put("R15", 15);
    }

    public void addEntry(String symbol, int address)
    {
        symboltable.put(symbol, address);
    }

    public boolean contains(String symbol)
    {
        return symboltable.containsKey(symbol);
    }

    public int getAddress(String symbol)
    {
        if(symboltable.containsKey(symbol))
        {
            return symboltable.get(symbol);
        }
        else
        {
            return -1;
```

```
        }
    }
}
```

```
import java.lang.StringBuilder;

public class assembler {
    static int A_COMMAND = 0;
    static int C_COMMAND = 1;
    static int L_COMMAND = 2;
    static int VAR = 16;

    public static void main(String[] args)
    {
        String fileName = args[0];
        parser parser = new parser(fileName);
        code codes = new code();
        symbols sym = new symbols();

        firstPass(parser, sym);

        parser = new parser(fileName);
        String outName = fileName.split("[.]")[0];
        Out output = new Out(outName);
        secondPass(parser, codes, sym, output);

        output.close();
    }
    //see if L command or not
    public static void firstPass(parser parser, symbols sym)
    {
        int counter = -1;
        String symbol = null;
        while(parser.hasMoreCommands())
        {
            parser.advance();
            if(parser.commandType() == L_COMMAND)
            {
                symbol = parser.symbol();
                sym.addEntry(symbol, counter + 1);
            }
            else
            {
                counter++;
            }
        }
    }

    public static void secondPass(parser parser, code codes, symbol sym, Out output)
    {
    //get hack and translate into binary for a and c commands
        while(parser.hasMoreCommands())
        {
            parser.advance();
            if(parser.commandType() == A_COMMAND)
```

```
            {
                String symbola = parser.symbol();
                String comma = generateACommand(symbola, sym);
                output.println(comma);
            }
            else if(parser.commandType() == C_COMMAND)
            {
                String dest = parser.dest();
                String comp = parser.comp();
                String jump = parser.jump();
                String commc = generateCCommand(dest, comp, jump, codes);
                output.println(commc);
            }
        }
    }
//a commnads
    private static String generateACommand(String symbol, symbol sym)
    {
        String code = null;
        int address;
        if(isNumeric(symbol))
        {
            address = Integer.parseInt(symbol);
        }
        else
        {
            if(sym.contains(symbol))
            {
                address = sym.getAddress(symbol);
            }
            else
            {
                address = VAR++;
                sym.addEntry(symbol, address);
            }
        }
        String binary = Integer.toBinaryString(address);
        code = changeLength(binary, 16);
        return code;
    }

    private static boolean isNumeric(String str)
    {
        if(str.length() == 0)
        {
            return false;
        }
        for(char c : str.toCharArray())
        {
            if(!Character.isDigit(c))
            {
```

```
                return false;
            }
        }
        return true;
    }

    private static String changeLength(String str, int number)
    {
        int zero = number - str.length();
        StringBuilder zeros = new StringBuilder();
        for(int i = zero; i > 0; i--)
        {
            zeros.append("0");
        }
        zeros.append(str);
        return zeros.toString();
    }
//c commands
    public static String generateCCommand(String dest, String comp, String jump, code
codes)
    {
        StringBuilder code = new StringBuilder();
        code.append("111");
        String compCode = codes.comp(comp);
        code.append(compCode);
        String destCode = codes.dest(dest);
        code.append(destCode);
        String jumpCode = codes.jump(jump);
        code.append(jumpCode);
        return code.toString();
    }


}
```