
Amrita Vishwa Vidyapeetham
Centre for Excellence in Computational Engineering and Networking
Amrita School of Engineering, Coimbatore

# Playing with Strings and Arrays in Jack Programming language

**Prepared By:**
Aakash J.V.V
Rakhil M.L
R.Hema Radhika
Thanga Rohan

**Supervised By:**
Prof. Sreelakshmi K
Asst. Professor

An End Semester Project submitted to the **Centre for Excellence in Computational Engineering and Networking** in partial fulfillment of the requirements for the degree of B.tech in **Computer Engineering "Artificial Intelligence"**

July, 2022

# Acknowledgment

We are deeply thankful to **Center for Excellence in Computational Engineering and Networking (CEN) at Amrita Vishwa Vidyapeetham, Coimbatore** for providing us such a wonderful environment to peruse our research. We would like to express our sincere gratitude to **Prof. Sreelakshmi K, Asst. Professor, Department of Center for Excellence in CEN, Amrita Vishwa Vidyapeetham**. We have completed our research under her guidance. We found the research area, topic, and problem with her suggestions. She guided us with our study, and supplied us many research papers and academic resources in this area. She is patient and responsible. When we had questions and needed her help, she would always find time to meet and discuss with us no matter how busy she was.

We would also like to extend our gratitude to **Dr. Jyothish Lal G, Asst. Professor, Department of Center for Excellence in CEN, Amrita Vishwa Vidyapeetham**, who provided us with the base knowledge of this field of research, without him this research work would not exist in its present form. We would also like to acknowledge our team members for supporting each other and be grateful to our university for providing this opportunity for us. Lastly special thanks to Center for Excellence in CEN for providing this opportunity to research in this field.

Finally, we would like to extend our thanks to **Noam Nisan, Shimon Schocken and their team** for providing us wonderful tool and course named **Nand 2 Tetris**, which helped us a lot in this research work. You can find **Nand 2 Tetris** at `https://nand2tetris.org`

# Declaration

We do hereby declare that the research works presented in this project entitled, **"Playing with Strings and Arrays in Jack Programming language"** are the results of our own works. We further declare that the project has been compiled and written by us under the guidance of our supervisor. The materials that are obtained from other sources are duly acknowledged in this project.


**Authors**

Aakash J.V.V
ID: CB.EN.U4AIE21019

RAKHIL M.L
ID: CB.EN.U4AIE21048

R.Hema Radhika
ID:CB.EN.U4AIE21050

Thanga Rohan
ID: CB.EN.U4AIE21069

# Table of Contents

# Abstract

When you write code, you start by messing around with Arrays and Strings with your corresponding programming language.We looked the same way in Jack Programming Language and made few interesting application based on Strings and Array which include*Morse code*, *Tic-Tac-Toe*, *Queue*, *Stringlibrary*

# List of Tables

# List of Figures

# Definitions

**API:** API stands for *Application Programming Interface*, shows list of methods, functions and variables used in a certain program that can be used to make two programs talk to each other

**Strings:** String are a sequence of character. Jack Programming Language provides the String class to create and manipulate strings.

**Array:** *Array* is a variable which stores collection of items.

**Morse Code:** Morse code is a code that uses a series of dots and dashes to represent the different letters of alphabet or numbers.

**Queue:** Queue is a common data structure that places elements in a sequence, similar to a stack but queue uses the FIFO method (First In First Out), by which the first element that is enqueued will be the first one to be dequeued.

**Tic Tac Toe:** Tic Tac Toe (also known as Xs and Os) is a game in which two players take turns marking the spaces in a 3-by-3 grid with X or O. One wins if three of their marks are in horizontal, vertical or diagonal places. It's a draw if both cannot satisfy this condition.

.

# Chapter 1

# Introduction and Motivation

**Contents**



## 1.1 Motivation

The course *Nand to Tetris* allowed us to understand how computers work,by providing us the tool *Nand to Tetris* necessary for building a Simple computer.This tools consists of programming language named *Jack Programming Language*. The language allows you to do small projects mentioned in the course of *Nand to Tetris*. We wanted to explore this language in depth by find more possibilities of usage of Array and String.

## 1.2 Problem Statement

To fully understand the limits of the language,the peculiarities of inbuilt classes is to be understood.We tried to recreate a few inbuilt classes of Arrays and String for this purpose. While exploring any programming language, one of the basic challenges is implementing *Tic-Tac-Toe*. we also tried to implement a Data Structure named *Queue* using arrays.

# Chapter 2

# String Library

**Contents**



## 2.1 Analysing problem

### 2.1.1 Model Architecture

Functions in String library are few of inbuilt function which we have implemented in a fashionable way. These function include *Strlen, Strcmp, Strappend*. Returns the length of the String, Appends two Strings and returns the merged String, Checks if strings are equal or not respectively. These functions are written to show the internal working of the inbuilt library.
Extended class named String Split. Function contained by these class are *split* and *countwords*. where *countwords* Returns number of word in a given and *Split* Splits the String at de-lim and stores it in an array. These class is extension of the inbuilt library and which let's you split at desired character.

### 2.1.2 API

Class: String Library

functions

| Name | Parameter | Return Type | Description |
|---|---|---|---|
| Strlen | str (String) | int | Returns the length of string |
| Strappend | str1 (String) , str2 (String) | String | Appends the str2 to Str1 and returns the merged String; |
| Strcomp | str1 (String) , str2 (String) | String | Checks if strings are equal or not |

Table 2.1: String Library

The function strlen is used to access the memory where String library stores the iteration each character append used to form the string

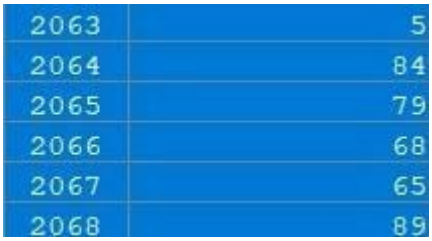


Figure 2.1: string location in global stack(string:today)

In figure 2.1 2064 to 2068 each char ASCII value of String: today and the iterations stored in 2063.

