BUILDING A MODERN COMPUTER FROM BASICS: NAND TO TETRIS

By

Rohit Subramanian 17BCE1291

Tarunika J 17BEC1091

Shruti S 17BEC1143

A report submitted to the

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING

in partial fulfilment of the requirements for the project of the course of

CSE2005 OPERATING SYSTEMS

in

B.Tech. COMPUTER SCIENCE AND ENGINEERING

VIT University, Chennai

Vandalur – Kelambakkam Road Chennai

600127

April 2019

BONAFIDE CERTIFICATE

Certified that this project report entitled "**BUILDING A MODERN COMPUTER FROM BASICS: NAND TO TETRIS ”** is a bonafide work of

Rohit Subramanian 17BCE1291

Tarunika J 17BEC1091

Shruti S 17BEC1143

who carried out the Project work under my supervision and guidance.

Prof. Thomas Abraham J V

Assistant Professor

School of Computing Science and Engineering (SCSE),

VIT University, Chennai

Chennai – 600 127.

ABSTRACT

In this project-centered course we will build a modern software hierarchy, designed to enable the translation and execution of object-based, high-level languages on a bare-bone computer hardware platform. In particular, we will implement a virtual machine and a compiler for a simple, Java-like programming language, and we will develop a basic operating system that closes gaps between the high-level language and the underlying hardware platform. In the process, we will gain a deep, hands-on understanding of numerous topics in applied computer science, e.g. stack processing, parsing, code generation, and classical algorithms and data structures for memory management, vector graphics, input-output handling, and various other topics that lie at the very core of every modern computer system.

ACKNOWLEDGEMENT

We wish to express our sincere thanks and deep sense of gratitude to our project guide, **Prof. Thomas Abraham J V,** Assistant Professor, School of Computing Science and Engineering, for his consistent encouragement and valuable guidance offered to us in a pleasant manner throughout the course of the project work.

We are extremely grateful to **Dr. Vaidehi Vijayakumar**, Professor, Dean of the School of Computing Science and Engineering, VIT Chennai, for extending the facilities of the School towards our project and for her unstinting support.

We express our thanks to our Programme Chair **Dr. Rajesh Kanna B**, Associate Professor for his support throughout the course of this project.

We also take this opportunity to thank all the faculty of the School for their support and their wisdom imparted to us throughout the course.

We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoing this course in such a prestigious institution.

ROHIT **TARUNIKA** **SHRUTI**

INTRODUCTION :

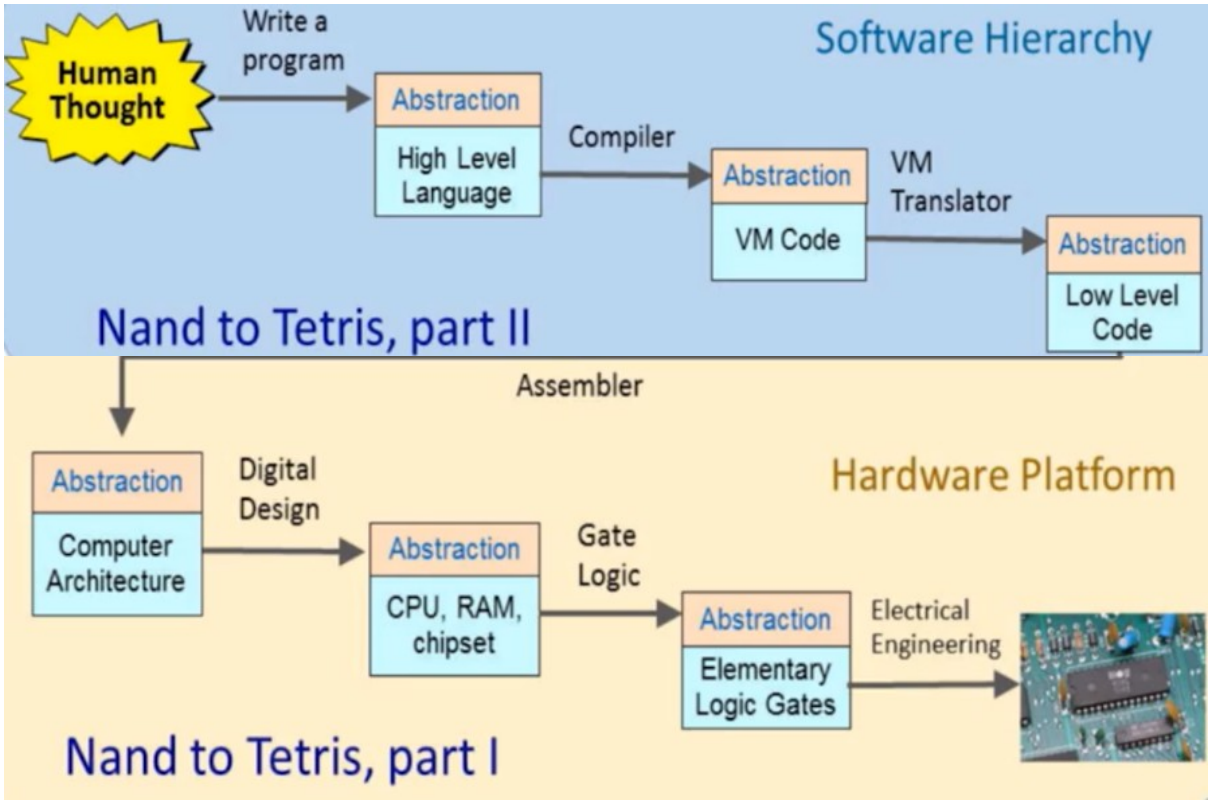
An operating system is a collection of software services designed to close gaps between high-level programs and the underlying hardware on which they run.

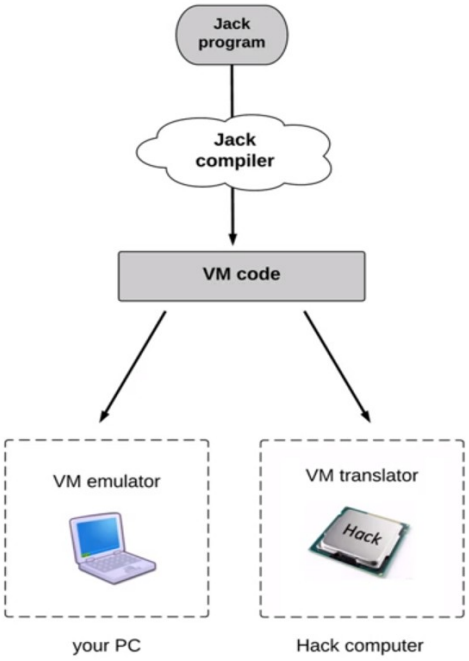
In particular, operating systems provide numerous low-level services such as accessing the computer's RAM, keyboard, and screen. In addition, a typical operating systems provides libraries for common mathematical operations, string processing operations, and more.

Modern languages like Java and Python are deployed together with a standard class libraries that implement many such OS services. In this module we'll develop a basic OS that will be packaged in a similar set of class libraries. The OS will be developed in Jack, using a bootstrapping strategy, similar to how Linux was developed in C.

Operating system code must be highly efficient. With that in mind, we'll devote a considerable part of this module for presenting elegant and efficient algorithms. Taken together, these algorithms form a cool display of computer science gems. Learning and implementing these algorithms in the context of an OS will be a nice way to celebrate the end of your Nand to Tetris journey.

We are mainly focusing on the software development on top a hack computer (computer with basic functionalities built from the ground up). We use a programming language called JACK programming language to build an operating system called JACK operating system. We also build a vm translator and compiler in the process.





1.VM TRANSLATOR :

In computing, a **virtual machine** (**VM**) is an emulation of a computer system. Virtual machines are based on computer architectures and provide functionality of a physical computer. Their implementations may involve specialized hardware, software, or a combination.

The VM code is written in the VM language and the VM language consists of four categories of commands So we're going to develop a basic VM translator that is capable of translating VM programs that include arithmetic, logical, and memory access commands into machine language. It translates the VM code to hack assembly code.

Proposed design:

- `Parser`: parses each VM command into its lexical elements
- `CodeWriter`: writes the assembly code that implements the parsed command
- `Main`: drives the process (`VMTranslator`)

`Main` (`VMTranslator`)

Input: *fileName*`.vm`

Output: *fileName*`.asm`

All the codes for the translator have been written in Java programming language.

