Contents

1 Basic Test Results

```
********** TESTING FOLDER STRUCTURE START **********
Checking your submission for presence of invalid (non-ASCII) characters...
No invalid characters found.
Submission logins are: linorcohen
Is this OK?
**********  TESTING FOLDER STRUCTURE END  **********

********** PROJECT TEST START **********
Running 'make'.
'make' ran successfully.
Testing.

Running your program with command: './Assembler Add.asm'.
diff succeeded on the test.

Running your program with command: './Assembler Max.asm'.
diff succeeded on the test.

Running your program with command: './Assembler Rect.asm'.
diff succeeded on the test.
**********  PROJECT TEST END  **********

Note: the tests you see above are all the presubmission tests
for this project. The tests might not check all the different
parts of the project or all corner cases, so write your own
tests and use them!
```

2 AUTHORS

```
linorcohen
Partner 1: Linor Cohen, linor.cohen@mail.huji.ac.il, 318861226
Remarks:
```

3 Assembler

```
#!/bin/sh
# This file only works on Unix-like operating systems, so it won't work on Windows.

## Why do we need this file?
# The purpose of this file is to run your project.
# We want our users to have a simple API to run the project.
# So, we need a "wrapper" that will hide all  details to do so,
# enabling users to simply type 'Assembler <path>' in order to use it.

## What are '#!/bin/sh' and '$*'?
# '$*' is a variable that holds all the arguments this file has received. So, if you
# run "Assembler trout mask replica", $* will hold "trout mask replica".

## What should I change in this file to make it work with my project?
# IMPORTANT: This file assumes that the main is contained in "Main.py".
#            If your main is contained elsewhere, you will need to change this.

python3 Main.py $*

# This file is part of nand2tetris, as taught in The Hebrew University, and
# was written by Aviv Yaish. It is an extension to the specifications given
# in https://www.nand2tetris.org (Shimon Schocken and Noam Nisan, 2017),
# as allowed by the Creative Common Attribution-NonCommercial-ShareAlike 3.0
# Unported License: https://creativecommons.org/licenses/by-nc-sa/3.0/
```

4 Code.py

```python
"""
This file is part of nand2tetris, as taught in The Hebrew University, and
was written by Aviv Yaish. It is an extension to the specifications given
[here](https://www.nand2tetris.org) (Shimon Schocken and Noam Nisan, 2017),
as allowed by the Creative Common Attribution-NonCommercial-ShareAlike 3.0
Unported [License](https://creativecommons.org/licenses/by-nc-sa/3.0/).
"""

from typing import Dict


class Code:
    """Translates Hack assembly language mnemonics into binary codes."""

    dest_table = {"null": "000", "M": "001", "D": "010", "DM": "011",
                  "A": "100", "AM": "101", "AD": "110", "AMD": "111",
                  "ADM": "111", "MAD": "111"}

    comp_table = {"0": "0101010", "1": "0111111", "-1": "0111010",
                  "D": "0001100", "A": "0110000", "!D": "0001101",
                  "!A": "0110001", "-D": "0001111", "-A": "0110011",
                  "D+1": "0011111", "A+1": "0110111", "D-1": "0001110",
                  "A-1": "0110010", "D+A": "0000010", "D-A": "0010011",
                  "D&A": "0000000", "D|A": "0010101", "M": "1110000",
                  "!M": "1110001", "-M": "1110011", "M+1": "1110111",
                  "M-1": "1110010", "D+M": "1000010", "D-M": "1010011",
                  "M-D": "1000111", "D&M": "1000000", "D|M": "1010101",
                  "A-D": "0000111", "D<<": "0110000", "A<<": "0100000",
                  "M<<": "1100000", "D>>": "0010000", "A>>": "0000000",
                  "M>>": "1000000"}

    jump_table = {"null": "000", "JGT": "001", "JEQ": "010", "JGE": "011",
                  "JLT": "100", "JNE": "101", "JLE": "110", "JMP": "111"}

    @staticmethod
    def dest(mnemonic: str) -> str:
        """
        Args:
            mnemonic (str): a dest mnemonic string.

        Returns:
            str: 3-bit long binary code of the given mnemonic.
        """
        return Code.__fetch_from_table(mnemonic, Code.dest_table)

    @staticmethod
    def comp(mnemonic: str) -> str:
        """
        Args:
            mnemonic (str): a comp mnemonic string.

        Returns:
            str: the binary code of the given mnemonic.
        """
        return Code.__fetch_from_table(mnemonic, Code.comp_table)

    @staticmethod
    def jump(mnemonic: str) -> str:
        """
        Args:
            mnemonic (str): a jump mnemonic string.

        Returns:
            str: 3-bit long binary code of the given mnemonic.
        """
        return Code.jump_table[mnemonic]

    @staticmethod
    def __fetch_from_table(mnemonic: str, table: Dict[str, str]) -> str:
        if mnemonic not in table:
            return table[mnemonic[::-1]]  # support reverse
        return table[mnemonic]
```