The function strcomp stores each character of given string in character Arrays and checks each character at each index.

String Extended

Class: String Split

functions

| Name | Parameter | Return Type | Description |
|---|---|---|---|
| Split | str (String), String (delim) | Array | Splits the String before de-lim and stores it in an array. |
| countWords | str (String), String (delim) | int | Returns number of word in a given String |

Table 2.2: String Extended

## 2.2 Output

### 2.2.1 Execution

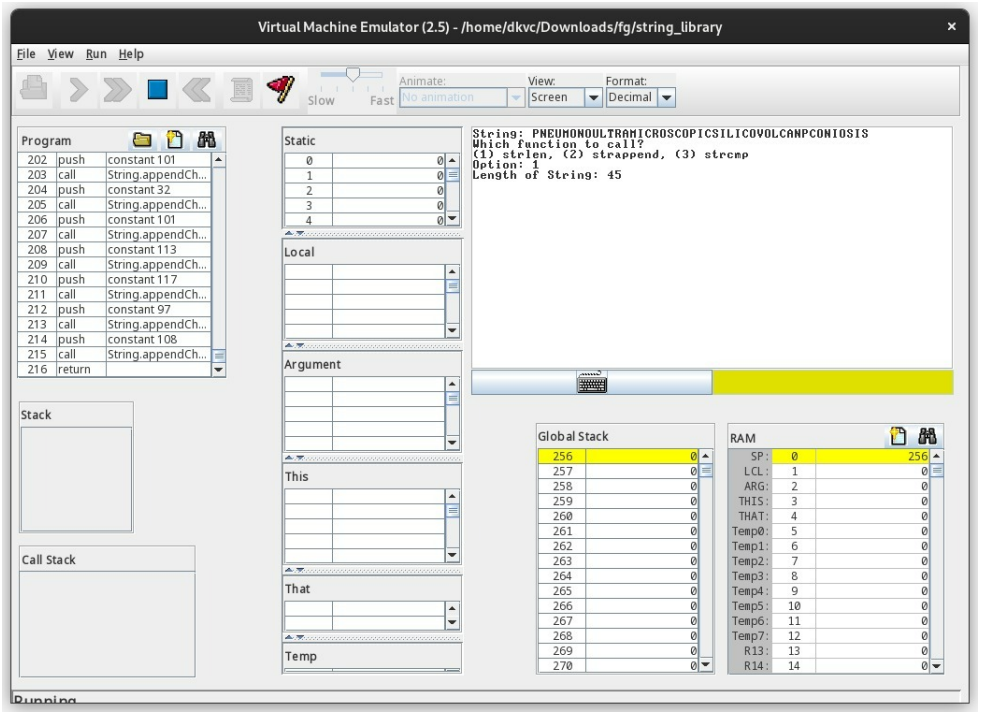


Figure 2.2: Finding the length of largest word (which is 45)