1.Codewriter.java:

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;


public class CodeWriter {

    private int arthJumpFlag;
    private PrintWriter outPrinter;

     public CodeWriter(File fileOut) {

        try {

            outPrinter = new PrintWriter(fileOut);
            arthJumpFlag = 0;

        } catch (FileNotFoundException e) {

            e.printStackTrace();

        }

    }


    public void setFileName(File fileOut){

    }

    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(pushTemplate1(String.valueOf(16 + index),index,true));
        }
    }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(popTemplate1(String.valueOf(16 + index),index,true));
        }
    }else {
        throw new IllegalArgumentException("Call writePushPop() for a non-pushpop command");
    }
}
```

```
    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){

        //When it is a pointer, just read the data stored in THIS or THAT
        //When it is static, just read the data stored in that address
        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){
```

```
        //When it is a pointer R13 will store the address of THIS or THAT
        //When it is a static R13 will store the index address
        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";

    }

}
```

2.Parser.java:

```
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");arit
hmeticCmds.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")){
        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 "";

        }
    }
}
```

3.Vmtranslator.java:

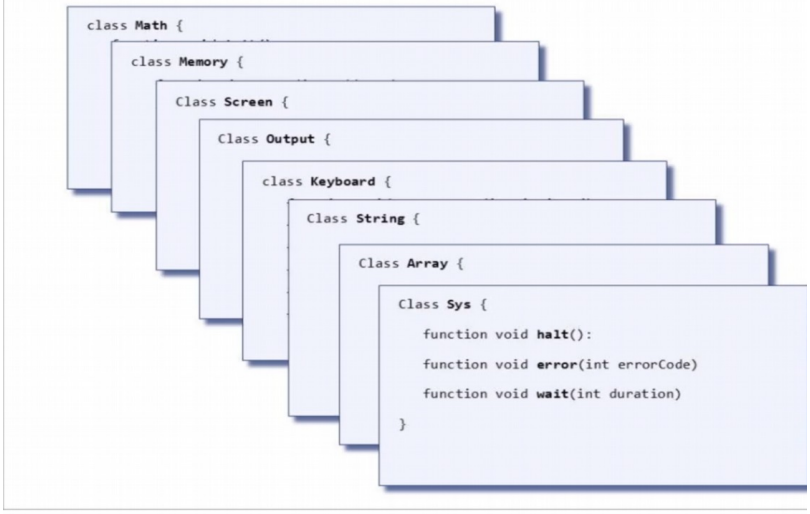
```
import java.io.File;
import java.util.ArrayList;
import java.util.HashMap;

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 {
        File fileIn = new File(args[0]);
        String fileOutPath = "";
        File fileOut;
        CodeWriter writer;
        ArrayList<File> vmFiles = new ArrayList<File>();
        if (fileIn.isFile()) {
            //if it is a single file, see whether it is a vm file
            String path = fileIn.getAbsolutePath();
            if (!Parser.getExt(path).equals(".vm")) {
                throw new IllegalArgumentException(".vm file is required!");
            }
            vmFiles.add(fileIn);
            fileOutPath = fileIn.getAbsolutePath().substring(0, fileIn.getAbsolutePath().lastIndexOf(".")) + ".asm";
        } else if (fileIn.isDirectory()) {
            //if it is a directory get all vm files under this directory
            vmFiles = getVMFiles(fileIn);
```

```
            //if no vn file in this directory
            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);

        for (File f : vmFiles) {

            Parser parser = new Parser(f);

            int type = -1;

            //start parsing
            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());

                }

            }

        }

        //save file
        writer.close();

        System.out.println("File created : " + fileOutPath);
    }
  }

}
```

2.JACK OPERATING SYSTEM :

Jack Operating System is developed using Jack Programming language. It has 8 main classes :



1.MATH

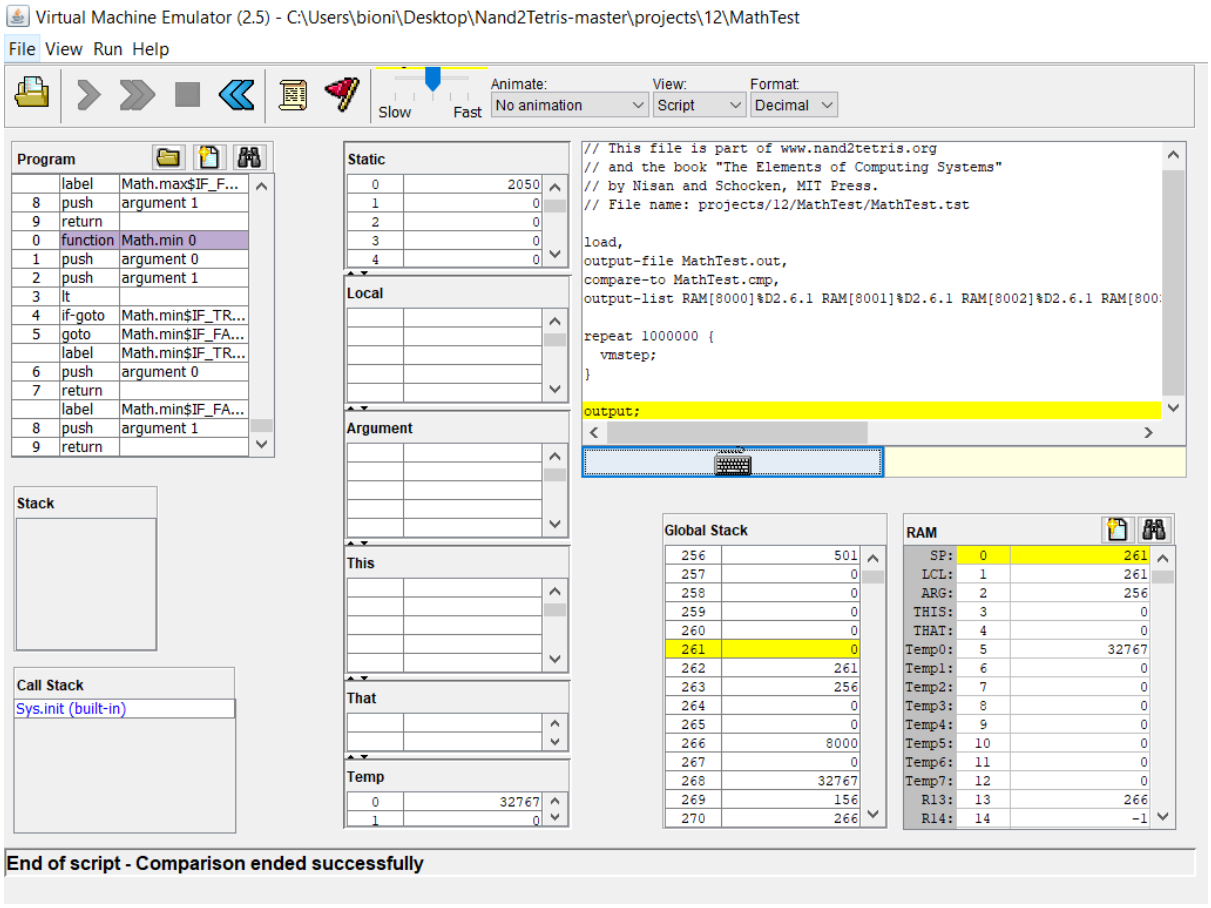
```
/**
 * A basic math library.
 */
class Math {

    static Array bitArray;

    /** Initializes the library. */
    function void init() {
        let bitArray = Array.new(16);
        let bitArray[0] = 1;
        let bitArray[1] = 2;
        let bitArray[2] = 4;
        let bitArray[3] = 8;
        let bitArray[4] = 16;
        let bitArray[5] = 32;
        let bitArray[6] = 64;
        let bitArray[7] = 128;
        let bitArray[8] = 256;
        let bitArray[9] = 512;
        let bitArray[10] = 1024;
        let bitArray[11] = 2048;
        let bitArray[12] = 4096;
```

```
    let bitArray[13] = 8192;
    let bitArray[14] = 16384;
    let bitArray[15] = 16384 + 16384;
    return;
}

/** Returns if jth bit of x is 1 */
function boolean bit(int x, int j){
    return ~((x & bitArray[j]) = 0);
}

/** Returns the absolute value of x. */
function int abs(int x) {
    if(x < 0){
        let x = -x;
    }
    return x;
}

/** Returns the product of x and y. */
function int multiply(int x, int y) {
    var int sum, shiftedX,j;

    let sum = 0;
    let shiftedX = x;
    let j = 0;

    while(j < 16){
        if(Math.bit(y,j)){
            let sum = sum + shiftedX;
        }
        let shiftedX = shiftedX + shiftedX;
        let j = j + 1;
    }
```

```
    return sum;
}

/** Returns the integer part of x/y. */
function int divide(int x, int y) {
    var int q,result;
    var boolean pos;

    let pos = ((x < 0) = (y < 0));

    let x = Math.abs(x);
    let y = Math.abs(y);

    if(y > x){
        return 0;
    }

    let q = Math.divide(x,y + y);

    if((x - (2 * q * y)) < y){
        let result = q + q;
    }else{
        let result = q + q + 1;
    }

    if(pos){
        return result;
    }else{
        return -result;
    }

}

/** Returns the integer part of the square root of x. */
function int sqrt(int x) {
```