5 Main.py

```python
"""
This file is part of nand2tetris, as taught in The Hebrew University, and
was written by Aviv Yaish. It is an extension to the specifications given
[here](https://www.nand2tetris.org) (Shimon Schocken and Noam Nisan, 2017),
as allowed by the Creative Common Attribution-NonCommercial-ShareAlike 3.0
Unported [License](https://creativecommons.org/licenses/by-nc-sa/3.0/).
"""
import os
import sys
import typing
from SymbolTable import SymbolTable
from Parser import Parser
from Code import Code

INITIAL_ADDRESS = 16
ZERO_FILL = 15
NOT_FOUND = -1
LEFT_SHIFT = "<<"
RIGHT_SHIFT = ">>"
SHIFT_CODE = "101"
C_CODE = "111"
A_CODE = "0"


def assemble_file(
        input_file: typing.TextIO, output_file: typing.TextIO) -> None:
    """Assembles a single file.

    Args:
        input_file (typing.TextIO): the file to assemble.
        output_file (typing.TextIO): writes all output to this file.
    """
    # Initialization
    first_parser = Parser(input_file)
    input_file.seek(0)
    sec_parser = Parser(input_file)
    symbol_table = SymbolTable()
    available_address_idx = INITIAL_ADDRESS

    # First Pass
    while first_parser.has_more_commands():
        first_parser.advance()
        if first_parser.command_type() == first_parser.L_COMMAND:
            l_symbol = first_parser.symbol()
            symbol_table.add_entry(l_symbol, first_parser.command_idx + 1)

    # Second Pass
    while sec_parser.has_more_commands():
        sec_parser.advance()
        # If the instruction is @ symbol
        if sec_parser.command_type() == sec_parser.A_COMMAND:
            cur_address, address_idx = get_cur_address(available_address_idx, sec_parser,
                                                       symbol_table)
            available_address_idx = address_idx
            # Translates the symbol to its binary value
            output_file.write(
                A_CODE + bin(int(cur_address))[2:].zfill(ZERO_FILL) + '\n')

        # If the instruction is dest =comp ; jump
        elif sec_parser.command_type() == sec_parser.C_COMMAND:
            output_file.write(get_full_c_command(sec_parser))


def get_full_c_command(sec_parser: Parser) -> str:
    """
    This function returns the full binary command for type C_COMMAND
    :param sec_parser: current parser
    :return: string represent the binary code of the current C_COMMAND
    """
    comp = sec_parser.comp()
    full_command = Code.comp(comp) + Code.dest(sec_parser.dest()) + Code.jump(
        sec_parser.jump()) + '\n'
    if comp.find(LEFT_SHIFT) != NOT_FOUND or comp.find(
            RIGHT_SHIFT) != NOT_FOUND:
        return SHIFT_CODE + full_command
    return C_CODE + full_command


def get_cur_address(address_idx, sec_parser, symbol_table):
    """
    get the current symbol address from the symbol table
    :param address_idx: current available address
    :param sec_parser: secondary parser
    :param symbol_table: the symbol table to fetch from
    :return: the symbol address, current available address
    """
    cur_symbol = sec_parser.symbol()
    if not cur_symbol.isnumeric():
        # If symbol is not in the symbol table, adds it
        if not symbol_table.contains(cur_symbol):
            symbol_table.add_entry(cur_symbol, address_idx)
            address_idx += 1
        return symbol_table.get_address(cur_symbol), address_idx
    return cur_symbol, address_idx


if "__main__" == __name__:
    # Parses the input path and calls assemble_file on each input file.
    # This opens both the input and the output files!
    # Both are closed automatically when the code finishes running.
    # If the output file does not exist, it is created automatically in the
    # correct path, using the correct filename.
    if not len(sys.argv) == 2:
        sys.exit("Invalid usage, please use: Assembler <input path>")
    argument_path = os.path.abspath(sys.argv[1])
    if os.path.isdir(argument_path):
        files_to_assemble = [
            os.path.join(argument_path, filename)
            for filename in os.listdir(argument_path)]
    else:
        files_to_assemble = [argument_path]
    for input_path in files_to_assemble:
        filename, extension = os.path.splitext(input_path)
        if extension.lower() != ".asm":
            continue
        output_path = filename + ".hack"
        with open(input_path, 'r') as input_file, \
                open(output_path, 'w') as output_file:
            assemble_file(input_file, output_file)
```