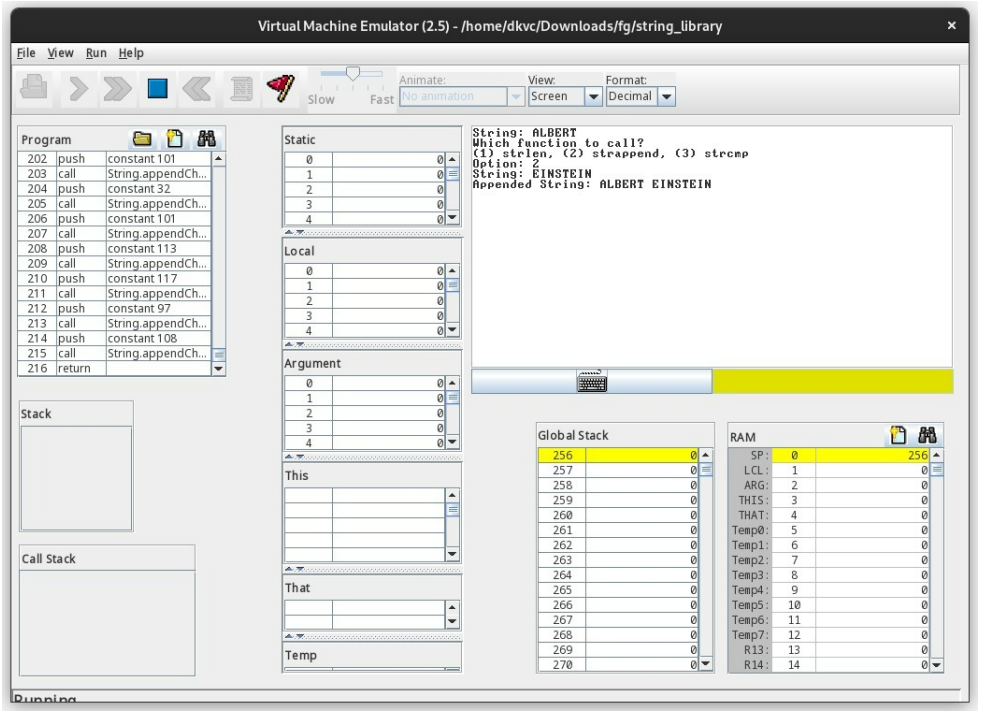


Figure 2.3: Appending first and last name of Albert Einstein

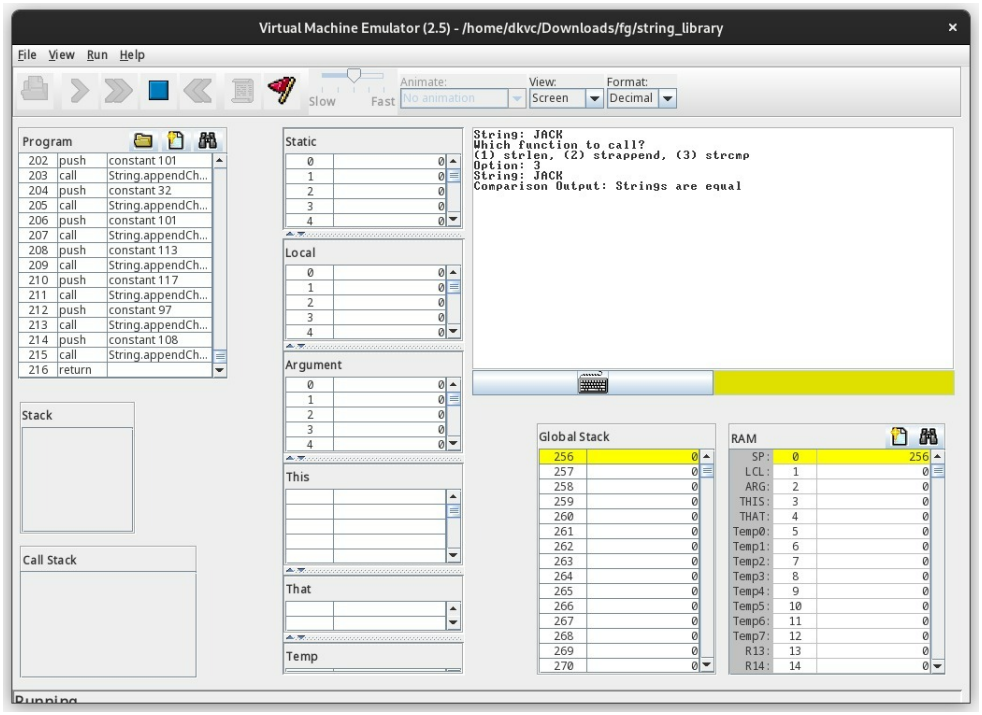


Figure 2.4: case 1: Comparing Strings(equal)

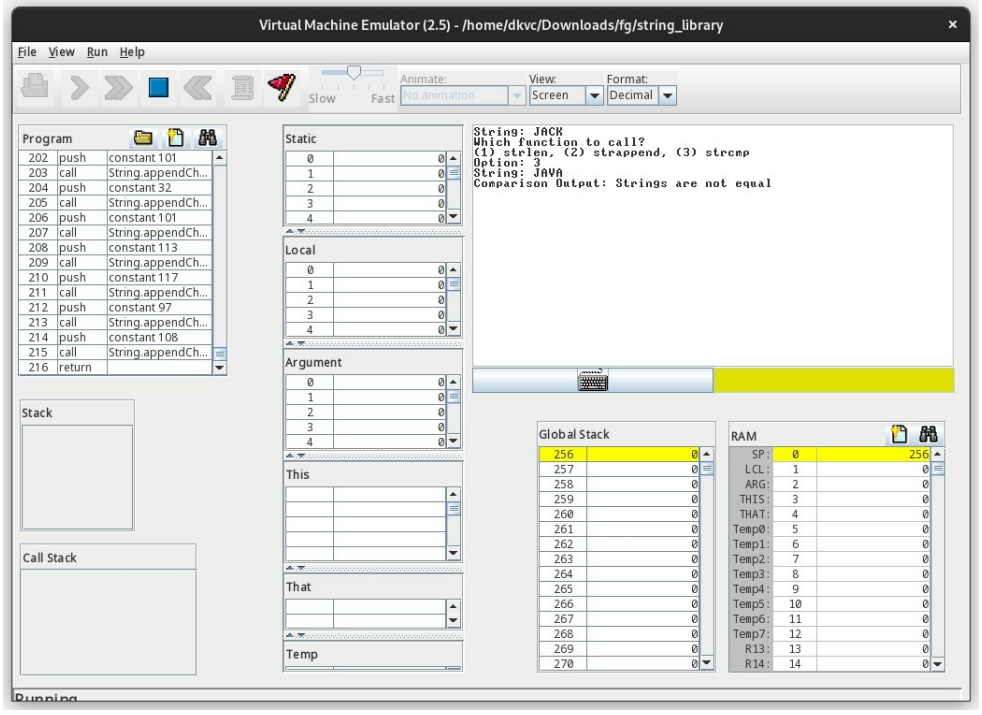


Figure 2.5: Case 2: Comparing String(non equal)