```
    var int y,j,temp,tempQ;

    let y = 0;
    let j = 7;

    while(~(j < 0)){
        let temp = y + bitArray[j];
        let tempQ = temp * temp;
        //avoid overflow
        if(~(tempQ > x) & (tempQ > 0)){
            let y = temp;
        }
        let j = j - 1;
    }

    return y;
}

/** Returns the greater number. */
function int max(int a, int b) {
    if(a > b){
        return a;
    }
    return b;
}

/** Returns the smaller number. */
function int min(int a, int b) {
    if(a < b){
        return a;
    }
    return b;
}

/** helper function: mod/
```

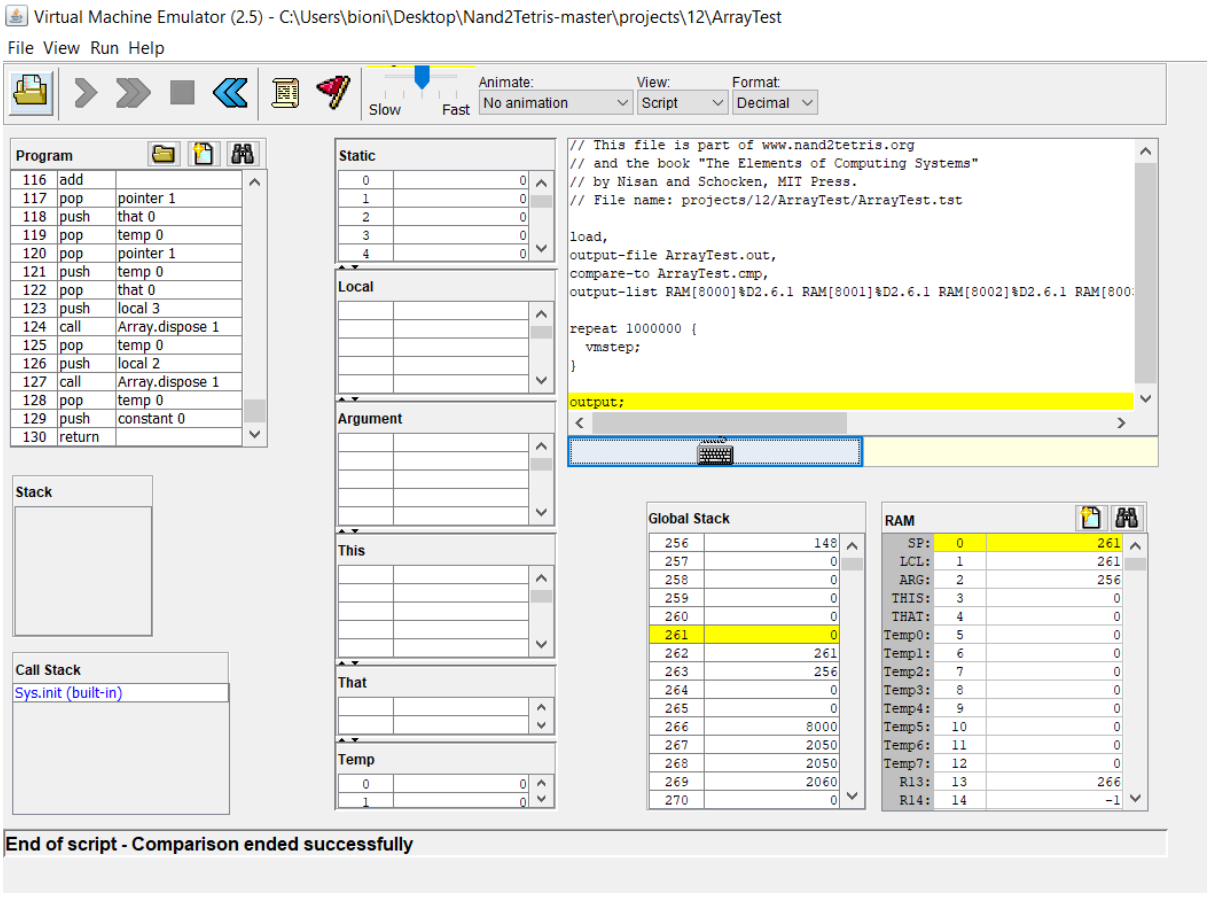
```
    function int mod(int a, int b){
        return x - (Math.divid(a,b) * b);
    }

    /** helper function two to the*/
    function int twoToThe(int i){
        return bitArray[i];
    }
}
```



2.ARRAY

```
/**
 * Represents an array. Can be used to hold any type of object.
 */
class Array {

    /** Constructs a new Array of the given size. */
    function Array new(int size) {
```

```
        return Memory.alloc(size);
    }

    /** De-allocates the array and frees its space. */
    method void dispose() {
        do Memory.deAlloc(this);
        return;
    }
}
```



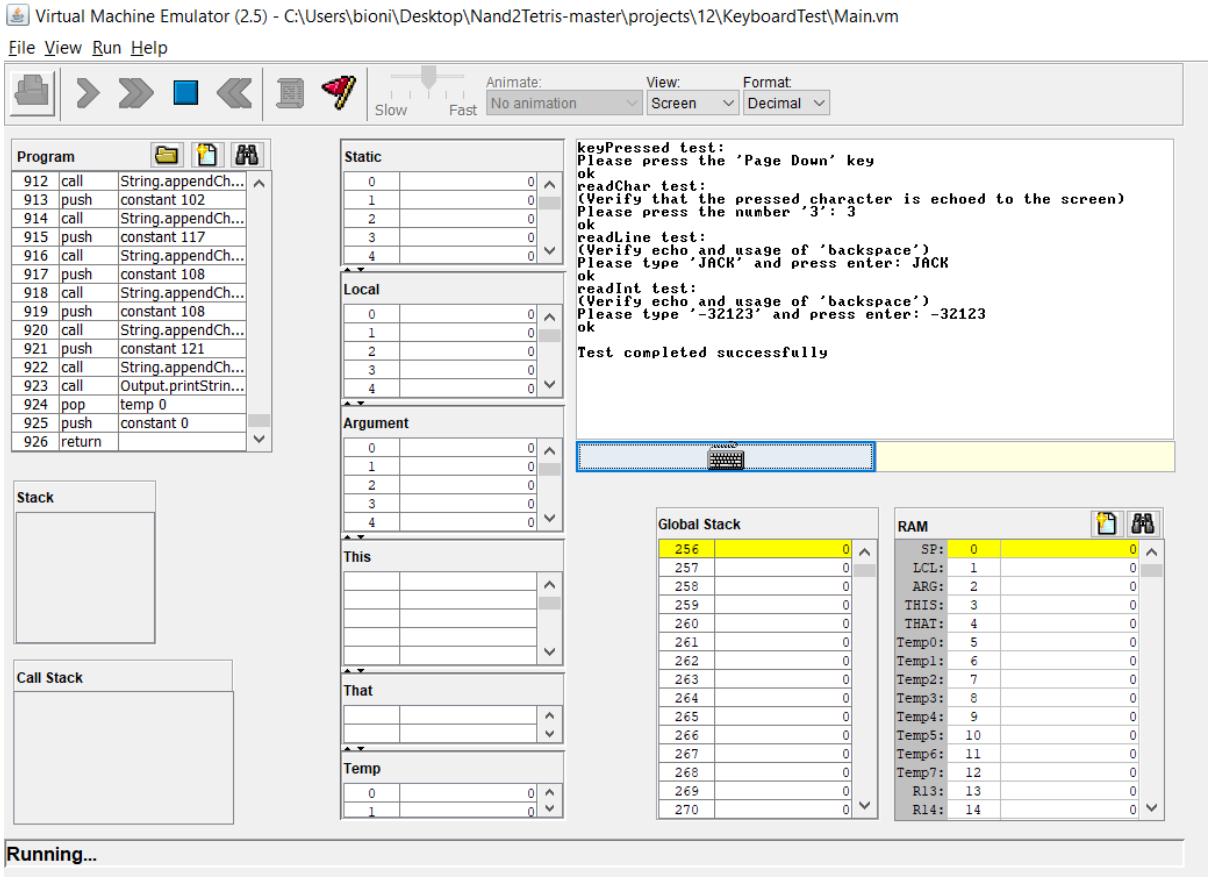
3.KEYBOARD

```
/**
 * A library for handling user input from the keyboard.
 */
class Keyboard {

    static Array keyboard;
```

```
/** Initializes the keyboard. */
function void init() {
    let keyboard = 24576;
    return;
}