6 Makefile

```
# Makefile for a script (e.g. Python)

## Why do we need this file?
# We want our users to have a simple API to run the project.
# So, we need a "wrapper" that will hide all  details to do so,
# thus enabling our users to simply type 'Assembler <path>' in order to use it.

## What are makefiles?
# This is a sample makefile.
# The purpose of makefiles is to make sure that after running "make" your
# project is ready for execution.

## What should I change in this file to make it work with my project?
# Usually, scripting language (e.g. Python) based projects only need execution
# permissions for your run file executable to run.
# Your project may be more complicated and require a different makefile.

## What is a makefile rule?
# A makefile rule is a list of prerequisites (other rules that need to be run
# before this rule) and commands that are run one after the other.
# The "all" rule is what runs when you call "make".
# In this example, all it does is grant execution permissions for your
# executable, so your project will be able to run on the graders' computers.
# In this case, the "all" rule has no preqrequisites.

## How are rules defined?
# The following line is a rule declaration:
# all:
#     chmod a+x Assembler

# A general rule looks like this:
# rule_name: prerequisite1 prerequisite2 prerequisite3 prerequisite4 ...
#    command1
#    command2
#    command3
#    ...
# Where each preqrequisite is a rule name, and each command is a command-line
# command (for example chmod, javac, echo, etc').

# Beginning of the actual Makefile
all:
    chmod a+x *

# This file is part of nand2tetris, as taught in The Hebrew University, and
# was written by Aviv Yaish. It is an extension to the specifications given
# in https://www.nand2tetris.org (Shimon Schocken and Noam Nisan, 2017),
# as allowed by the Creative Common Attribution-NonCommercial-ShareAlike 3.0
# Unported License: https://creativecommons.org/licenses/by-nc-sa/3.0/
```