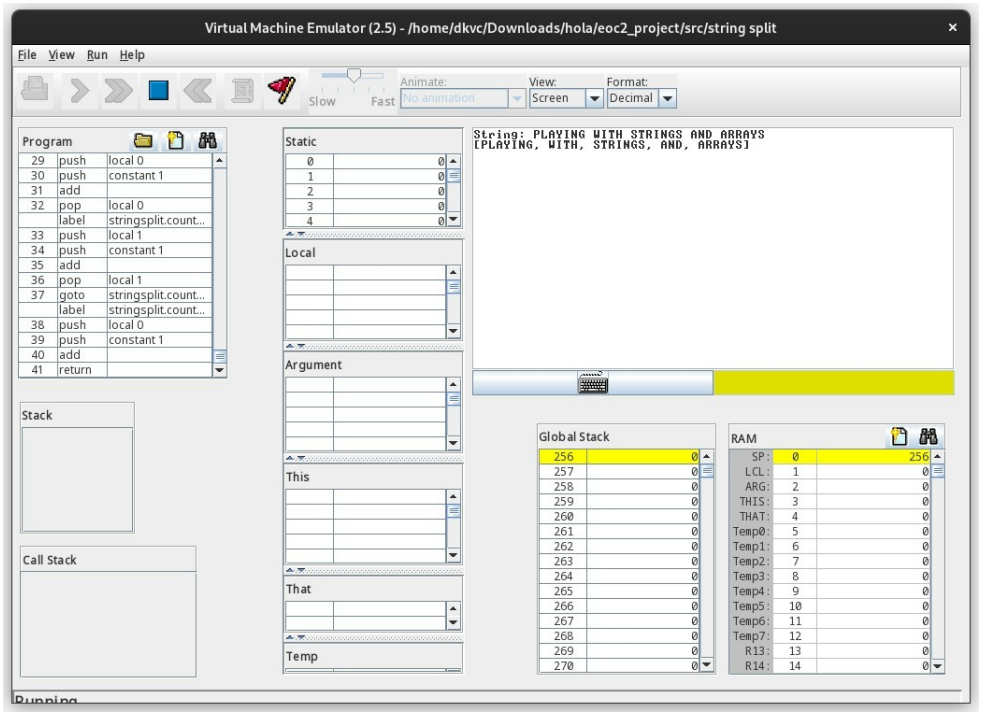


Figure 2.6: Splitting a String

# Chapter 3

# Morse Code

**Contents**



## 3.1 Analysing problem

### 3.1.1 Model Architecture

We created a class for morsecode and also create a constructor for the class in which it has variables, *eng* to store alphabets and *Morse* to store Morse code of the respective alphabet or letter. Then method *toMorse* is used to take String input and then comparing each of character to the array and print its Morse code.

### 3.1.2 API

Class: Morse code

Fields

| Name | Data Type | Description |
|---|---|---|
| eng | String | Stores alphabets |
| Morse | Array | stores the Morse code of the respective alphabet or letter |

Table 3.1: Field

Methods

| Name | Return Type | Return Type | Description |
|---|---|---|---|
| MorseCode(constructor) | None | Morse code(object) | constructs a morse code object |
| toMorse | Str(String) | void | Convert the String to Morse code |

Table 3.2: Method

## 3.2 Output

### 3.2.1 Execution

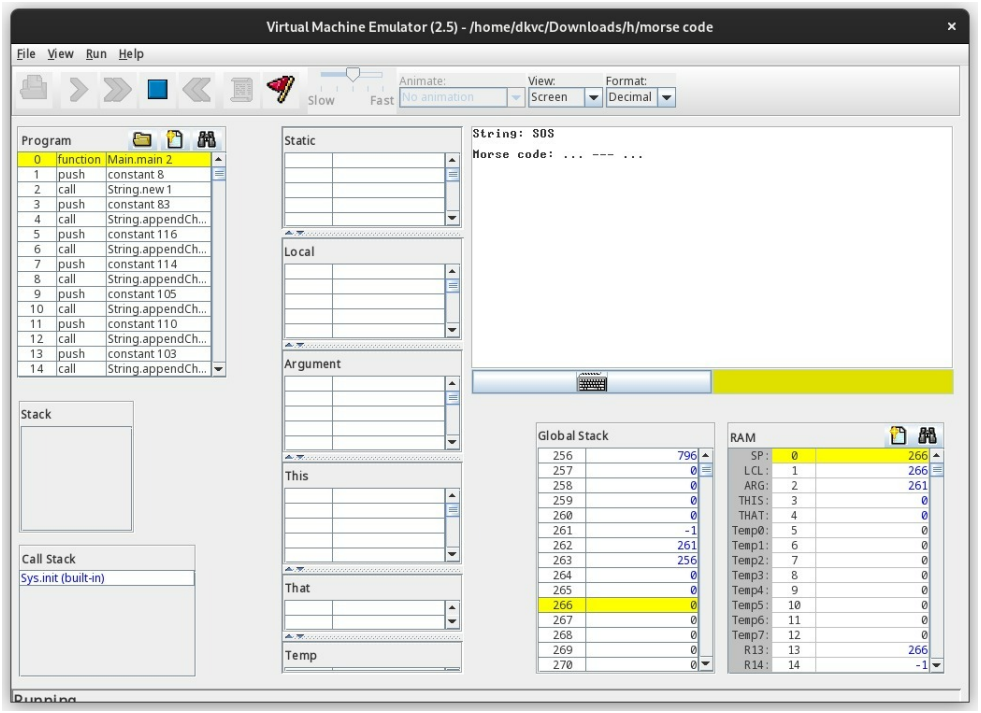


Figure 3.1: convert English to Morse

## 3.3 Application

- Morse Code is most prevalent in Aviation and Aeronautical fields since radio navigational aids such as VOR's and NDB's still identify in Morse Code.
- US Navy and Coast Guard still use signal lamps to communicate via Morse Code.
- Morse Code has also been used as an alternative form of communication for people with disabilities.

- There have been several cases where individuals whom have their abilities to communicate impaired by stroke, heart attack, or paralysis have been able to use their eyelids to communicate in Morse Code by using a series of long and quick blinks to represent that dots and dashes.

# Chapter 4

# Queue

**Contents**



## 4.1 Analysing problem

### 4.1.1 Model Architecture

We have created an Queue using Array. We have implement method of Queue *enqueue, dequeue, Front, Rear, size.* Two check methods *isEmpty* and *isFull* . A variable front is the index of the first element of the array and rear is variable which Returns the position of value last added to Queue

### 4.1.2 API

**Fields for storing values in Queue**

| Name | Data Type | Description |
|---|---|---|
| arr | Integer | Stores corresponding values of Queue |
| front | Integer | Returns the position of value First to Queue |
| rear | Integer | Returns the position of value last to Queue |
| capacity | Integer | store the length of arr |

Table 4.1: Fields for storing values in Queue

**Methods for Queue class**

| Name | Parameters | Return Type | Description |
|---|---|---|---|
| constructor | size (Integer) | Object (Queue) | Creates an instance of class Queue |
| enqueue | value (Integer) | Void | adds a value to the corresponding Queue |
| dequeue | - | Integer | Removes a value from the corresponding queue |
| capacity | - | Integer | Returns size of corresponding queue |
| Front | - | Integer | Returns value at top of Queue |
| Rear | - | Integer | Returns value at bottom of queue |
| display | - | Integer | Returns the elements of the queue |

Table 4.2: Methods for Queue class

**Check Methods for Queue class**

| Name | Parameters | Return Type | Description |
|---|---|---|---|
| isEmpty | - | Boolean | Checks whether the Queue is empty or not |
| isFull | - | Boolean | Checks whether the Queue is full or not |

Table 4.3: Check Methods for Queue class

The Check Methods *isEmpty* and *isFull* are implemented by checking whether *front* is equal to *rear*, and *capacity* is equal to *rear* respectively.