/**
 * Returns the ASCII code (as char) of the currently pressed key,
 * or 0 if no key is currently pressed.
 * Recognizes all ASCII characters, as well as the following extension
 * of action keys:
 * New line = 128 = String.newline()
 * Backspace = 129 = String.backspace()
 * Left Arrow = 130
 * Up Arrow = 131
 * Right Arrow = 132
 * Down Arrow = 133
 * Home = 134
 * End = 135
 * Page Up = 136
 * Page Down = 137
 * Insert = 138
 * Delete = 139
 * ESC = 140
 * F1 - F12 = 141 - 152
 */
function char keyPressed() {
    return keyboard[0];
}

/**
 * Reads the next character from the keyboard.
 * waits until a key is pressed and then released, then echoes
 * the key to the screen, and returns the value of the pressed key.
 */
```

```
function char readChar() {
    var char key;
    while(Keyboard.keyPressed() = 0){}
    let key = Keyboard.keyPressed();
    while(~(Keyboard.keyPressed() = 0)){}
    do Output.printChar(key);
    return key;
}

/**
 * Prints the message on the screen, reads the next line
 * (until a newline character) from the keyboard, and returns its value.
 */
function String readLine(String message) {
    var String line;
    var char c;

    do Output.printString(message);

    let line = String.new(50);

    let c = Keyboard.readChar();
    while(~(c = String.newLine())){
        if(c = String.backSpace()){
            do line.eraseLastChar();
        }else{
            do line.appendChar(c);
        }
        let c = Keyboard.readChar();
    }

    return line;
}

/**
```

```
    * Prints the message on the screen, reads the next line
    * (until a newline character) from the keyboard, and returns its
    * integer value (until the first non numeric character).
    */
   function int readInt(String message) {
      var String line;

      let line = Keyboard.readLine(message);
      return line.intValue();
   }
}
```



4.STRING

```
/**
 * Represents a String object. Implements the String type.
 */
class String {
```

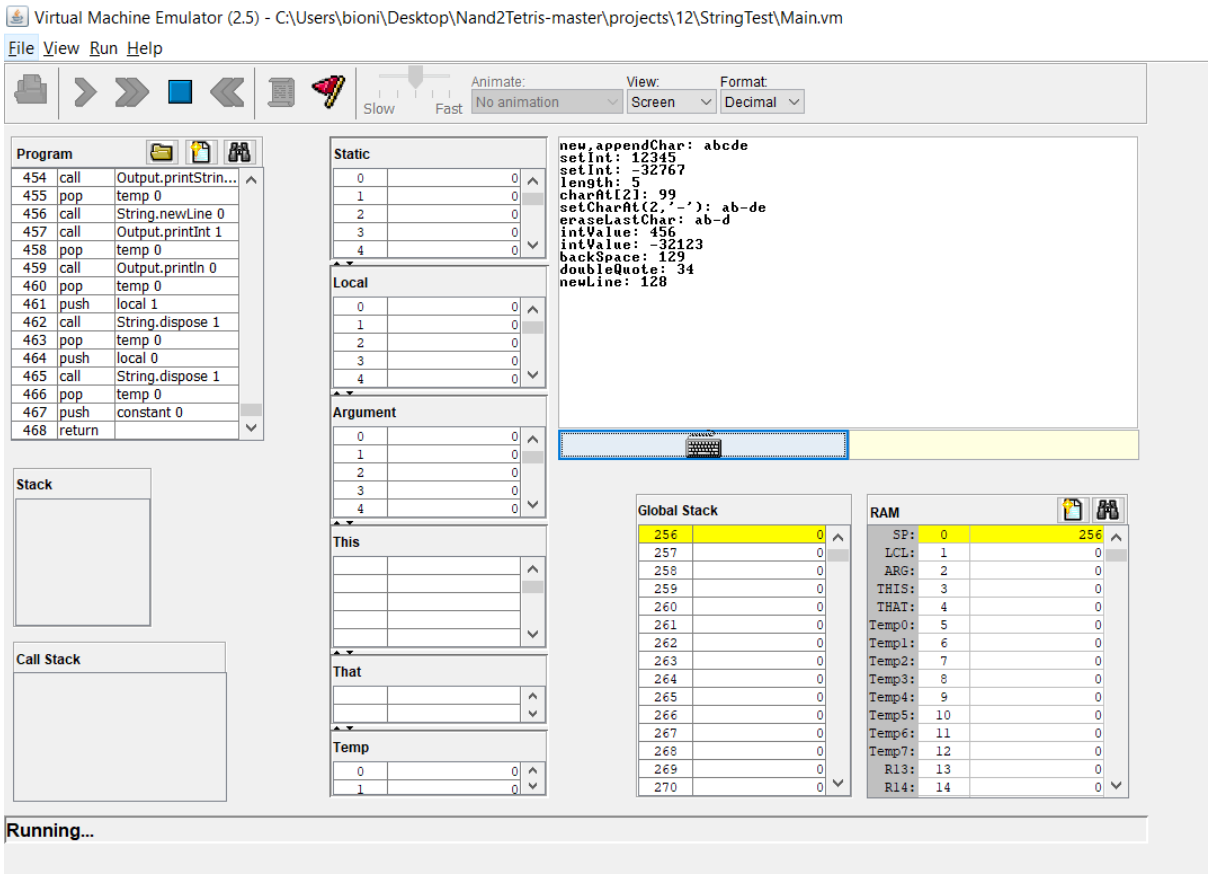
```
field int len;
field int maxLen;
field Array chars;

/** Constructs a new empty String with a maximum length of maxLength. */
constructor String new(int maxLength) {

   if(maxLength = 0){
     let maxLength = 1;
   }

   let len = 0;
   let maxLen = maxLength;
   let chars = Array.new(maxLength);
   return this;
}

/** De-allocates the string and frees its space. */
method void dispose() {
   do chars.dispose();
   return;
}

/** Returns the current length of this String. */
method int length() {
   return len;
}

/** Returns the character at location j. */
method char charAt(int j) {
   return chars[j];
}
```

```
/** Sets the j'th character of this string to be c. */
method void setCharAt(int j, char c) {
   let chars[j] = c;
   return;
}

/** Appends the character c to the end of this String.
 *  Returns this string as the return value. */
method String appendChar(char c) {
   //check if len is less than maxLen
   if(len < maxLen){
      let chars[len] = c;
      let len = len + 1;
   }
   return this;
}

/** Erases the last character from this String. */
method void eraseLastChar() {
   if(len > 0){
      let len = len - 1;
   }
   return;
}

/** Returns the integer value of this String until the first non
 *  numeric character. */
method int intValue() {
   var int intVal,index;
   var boolean neg;
```

```
    let intVal = 0;

    //check if first char is '-'
    if((len > 0) & (chars[0] = 45)){
        let neg = true;
        let index = 1;
    }else{
        let neg = false;
        let index = 0;
    }

    while((index < len) & String.isDigit(chars[index])){
        let intVal = (intVal * 10) + String.charToDigit(chars[index]);
        let index = index + 1;
    }

    if(neg){
        return -intVal;
    }else{
        return intVal;
    }

}

/** 48 <= ascii(c) <= 57 is digit*/
function boolean isDigit(char c){
    return ~(c < 48) & ~(c > 57);
}

/** must be called after String.isDigit */
function int charToDigit(char c){
    return c - 48;
```

```
}

/** must have 0<= d <= 9*/
function char digitToChar(int d){
   return d + 48;
}

/** Sets this String to hold a representation of the given number. */
method void setInt(int number) {
   //clear string first
   let len = 0;

   if(number < 0){
      let number = -number;
      do appendChar(45);
   }
   do setIntHelper(number);
   return;
}
method void setIntHelper(int number){
   var int nextNum;
   if(number < 10){
      do appendChar(String.digitToChar(number));
   }else{
      let nextNum = number / 10;
      do setIntHelper(nextNum);
      do appendChar(String.digitToChar(number - (nextNum * 10)));
   }
   return;
}
/** Returns the new line character. */
function char newLine() {
```

```
    return 128;
  }
  /** Returns the backspace character. */
  function char backSpace() {
    return 129;
  }
  /** Returns the double quote (") character. */
  function char doubleQuote() {
    return 34;
  }
}
```



5.MEMORY