7 Parser.py

```python
"""
This file is part of nand2tetris, as taught in The Hebrew University, and
was written by Aviv Yaish. It is an extension to the specifications given
[here](https://www.nand2tetris.org) (Shimon Schocken and Noam Nisan, 2017),
as allowed by the Creative Common Attribution-NonCommercial-ShareAlike 3.0
Unported [License](https://creativecommons.org/licenses/by-nc-sa/3.0/).
"""
import typing
import re


class Parser:
    """Encapsulates access to the input code. Reads an assembly program
    by reading each command line-by-line, parses the current command,
    and provides convenient access to the commands components (fields
    and symbols). In addition, removes all white space and comments.
    """
    A_COMMAND = "A_COMMAND"
    C_COMMAND = "C_COMMAND"
    L_COMMAND = "L_COMMAND"
    INITIAL_VAL = -1
    COMMENT = "//"
    NULL = "null"
    EMPTY = ""
    NOT_FOUND = -1

    def __init__(self, input_file: typing.TextIO) -> None:
        """Opens the input file and gets ready to parse it.

        Args:
            input_file (typing.TextIO): input file.
        """
        self.input_lines = input_file.read().splitlines()
        self.n = self.INITIAL_VAL
        self.command_idx = self.INITIAL_VAL
        self.cur_instruction = self.EMPTY

    def has_more_commands(self) -> bool:
        """Are there more commands in the input?

        Returns:
            bool: True if there are more commands, False otherwise.
        """
        while len(self.input_lines) - 1 != self.n:
            self.n += 1
            self.cur_instruction = self.input_lines[self.n].strip(). \
                replace(" ", "")
            if self.cur_instruction != self.EMPTY and self.cur_instruction[
                                                      0:2] != self.COMMENT:
                return True
        return False

    def advance(self) -> None:
        """Reads the next command from the input and makes it the current command.
        Should be called only if has_more_commands() is true.
        """
        if self.cur_instruction[0] != "(":  # not L_COMMAND
            self.command_idx += 1
        # remove inline comments:
```

```python
        inline_comment_idx = self.cur_instruction.find(self.COMMENT)
        if inline_comment_idx != self.NOT_FOUND:
            self.cur_instruction = self.cur_instruction[0:inline_comment_idx]
        # remove all additional tags:
        self.cur_instruction = ''.join(self.cur_instruction.split())

    def command_type(self) -> str:
        """
        Returns:
            str: the type of the current command:
            "A_COMMAND" for @Xxx where Xxx is either a symbol or a decimal number
            "C_COMMAND" for dest=comp;jump
            "L_COMMAND" (actually, pseudo-command) for (Xxx) where Xxx is a symbol
        """
        first_param = self.cur_instruction[0]
        if first_param == "(":
            return self.L_COMMAND
        elif first_param == "@":
            return self.A_COMMAND
        return self.C_COMMAND

    def symbol(self) -> str:
        """
        Returns:
            str: the symbol or decimal Xxx of the current command @Xxx or
            (Xxx). Should be called only when command_type() is "A_COMMAND" or
            "L_COMMAND".
        """
        command_type = self.cur_instruction[0]
        symbol = self.cur_instruction[1:]
        if command_type == "@":  # A_COMMAND symbol
            return symbol
        return symbol[:-1]  # L_COMMAND symbol

    def dest(self) -> str:
        """
        Returns:
            str: the dest mnemonic in the current C-command. Should be called
            only when commandType() is "C_COMMAND".
        """
        dest_idx = self.cur_instruction.find("=")
        if dest_idx == self.NOT_FOUND:
            return self.NULL
        return self.cur_instruction[0:dest_idx]

    def comp(self) -> str:
        """
        Returns:
            str: the comp mnemonic in the current C-command. Should be called
            only when commandType() is "C_COMMAND".
        """
        return re.split(';', re.split('=', self.cur_instruction)[-1])[0]

    def jump(self) -> str:
        """
        Returns:
            str: the jump mnemonic in the current C-command. Should be called
            only when commandType() is "C_COMMAND".
        """
        jump_idx = self.cur_instruction.find(";")
        if jump_idx == self.NOT_FOUND or self.cur_instruction[jump_idx + 1:] == '':
            return self.NULL
        return self.cur_instruction[jump_idx + 1:]
```