## 4.2 Output

### 4.2.1 execution

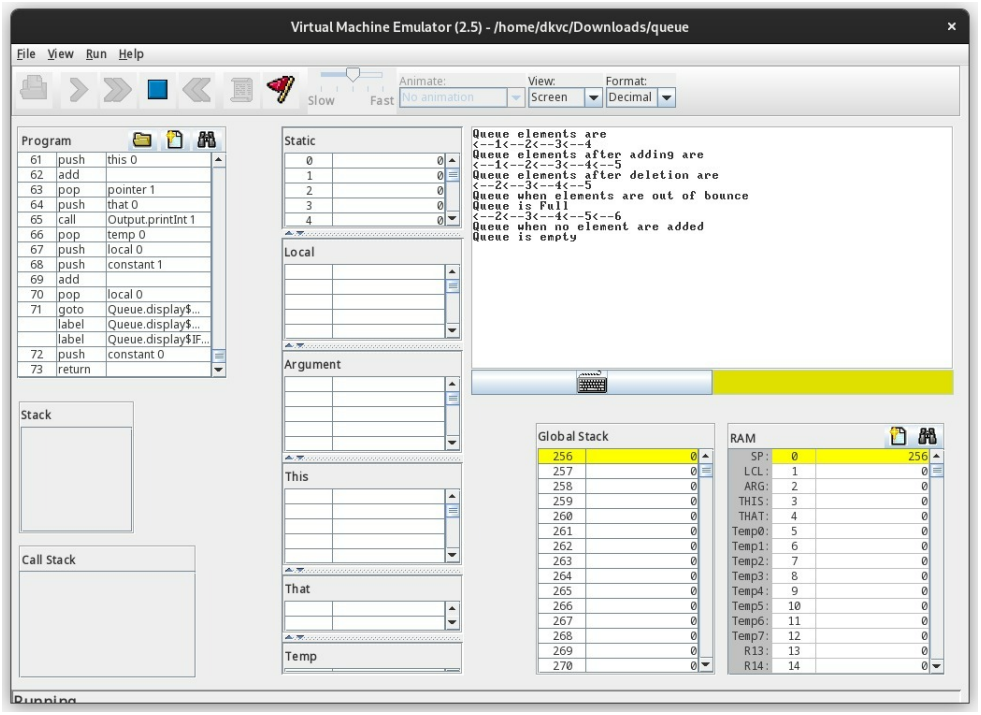


Figure 4.1: Testing queue

### 4.2.2 working

First creating a new Queue of Capacity 5. Here we are checking if the Queue work by test the four criteria.
First adding - We are adding four elements (1,2,3,4)respectfully and checking if they are added by printing the elements of Queue using Display().
Second removing - We are removing an element and checking if the first element is removed by printing the elements of Queue using Display().
Third if more elements are add to a queue - Adding 5 elements (2, 3, 4, 5, 6) respectfully to a queue of capacity 5 and adding the 7th element to check if the queue is going out of bounce.
Fourth isEmpty - Removing all the elements and checking if the string is empty.

# Chapter 5

# Tic Tac Toe

**Contents**



## 5.1 Analyzing Problem

### 5.1.1 Model Architecture

We created a class for *Tic Tac Toe*, which has variables to store current board data in a array of size 3, (which contains arrays of size 3), making it as a 3-by-3 square grid. We store current value (X or O) and current player (Player 1 or Player 2). We ask the input for position to place the corresponding value from current player. Finally, we check if player is won, or game is a draw.

```
[ [ 1, 2, 3],
  [ 4, 5, 6],
  [ 7, 8, 9] ]
```

Figure 5.1: *Tic-Tac-Toe* Grid in the form of Arrays

We analyzed the problem in 3 parts:

- **User Interface (UI):** Draws the user interface - square grid, or asking for input (position) from the player.
- **Change Interface:** Changing the UI such as changing the player, or putting corresponding value X or O after current player is done.
- **Logic:** Checking the player is won, or game is a draw.

### 5.1.2 API

**Main**

| Name | Parameter | Return Type | Description |
|---|---|---|---|
| Split | str (String), String (delim) | Array | Splits the String before de-lim and stores it in an array. |
| countWords | str (String), String (delim) | int | Returns number of word in a given String |

Table 5.1: Main(tic tac toe)

| Name | Data Type | Description |
|---|---|---|
| Board | Array of Arrays | Stores the values in the board |
| currentPlayer | Integer | Who is the current Player? |
| currentPlayerSymbol | String | What is the current Value? (X or O) |

Table 5.2: Fields for storing values in Tic Tac Toe

The field *Board* are used for creation of 3-by-3 grid (an array of size 3 which contains arrays of size 3). The fields *currentPlayer* and *currentValue* stores the current player (Player 1 or Player 2) and current value (X or O) respectively.

**Fields (Used for Comparing)**

| Name | Data Type | Description |
|---|---|---|
| blank (“ “) | String | Blank String for comparing |
| X (“X”) | String | String Letter “X” for comparing |
| O (“O”) | String | String Letter “O” for comparing |

Table 5.3: Fields used for comparing in Tic Tac Toe

The fields *Blank, X* and *O*and Function*Stringcmp* are used for comparing Strings.