```
/**
 * Memory operations library.
 */
```

```
class Memory {
    static Array memory;
    static Array freeList;
    static Array memoryBottom;

    static int heapBottom;//16384
    static int heapBase;//2048

    static int LENGTH;//the segment's length
    static int NEXT;//pointer to the next segment in the list

    static int ALLOC_LENGTH; //the alloc block's length

    /** Initializes memory parameters. */
    function void init() {

        let heapBase = 2048;
        let heapBottom = 16384;

        let memory = 0;//memory base
        let freeList = heapBase;//heap base
        let LENGTH = 0;
        let NEXT = 1;
        let freeList[LENGTH] = heapBottom - heapBase;//which is 16384 - 2048
        let freeList[NEXT] = null;

        let ALLOC_LENGTH = -1;//block[-1] store the length of alloc block

        return;
    }

    /** Returns the value of the main memory at the given address. */
```

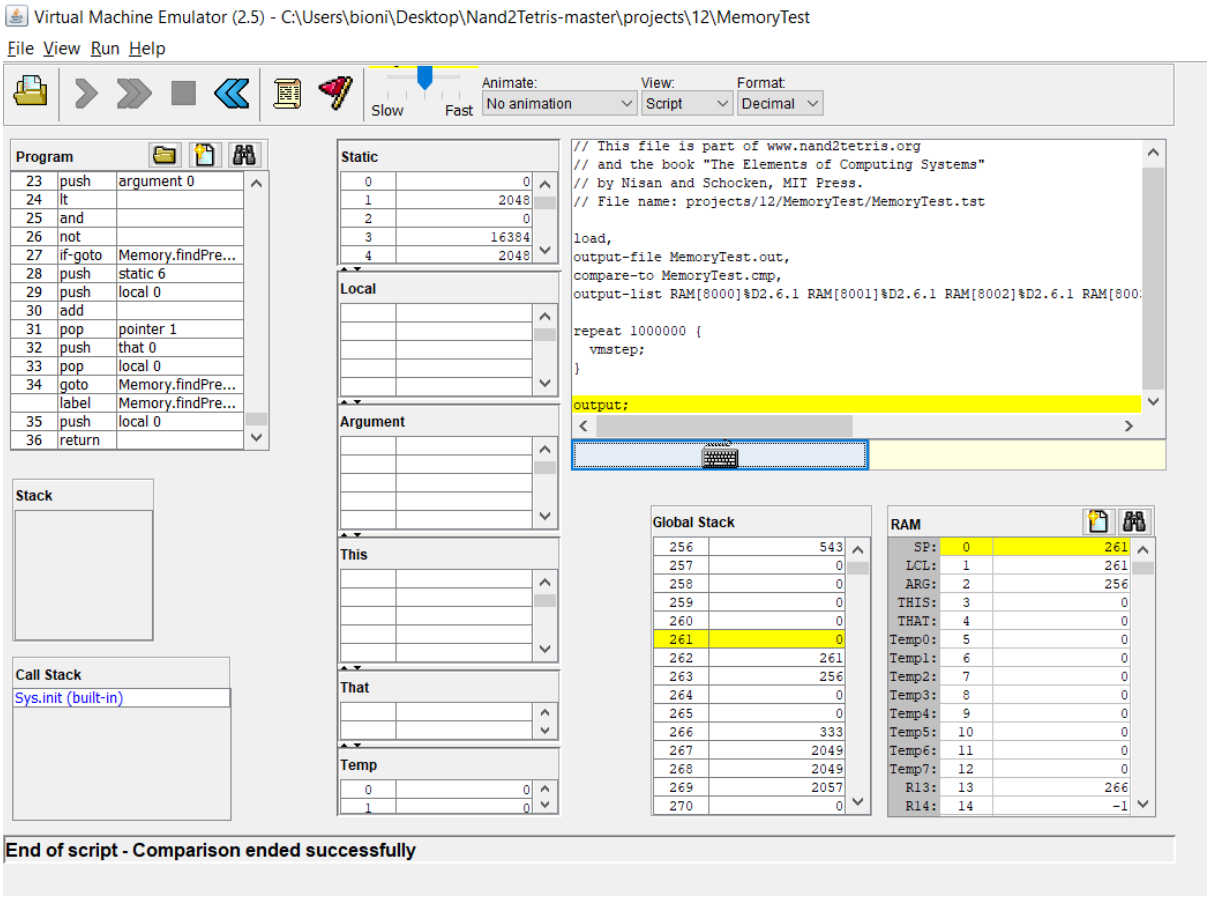
```
function int peek(int address) {
   return memory[address];
}

/** Sets the value of the main memory at this address
 *  to the given value. */
function void poke(int address, int value) {
   let memory[address] = value;
   return;
}

/**
 * start from freeList
 * keep finding next free block
 * if blockSize < bestSize and >= size then set it as bestSize
 * until next is null
 * Three cases:
 * -block found
 * -block not found, all blocks are full
 * -freeList is the block
 */
function Array bestFit(int size){
   var Array curBlock, bestBlock;
   var int bestSize,curSize;

   let bestBlock = null;//init null
   let bestSize = heapBottom - heapBase;
   let curBlock = freeList;

   if(curBlock[NEXT] = null){
      return curBlock;
   }
```

```
    while(~(curBlock = null)){

        let curSize = curBlock[LENGTH] - 1; //alloced block only need one header

        if(~(curSize < size) & (curSize < bestSize)){
            let bestBlock = curBlock;
            let bestSize = curSize;
        }
        let curBlock = curBlock[NEXT];
    }
    return bestBlock;
}
/** finds and allocates from the heap a memory block of the
 *  specified size and returns a reference to its base address. */
function int alloc(int size) {

    var Array foundBlock,nextBlock,result;

    let foundBlock = Memory.bestFit(size);
    let result = foundBlock + 1;

    if(~(foundBlock = null)){
        //alloc this block, check if only need to alloc part of this block or the whole block
        if(foundBlock[LENGTH] > (size + 3)){
            let nextBlock = foundBlock + size + 1;
            let nextBlock[NEXT] = foundBlock[NEXT];
            let nextBlock[LENGTH] = foundBlock[LENGTH] - size - 1;
            let result[ALLOC_LENGTH] = size + 1;
            let freeList = nextBlock;
        }else{
            //alloc the whole block
```

```
            let nextBlock = foundBlock[NEXT];
            let result[ALLOC_LENGTH] = foundBlock[LENGTH];
        }

        let foundBlock = nextBlock;

    }

    return result;
}

/** De-allocates the given object and frees its space. */
function void deAlloc(int object) {
    var Array preBlock,nextBlock;
    var int size;

    //if there exists pre free block, then link this block to pre block
    //else let this block be free list head and link to freelist
        //if pre block clings to this block,join two block together
        //else link together

    let size = object[ALLOC_LENGTH];
    let object = object - 1;//must consider block[-1]
    let preBlock = Memory.findPreFree(object);

    if(preBlock = null){

        let object[LENGTH] = size;
        let object[NEXT] = freeList;
        let freeList = object;

    }else{
```

```
        if((preBlock + preBlock[LENGTH]) = object){

            let preBlock[LENGTH] = preBlock[LENGTH] + size;
            let object = preBlock;
        }else{
            let object[LENGTH] = size;
            let object[NEXT] = preBlock[NEXT];
            let preBlock[NEXT] = object;
        }
    }
    //after linking, check if this block can be joined to next block
    if((object + object[LENGTH]) = object[NEXT]){
        let nextBlock = object[NEXT];
        let object[LENGTH] = object[LENGTH] + nextBlock[LENGTH];
        let object[NEXT] = nextBlock[NEXT];
    }
    return;
}
/**
 * Find previous free block
 * helper function for deAlloc
 */
function Array findPreFree(int object){
    var Array preBlock;

    //no freeBlock
    if(freeList > object){
        return null;
    }

    let preBlock = freeList;
```

```
    //preBlock.next != null && preBlock.next < object
    while(~(preBlock[NEXT] = null) & (preBlock[NEXT] < object)){
      let preBlock = preBlock[NEXT];
    }

    return preBlock;
  }
}
```



6.SCREEN

```
/**
 * Graphic screen library.
 */
class Screen {
    static Array screen;
    static boolean color;//true for black, false for white
```

```
/** Initializes the Screen. */
function void init() {
   let screen = 16384;
   let color = true;
   return;
}

/** Erases the whole screen. */
function void clearScreen() {
   var int i;
   let i = 0;
   while(i < 8192){
      let screen[i] = false;
   }
   return;
}

/** Sets the color to be used in further draw commands
 *  where white = false, black = true. */
function void setColor(boolean b) {
   let color = b;
   return;
}