8 SymbolTable.py

```python
"""
This file is part of nand2tetris, as taught in The Hebrew University, and
was written by Aviv Yaish. It is an extension to the specifications given
[here](https://www.nand2tetris.org) (Shimon Schocken and Noam Nisan, 2017),
as allowed by the Creative Common Attribution-NonCommercial-ShareAlike 3.0
Unported [License](https://creativecommons.org/licenses/by-nc-sa/3.0/).
"""


class SymbolTable:
    """
    A symbol table that keeps a correspondence between symbolic labels and
    numeric addresses.
    """

    def __init__(self) -> None:
        """Creates a new symbol table initialized with all the predefined symbols
        and their pre-allocated RAM addresses, according to section 6.2.3 of the
        book.
        """
        self.symbol_table = {"SP": 0, "LCL": 1, "ARG": 2, "THIS": 3, "THAT": 4,
                             "R0": 0,
                             "R1": 1,
                             "R2": 2,
                             "R3": 3,
                             "R4": 4,
                             "R5": 5,
                             "R6": 6,
                             "R7": 7,
                             "R8": 8,
                             "R9": 9,
                             "R10": 10,
                             "R11": 11,
                             "R12": 12,
                             "R13": 13,
                             "R14": 14,
                             "R15": 15,
                             "SCREEN": 16384, "KBD": 24576}

    def add_entry(self, symbol: str, address: int) -> None:
        """Adds the pair (symbol, address) to the table.

        Args:
            symbol (str): the symbol to add.
            address (int): the address corresponding to the symbol.
        """
        self.symbol_table[symbol] = address

    def contains(self, symbol: str) -> bool:
        """Does the symbol table contain the given symbol?

        Args:
            symbol (str): a symbol.

        Returns:
            bool: True if the symbol is contained, False otherwise.
        """
        return symbol in self.symbol_table

    def get_address(self, symbol: str) -> int:
        """Returns the address associated with the symbol.

        Args:
            symbol (str): a symbol.

        Returns:
            int: the address associated with the symbol.
        """
        return self.symbol_table[symbol]
```

9 rect/Rect.asm

```
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/06/rect/Rect.asm

// Draws a rectangle at the top-left corner of the screen.
// The rectangle is 16 pixels wide and R0 pixels high.

   @0
   D=M
   @INFINITE_LOOP
   D;JLE
   @counter
   M=D
   @SCREEN
   D=A
   @address
   M=D
(LOOP)
   @address
   A=M
   M=-1
   @address
   D=M
   @32
   D=D+A
   @address
   M=D
   @counter
   MD=M-1
   @LOOP
   D;JGT
(INFINITE_LOOP)
   @INFINITE_LOOP
   0;JMP
```

10 rect/Rect.hack

```
0000000000000000
1111110000010000
0000000000010111
1110001100000110
0000000000010000
1110001100001000
0100000000000000
1110110000010000
0000000000010001
1110001100001000
0000000000010001
1111110000100000
1110111010001000
0000000000010001
1111110000010000
0000000000100000
1110000010010000
0000000000010001
1110001100001000
0000000000010000
1111110010011000
0000000000001010
1110001100000001
0000000000010111
1110101010000111
```

11 rect/RectL.asm

```
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/06/rect/RectL.asm

// Symbol-less version of the Rect.asm program.

@0
D=M
@23
D;JLE
@16
M=D
@16384
D=A
@17
M=D
@17
A=M
M=-1
@17
D=M
@32
D=D+A
@17
M=D
@16
MD=M-1
@10
D;JGT
@23
0;JMP
```

12 rect/RectL.hack

```
0000000000000000
1111110000010000
0000000000010111
1110001100000110
0000000000010000
1110001100001000
0100000000000000
1110110000010000
0000000000010001
1110001100001000
0000000000010001
1111110000100000
1110111010001000
0000000000010001
1111110000010000
0000000000100000
1110000010010000
0000000000010001
1110001100001000
0000000000010000
1111110010011000
0000000000001010
1110001100000001
0000000000010111
1110101010000111
```