## Methods (User Interface)

| Name | Parameters | Return Type | Description |
|---|---|---|---|
| TicTacToe (Constructor) | - | Object (TicTacToe) | Creates an instance of class |
| drawBoard | - | Void | Draws the Tic Tac Toe Grid |
| UI | - | Void | Draws Interface for Input and Grid |

Table 5.4: Methods for creating the User Interface in Tic Tac Toe

## Methods (Methods for Changing the User Interface)

| Name | Parameters | Return Type | Description |
|---|---|---|---|
| changeCurrentPlayer | - | Void | Changes the Current Player and value |
| setX | Position (int) | Void | Changes Value from blank to X at given position |
| setO | Position (int) | Void | Changes Value from blank to O at given position |

Table 5.5: Methods for Changing the UI in Tic Tac Toe

The methods *changeCurrentPlayer, setX* and *setO* are used to change the current player and currentValue, changing the value from blank to X, or O, respectively.

## Methods (Helper Methods for Changing the User Interface)

| Name | Parameters | Return Type | Description |
|---|---|---|---|
| currentPlayerSymbol | Position (int) | String | Returns the corresponding value at given position |
| setValue | Position (int), Value (String) | Void | Sets the given value at given position |
| remainder | Dividend (int), Divisior (int) | Integer | Returns the Remainder of given values |

Table 5.6: Helper Methods for Changing the UI in Tic Tac Toe

The method *remainder* allows us to make given values fixed within a range.
It is calculated using the Repetitive subtraction :

**while(dividend >= divisor)**
**let dividend = dividend - divisor;**

**Logical Methods for Checking the Conditions**

| Name | Parameters | Return Type | Description |
|---|---|---|---|
| checkIfWon | Void | Boolean | Check if current player is won |
| checkIfDraw | Void | Boolean | Check if the game is draw |

Table 5.7: Logical Methods for Tic Tac Toe

The method *checkIfWon* checks along horizontal, vertical and diagonal places for same strings. If they are equal, the method returns true, otherwise false. The method *checkIfDraw* checks whether all places are filled and none of the players won or not.