/** Draws the (x, y) pixel. */
function void drawPixel(int x, int y) {
   var int address,mask;

   let address = (y * 32) + (x / 16);
   let mask = Math.twoToThe(x & 15);
```

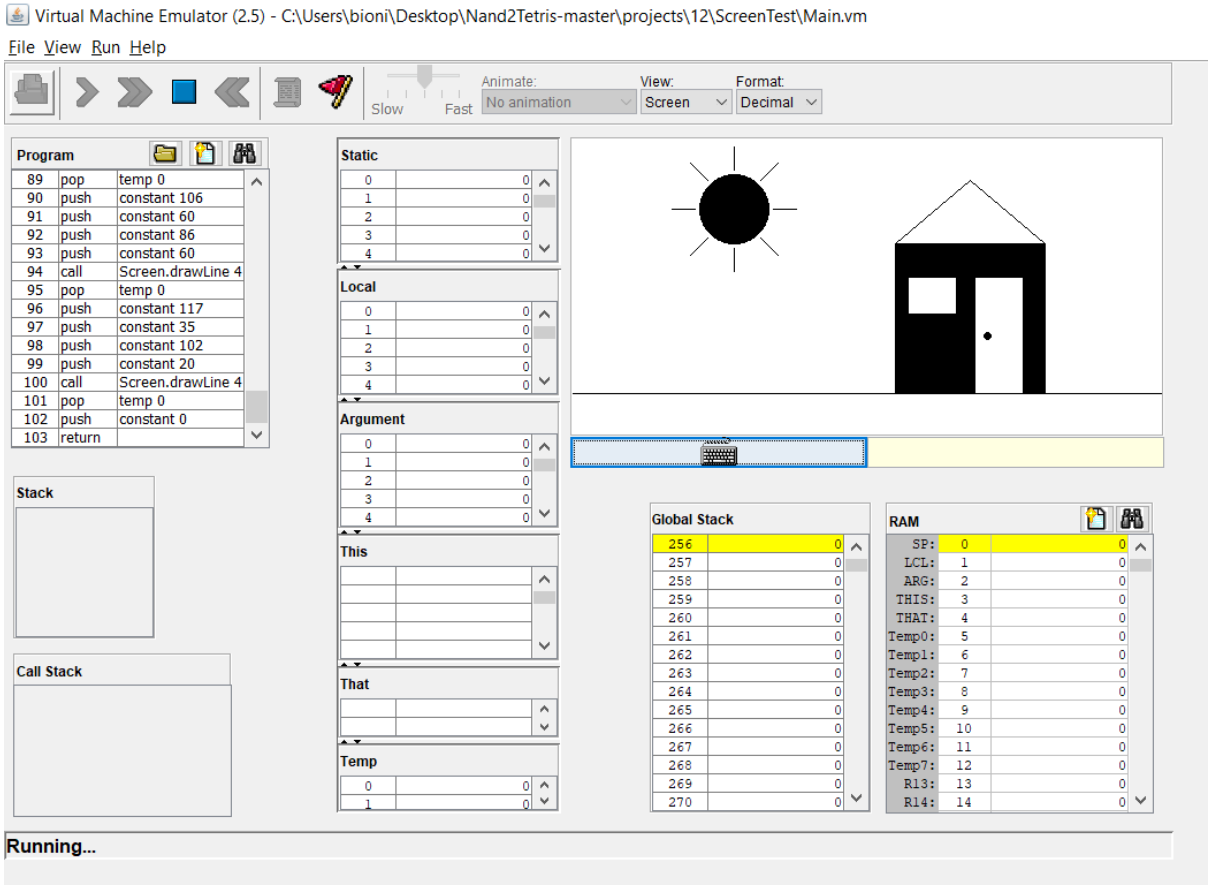
```
    if(color){
        let screen[address] = screen[address] | mask;
    }else{
        let screen[address] = screen[address] & ~mask;
    }

    return;

}

/** Draws a line from (x1, y1) to (x2, y2). */
function void drawLine(int x1, int y1, int x2, int y2) {
    var int dx, dy, a, b, temp, adyMinusbdx;

    if(x1 > x2){
        let temp = x1;
        let x1 = x2;
        let x2 = temp;

        let temp = y1;
        let y1 = y2;
        let y2 = temp;
    }

    let dx = x2 - x1;
    let dy = y2 - y1;
    let a = 0;
    let b = 0;

    if(dy = 0){

        //draw horizontal line from x1 to x2
```

```
    do Screen.drawHLine(x1, x2, y1);

  }else{

    if(dx = 0){

      //draw vertical line from y1 to y2
      do Screen.drawVLine(x1, y1, y2);

    }else{

      //draw diagonal line
      let adyMinusbdx = 0;

      if(y1 < y2){

        let a = 0;
        let b = 0;

        //x++,y++
        while(~(a > dx) & ~(b > dy)){

          do Screen.drawPixel(x1 + a, y1 + b);

          if(adyMinusbdx > 0){
            let a = a + 1;
            let adyMinusbdx = adyMinusbdx - dx;
          }else{
            let b = b + 1;
            let adyMinusbdx = adyMinusbdx + dy;
          }
```

```
            }

        }else{

            //x++,y--
            while(~(a > dx) & ~(b < dy)){

                do Screen.drawPixel(x1 + a, y1 - b);

                if(adyMinusbdx > 0){
                    let a = a + 1;
                    let adyMinusbdx = adyMinusbdx - dx;
                }else{
                    let b = b + 1;
                    let adyMinusbdx = adyMinusbdx + dy;
                }
            }
        }
    }
  }
   return;
}
//draw horizontal line
function void drawHLine(int x1, int x2, int y){
   var int temp;

   if(x1 > x2){
      let temp = x1;
      let x1 = x2;
      let x2 = temp;
   }
   while(~(x2 < x1)){
```

```
        do Screen.drawPixel(x1,y);
        let x1 = x1 + 1;
    }
    return;
}
//draw vertical line
function void drawVLine(int x, int y1, int y2){
    var int temp;
    if(y1 > y2){
        let temp = y1;
        let y1 = y2;
        let y2 = temp;
    }
    while(~(y2 < y1)){
        do Screen.drawPixel(x,y1);
        let y1 = y1 + 1;
    }
    return;
}
/** Draws a filled rectangle where the top left corner
 *  is (x1, y1) and the bottom right corner is (x2, y2). */
function void drawRectangle(int x1, int y1, int x2, int y2) {
    while( ~(y1 > y2) ) {
        do Screen.drawHLine(x1, x2, y1);
        let y1 = y1 + 1;
    }
    return;
}
/** Draws a filled circle of radius r around (cx, cy). */
function void drawCircle(int cx, int cy, int r) {
    var int dx, dy;
    var int r_squared;
```

```
        let dy = -r;
        let r_squared = r*r;
        while( ~(dy > r) ) {
          let dx = Math.sqrt(r_squared-(dy*dy));
          do Screen.drawHLine( cx-dx, cx+dx, cy+dy );
          let dy = dy + 1;
        }
        return;
    }
}
```



7.OUTPUT

```
/**
 * Handles writing characters to the screen.
 * The text screen (256 columns and 512 rows) is divided into 23 text rows (0..22),
 * each containing 64 text columns (0..63).
```

```
 * Each row is 11 pixels high (including 1 space pixel), and 8 pixels wide
 * (including 2 space pixels).
 */
class Output {

   // Character map for printing on the left of a screen word
   static Array charMaps;

   static int cursorX, cursorY;

   static Array screen;

   /** Initializes the screen and locates the cursor at the screen's top-left. */
   function void init() {
      let screen = 16384;
      let cursorX = 0;
      let cursorY = 0;
      do Output.initMap();
      return;
   }

   // Initalizes the character map array
   function void initMap() {
      var int i;

      let charMaps = Array.new(127);

      // black square (used for non printable characters)
      do Output.create(0,63,63,63,63,63,63,63,63,63,0,0);

      // Assigns the bitmap for each character in the charachter set.
      do Output.create(32,0,0,0,0,0,0,0,0,0,0,0);         //
```

```
do Output.create(33,12,30,30,30,12,12,0,12,12,0,0);  // !
do Output.create(34,54,54,20,0,0,0,0,0,0,0,0);      // "
do Output.create(35,0,18,18,63,18,18,63,18,18,0,0);  // #
do Output.create(36,12,30,51,3,30,48,51,30,12,12,0); // $
do Output.create(37,0,0,35,51,24,12,6,51,49,0,0);    // %
do Output.create(38,12,30,30,12,54,27,27,27,54,0,0); // &
do Output.create(39,12,12,6,0,0,0,0,0,0,0,0);       // '
do Output.create(40,24,12,6,6,6,6,6,12,24,0,0);     // (
do Output.create(41,6,12,24,24,24,24,24,12,6,0,0);   // )
do Output.create(42,0,0,0,51,30,63,30,51,0,0,0);     // *
do Output.create(43,0,0,0,12,12,63,12,12,0,0,0);     // +
do Output.create(44,0,0,0,0,0,0,0,12,12,6,0);       // ,
do Output.create(45,0,0,0,0,0,63,0,0,0,0,0);        // -
do Output.create(46,0,0,0,0,0,0,0,12,12,0,0);       // .
do Output.create(47,0,0,32,48,24,12,6,3,1,0,0);      // /

do Output.create(48,12,30,51,51,51,51,51,30,12,0,0); // 0
do Output.create(49,12,14,15,12,12,12,12,12,63,0,0); // 1
do Output.create(50,30,51,48,24,12,6,3,51,63,0,0);   // 2
do Output.create(51,30,51,48,48,28,48,48,51,30,0,0); // 3
do Output.create(52,16,24,28,26,25,63,24,24,60,0,0); // 4
do Output.create(53,63,3,3,3,31,48,48,48,51,30,0,0);   // 5
do Output.create(54,28,6,3,3,3,31,51,51,51,30,0,0);    // 6
do Output.create(55,63,49,48,48,24,12,12,12,12,0,0); // 7
do Output.create(56,30,51,51,51,30,51,51,51,30,0,0); // 8
do Output.create(57,30,51,51,51,62,48,48,24,14,0,0); // 9

do Output.create(58,0,0,12,12,0,0,12,12,0,0,0);      // :
do Output.create(59,0,0,12,12,0,0,12,12,6,0,0);      // ;
do Output.create(60,0,0,24,12,6,3,6,12,24,0,0);      // <
do Output.create(61,0,0,0,63,0,0,63,0,0,0,0);       // =
do Output.create(62,0,0,3,6,12,24,12,6,3,0,0);       // >
```

```
do Output.create(64,30,51,51,59,59,59,27,3,30,0,0);  // @
do Output.create(63,30,51,51,24,12,12,0,12,12,0,0);  // ?

do Output.create(65,12,30,51,51,63,51,51,51,51,0,0); // A
do Output.create(66,31,51,51,51,31,51,51,51,31,0,0); // B
do Output.create(67,28,54,35,3,3,3,35,54,28,0,0);    // C
do Output.create(68,15,27,51,51,51,51,51,27,15,0,0); // D
do Output.create(69,63,51,35,11,15,11,35,51,63,0,0); // E
do Output.create(70,63,51,35,11,15,11,3,3,3,0,0);    // F
do Output.create(71,28,54,35,3,59,51,51,54,44,0,0);  // G
do Output.create(72,51,51,51,51,63,51,51,51,51,0,0); // H
do Output.create(73,30,12,12,12,12,12,12,12,30,0,0); // I
do Output.create(74,60,24,24,24,24,24,27,27,14,0,0); // J
do Output.create(75,51,51,51,27,15,27,51,51,51,0,0); // K
do Output.create(76,3,3,3,3,3,3,35,51,63,0,0);       // L
do Output.create(77,33,51,63,63,51,51,51,51,51,0,0); // M
do Output.create(78,51,51,55,55,63,59,59,51,51,0,0); // N
do Output.create(79,30,51,51,51,51,51,51,51,30,0,0); // O
do Output.create(80,31,51,51,51,31,3,3,3,3,0,0);     // P
do Output.create(81,30,51,51,51,51,51,63,59,30,48,0);// Q
do Output.create(82,31,51,51,51,31,27,51,51,51,0,0); // R
do Output.create(83,30,51,51,6,28,48,51,51,30,0,0);  // S
do Output.create(84,63,63,45,12,12,12,12,12,30,0,0); // T
do Output.create(85,51,51,51,51,51,51,51,51,30,0,0); // U
do Output.create(86,51,51,51,51,51,30,30,12,12,0,0); // V
do Output.create(87,51,51,51,51,51,63,63,63,18,0,0); // W
do Output.create(88,51,51,30,30,12,30,30,51,51,0,0); // X
do Output.create(89,51,51,51,51,30,12,12,12,30,0,0); // Y
do Output.create(90,63,51,49,24,12,6,35,51,63,0,0);  // Z

do Output.create(91,30,6,6,6,6,6,6,6,30,0,0);        // [
do Output.create(92,0,0,1,3,6,12,24,48,32,0,0);      // \
```

```
do Output.create(93,30,24,24,24,24,24,24,24,30,0,0);   // ]
do Output.create(94,8,28,54,0,0,0,0,0,0,0,0);         // ^
do Output.create(95,0,0,0,0,0,0,0,0,0,63,0);          // _
do Output.create(96,6,12,24,0,0,0,0,0,0,0,0);         // `

do Output.create(97,0,0,0,14,24,30,27,27,54,0,0);     // a
do Output.create(98,3,3,3,15,27,51,51,51,30,0,0);     // b
do Output.create(99,0,0,0,30,51,3,3,51,30,0,0);       // c
do Output.create(100,48,48,48,60,54,51,51,51,30,0,0);  // d
do Output.create(101,0,0,0,30,51,63,3,51,30,0,0);     // e
do Output.create(102,28,54,38,6,15,6,6,6,15,0,0);     // f
do Output.create(103,0,0,30,51,51,51,62,48,51,30,0);   // g
do Output.create(104,3,3,3,27,55,51,51,51,51,0,0);    // h
do Output.create(105,12,12,0,14,12,12,12,12,30,0,0);   // i
do Output.create(106,48,48,0,56,48,48,48,48,51,30,0);  // j
do Output.create(107,3,3,3,51,27,15,15,27,51,0,0);    // k
do Output.create(108,14,12,12,12,12,12,12,12,30,0,0);  // l
do Output.create(109,0,0,0,29,63,43,43,43,43,0,0);    // m
do Output.create(110,0,0,0,29,51,51,51,51,51,0,0);    // n
do Output.create(111,0,0,0,30,51,51,51,51,30,0,0);    // o
do Output.create(112,0,0,0,30,51,51,51,31,3,3,0);     // p
do Output.create(113,0,0,0,30,51,51,51,62,48,48,0);   // q
do Output.create(114,0,0,0,29,55,51,3,3,7,0,0);       // r
do Output.create(115,0,0,0,30,51,6,24,51,30,0,0);     // s
do Output.create(116,4,6,6,15,6,6,6,54,28,0,0);       // t
do Output.create(117,0,0,0,27,27,27,27,27,54,0,0);    // u
do Output.create(118,0,0,0,51,51,51,51,30,12,0,0);    // v
do Output.create(119,0,0,0,51,51,51,63,63,18,0,0);    // w
do Output.create(120,0,0,0,51,30,12,12,30,51,0,0);    // x
do Output.create(121,0,0,0,51,51,51,62,48,24,15,0);   // y
do Output.create(122,0,0,0,63,27,12,6,51,63,0,0);     // z
```

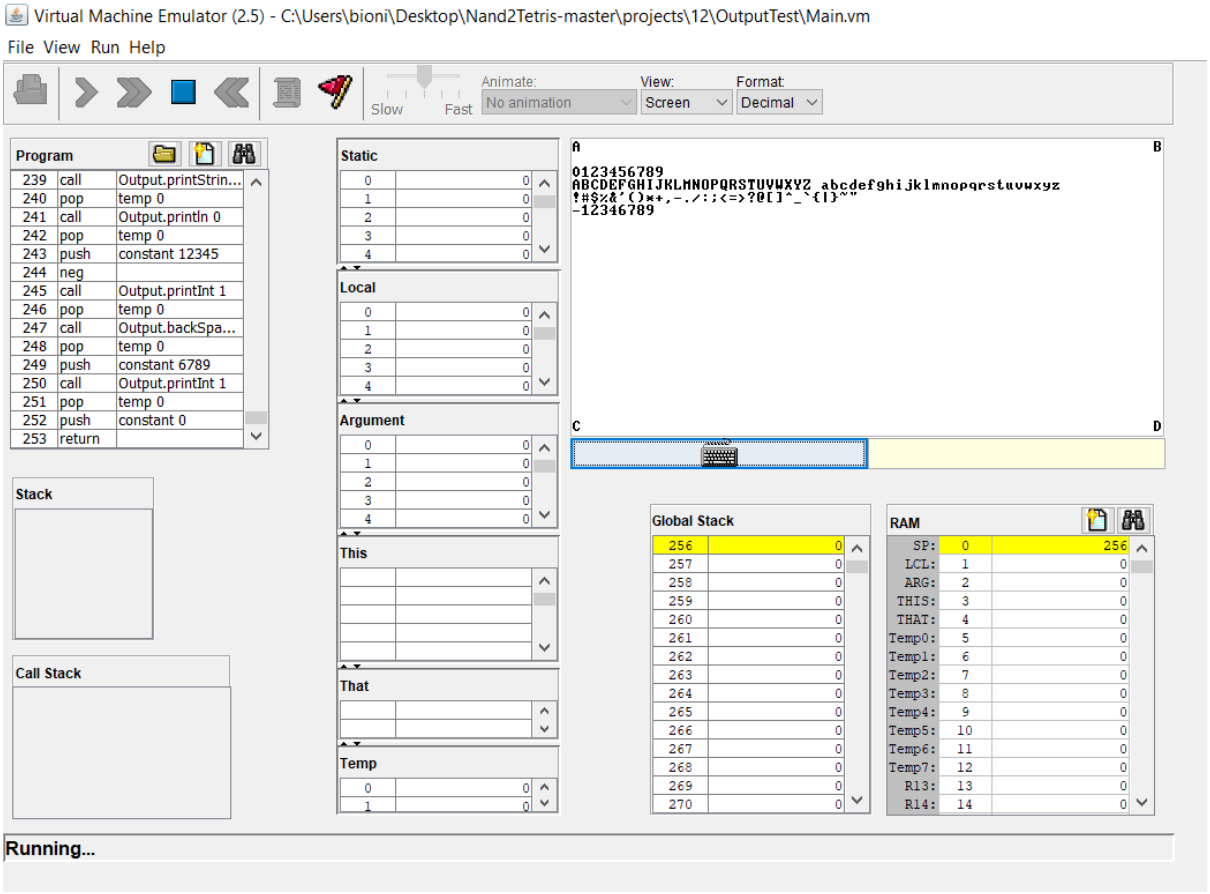
```
    do Output.create(123,56,12,12,12,7,12,12,12,56,0,0);   // {
    do Output.create(124,12,12,12,12,12,12,12,12,12,0,0);  // |
    do Output.create(125,7,12,12,12,56,12,12,12,7,0,0);    // }
    do Output.create(126,38,45,25,0,0,0,0,0,0,0,0);        // ~