## 5.2 Output

### 5.2.1 Execution

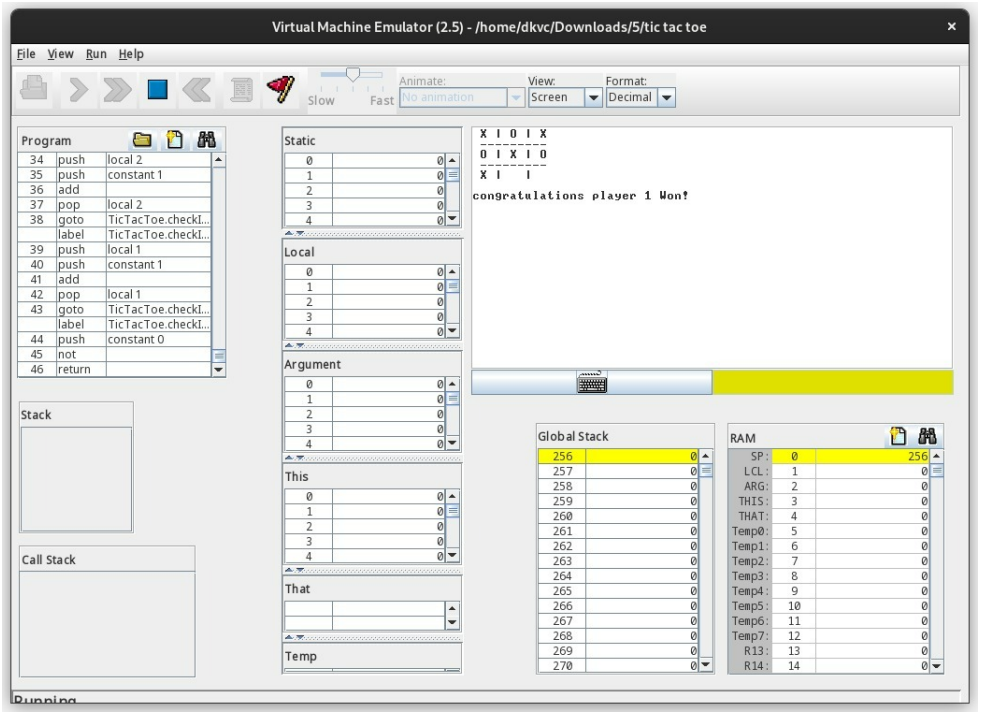


Figure 5.2: Case 1: winning of player one

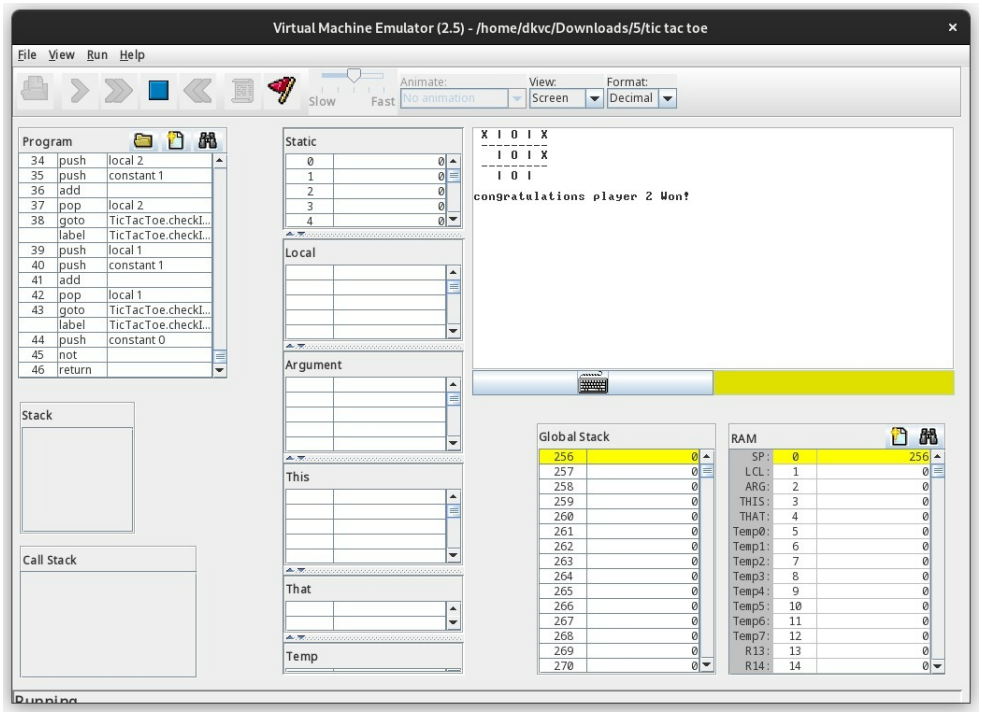


Figure 5.3: Case 2: winning of player two

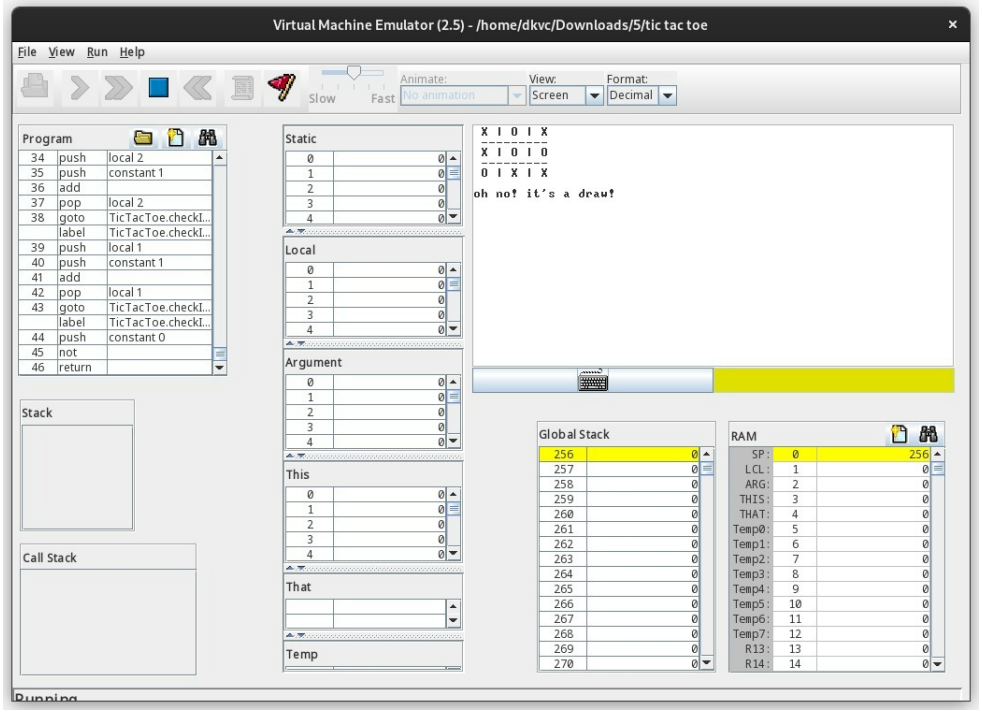


Figure 5.4: Case 3: Both players draw

# Chapter 6

# Conversion of Code from *Jack Programming Language* to *Assembly* and *Hardware Description Language (HDL)*

**Contents**



## 6.1 Conversion from *.jack* to *.vm* files

In *nand2tetris* tools suite, a tool named *Jack Compiler*, converts *.jack* files to *.vm* files. *.vm* files are executed by *VM Emulator* (another tool in *nand2tetris* suite which executes *.vm* files).

```
C:\Users\aksha\OneDrive\Desktop\nand2tetris (1)\nand2tetris\tools>JackCompiler.bat string_library
Compiling "C:\Users\aksha\OneDrive\Desktop\nand2tetris (1)\nand2tetris\tools\string_library"
```

Figure 6.1: Conversion of *.jack* files to *.vm* files

In the above figure, *Jack Compiler* is in tools directory of *nand2tetris*. The *.jack* files are stored in the code directory. *Jack Compiler* can compile *files* directly or all *files* in a certain directory.

## 6.2 Conversion from *.vm* to *.asm* files

*VM Translator* converts *.vm* files to *.asm* files. As a part of course, you are supposed to build your own *VM Translator* in any programming language. So, to hold the integrity of the course, *VM Translator* is not linked.

The *VM Translator* that was built by us has a limitation of converting only one file. In order to convert all files, you need to convert each file seperately.

```
C:\Users\RAKHIL.M.L\VM_HACK_TRANSLATOR>Python VMTranslator.py ../VM_HACK_TRANSLATOR/TicTacToe.vm
```

Figure 6.2: Conversion of *.vm* files to *.asm* files using *VM Translator*

## 6.3 Conversion from *.asm* to *.hack* files

There are two ways to convert *.asm* files to *.hack* files.

- Using own *Assembler* built in respective language
- Using *Assembler* given in *nand2tetris* tools suite

### 6.3.1 Using own *Assembler* built in respective language

*Assembler*, also known as *Hack Assembler* converts *.asm* files to *.hack* files. As a part of course, you are supposed to build your own *Assembler* in any programming language. So, to hold the integrity of the course, *Hack Assembler* is not linked.

The *Hack Assembler* that was built by us has a limitation of converting only one file. In order to convert all files, you need to convert each file separately.

```
s\tools\VM_HACK_TRANSLATOR>python Assembler.py Main.asm
```

Figure 6.3: Conversion of *.asm* files to *.hack* files using *Hack Assembler*

### 6.3.2 Using *Assembler* given in *nand2tetris* tools suite and comparison with code created by own *Assembler*

All *.asm* files are converted to *.hack* files and their comparisons are successful. Some of the outputs while comparing are shown in figures 6.4, 6.5 and 6.6.