    return;
}

// Creates a character map array of the given char index with the given values.
function void create(int index, int a, int b, int c, int d, int e,
                     int f, int g, int h, int i, int j, int k) {
    var Array map;

    let map = Array.new(11);
    let charMaps[index] = map;

    let map[0] = a;
    let map[1] = b;
    let map[2] = c;
    let map[3] = d;
    let map[4] = e;
    let map[5] = f;
    let map[6] = g;
    let map[7] = h;
    let map[8] = i;
    let map[9] = j;
    let map[10] = k;

    return;
}

// Returns the character map (array of size 11) for the given character
```

```
// If an invalid character is given, returns the character map of a black square.
function Array getMap(char c) {

   if ((c < 32) | (c > 126)) {
      let c = 0;
   }

   return charMaps[c];
}

/** Moves the cursor to the j❼th column of the i❼th row,
 *  and erases the character that was there. */
function void moveCursor(int i, int j) {
   let cursorX = j;
   let cursorY = i;
   return;
}

/** Prints c at the cursor location and advances the cursor one
 *  column forward. */
//reference:https://github.com/havivha/Nand2Tetris/blob/master/12/Output.jack
function void printChar(char c) {
   var Array map;
   var int address,mask,bitmap,i;

   // Get the character bitmap
   let map = Output.getMap(c);
   let address = (cursorY*32*11) + (cursorX/2);
   let mask = cursorX & 1;

   // Print the character
   let i = 0;
```

```
    while( i < 11 ) {
        let bitmap = map[i];
        if( mask = 1 ) {
            let bitmap = bitmap * 256;
            let screen[address] = screen[address] & (-1 & 255) | bitmap;
        }else{
            let screen[address] = screen[address] & 255 | bitmap;
        }

        let address = address + 32;
        let i = i + 1;
    }

    // Advance the cursor
    if( cursorX = 63 ) {
        do Output.println();
    }
    else {
        do Output.moveCursor(cursorY,cursorX+1);
    }

    return;
}

/** Prints s starting at the cursor location, and advances the
 *  cursor appropriately. */
function void printString(String s) {
    var int i;
    let i = 0;
    while( i < s.length() ) {
        do Output.printChar(s.charAt(i));
        let i = i + 1;
```

```
    }
    return;
}

/** Prints i starting at the cursor location, and advances the
 *  cursor appropriately. */
function void printInt(int i) {
    var String s;
    let s = String.new(10);
    do s.setInt(i);
    do Output.printString(s);
    do s.dispose();
    return;
}

/** Advances the cursor to the beginning of the next line. */
function void println() {
    if( cursorY < 22 ) {
        do Output.moveCursor(cursorY+1,0);
    }else {
        do Output.moveCursor(0,0);
    }
    return;
}



/** Moves the cursor one column back. */
function void backSpace() {
    if( cursorX = 0 ) {
        if( ~(cursorY = 0) ) {
            do Output.moveCursor((cursorY-1),63);
        }
```

```
    }
    else {
       do Output.moveCursor(cursorY,(cursorX-1));
    }

    return;
  }
}
```

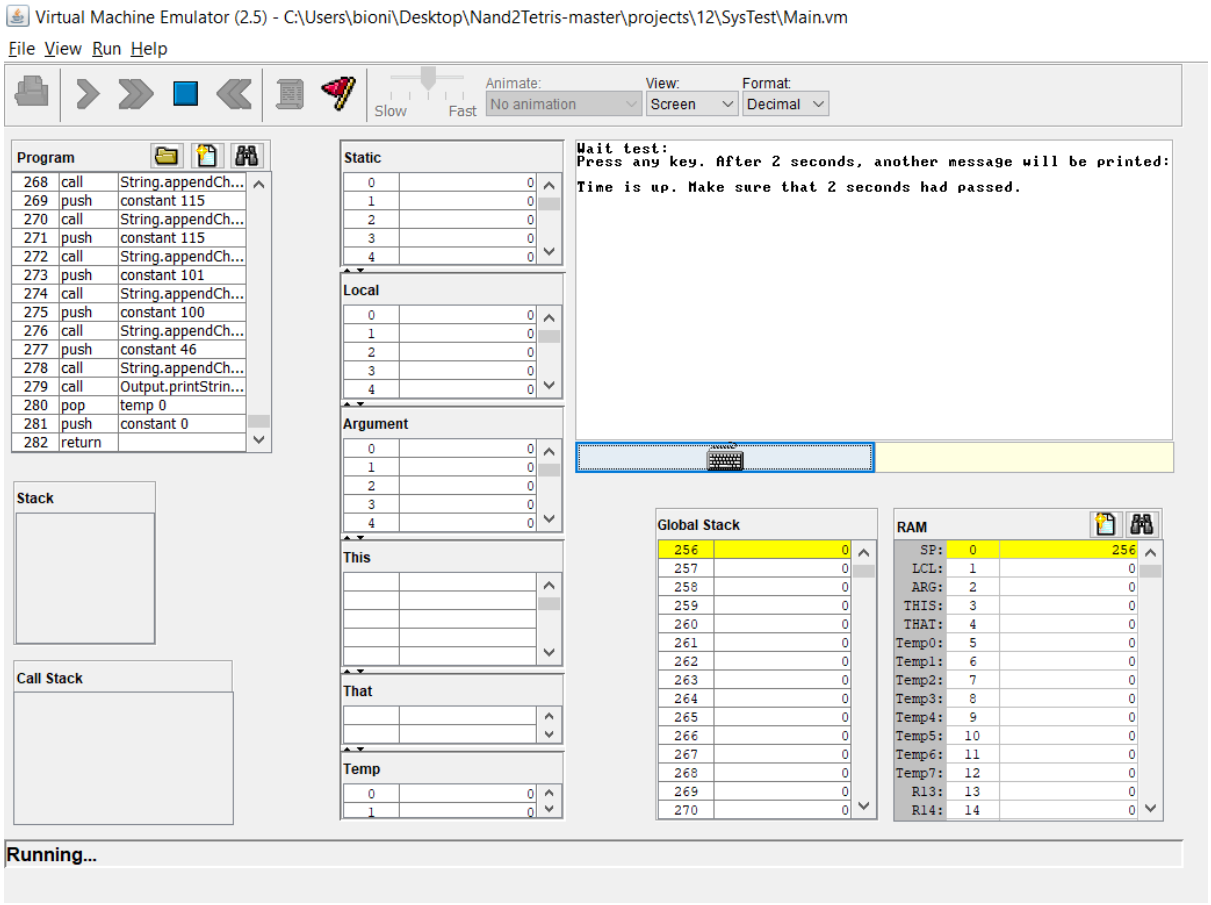


8.SYS

```
/**
 * A library of basic system services.
 */
class Sys {
```

```
/** Performs all the initializations required by the OS. */
function void init() {
   do Math.init();
   do Output.init();
   do Screen.init();
   do Keyboard.init();
   do Memory.init();
   do Main.main();
   do Sys.halt();
   return;
}

/** Halts execution. */
function void halt() {
   while(true){}
   return;
}

/** Waits approximately duration milliseconds and then returns. */
function void wait(int duration) {
   var int i, j;
   let i = 0;
   while(i < duration) {
      let j = 0;
      while(j < 100){
         let j = j + 1;
      }
      let i = i + 1;
   }
   return;
}
```

```
    /** Prints the given error code in the form "ERR<errorCode>", and halts. */
    function void error(int errorCode) {
        do Output.printString("Err");
        do Output.printInt(errCode);
        do Sys.halt();
        return;
    }
}
```



CONCLUSION:

The JACK operating system has been built and functions appropriately. All the classes work as mentioned and the implementation is successful.