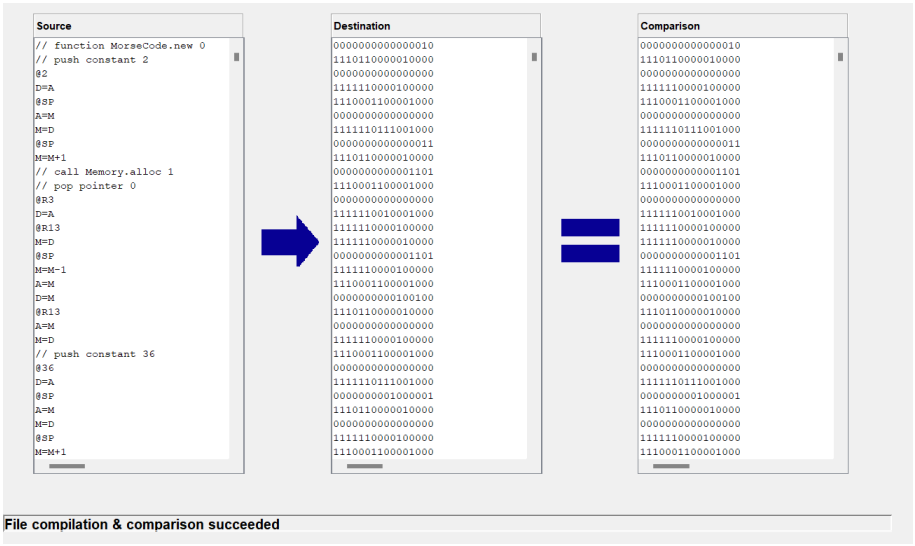


Figure 6.4: Output of comparison of file *Morsecode.jack* after conversion

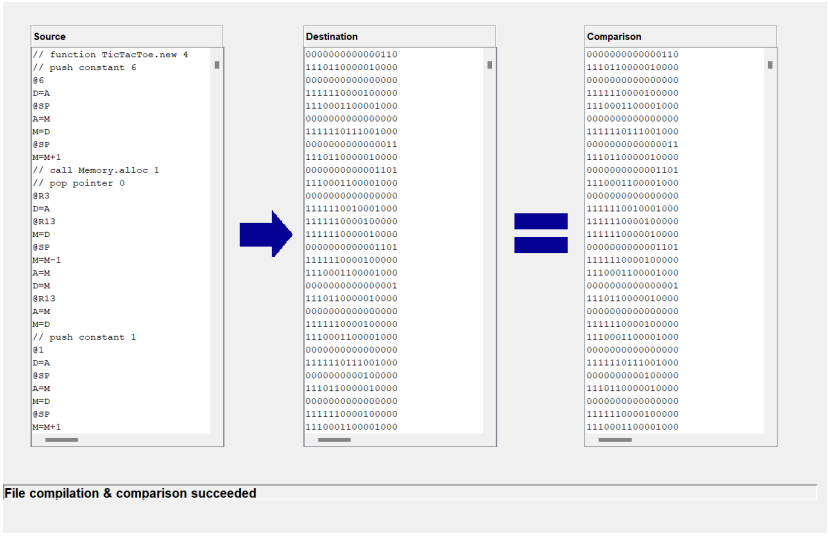


Figure 6.5: Output of comparison of file *TicTacToe.jack* after conversion

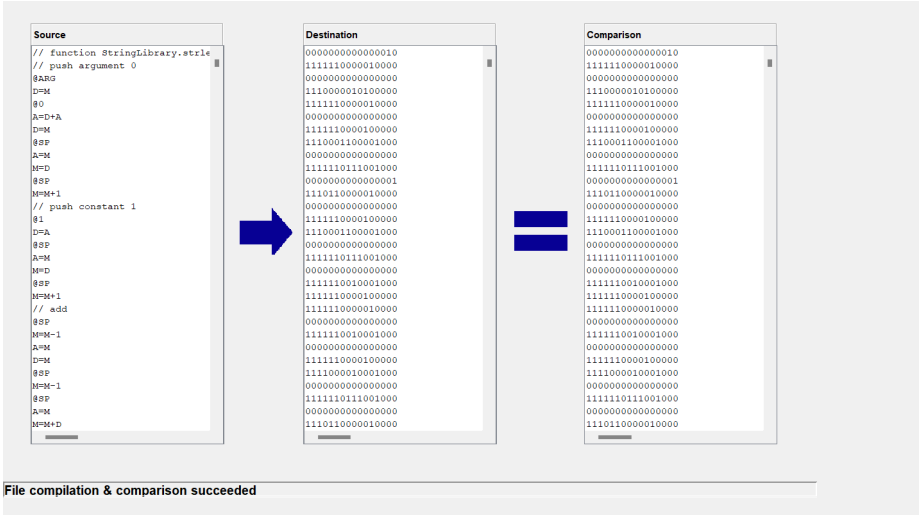


Figure 6.6: Output of comparison of file *String Library.jack* after conversion

# Chapter 7

# Conclusion and Future Work

While we tried to explain as easy as possible and tried to avoid errors, there is a possibility of human errors that can be hidden from our sight, or information gets outdated before we try to update it. We apologize for any errors you find in this paper. The code on the repository is licensed under *MIT License*, you can freely distribute or use the code as long as you bound by the license.

Finally, we hope you try the following projects and awesome tool *nand2tetris*, created by *Prof. Noam Nisan, Prof. Shimon Schocken and their team* on your computer. We consider this paper is useful for your next research work on this tool or something bigger

# Bibliography

https://www.nand2tetris.org/

https://docs.oracle.com/javase/tutorial/

https://www.geeksforgeeks.org/morse-code-implementation/

https://www.geeksforgeeks.org/array-implementation-of-queue-simple/

https://owlcation.com/humanities/morse_code