Problem #1

Subject: Hack Disassembler

Filename: hw02_01.py

Allowed modules: NONE

Write a program that opens asks the user for the name of a Hack machine code file (WITHOUT the extension). Open the file and read the contents. Output a disassembled code listing, to the console, in a format similar to the above (the original machine encoding, some space, and the disassembled instruction).

Do NOT make this harder than it is! You do not need to know what any of these instructions do, you are merely translating pa□erns of ones and zeroes into corresponding strings of text.

You may read the file a line at a time or all at once, it is up to you.

Use a separate dictionary for the three fields of the C-type instruction.

WORK

After prompting the user to input the filename of a .hack file, the program reads the contents of the file, storing each line as an instruction. For each instruction, the program disassembles it into assembly code based on predefined patterns. If the instruction starts with '0', it's considered an A-type instruction and converted accordingly. If the instruction starts with '111', it's considered a C-type instruction, and the computation, destination, and jump fields are mapped to their respective text strings using dictionaries. The disassembled assembly code is printed to the console. The main function encapsulates the execution of the program, ensuring it runs when the script is executed directly.

OUTPUT

DATA ENTRY

Enter the name of a .hack file: hackman

DISASSEMBLED OUTPUT

0000000000000000 @0

CODE

```
'''
PROGRAMMER: .... Jakob K. West
USERNAME: ...... jwest21
PROGRAM: ....... hw02_01.py
DESCRIPTION: Disassembling hackman
'''
# Function used to disassemble
def disassemble_instruction(instruction):
    if instruction.startswith('0'):
        return f"@{int(instruction[1:], 2)}"
    elif instruction.startswith('111'):
        comp = instruction[1:8]
        dest = instruction[8:11]
        jump = instruction[11:]
        return f"{comp_dict[comp]}={dest_dict[dest]};{jump_dict[jump]}"
def read_hack_file(filename):
    with open(filename + '.hack', 'r') as file:
        lines = file.readlines()
    return  [line.strip() for line in lines if line.strip()]
# Dictionary mapping comp, dest, and jump fields to their respective text strings
# Dictionary #1 - comp
comp_dict = {
    '0000000': '0',
    '0000001': '1',
```

```
    '0000010': '-1',
    '0001100': 'D',
    '0001101': 'A',
    '0001111': '!D',
    '0001110': '!A',
    '0011101': '-D',
    '0011111': '-A',
    '0011110': 'D+1',
    '0011011': 'A+1',
    '0000011': 'D-1',
    '0000111': 'A-1',
    '0001112': 'D+A',
    '0011000': 'D-A',
    '0010011': 'A-D',
    '0000110': 'D&A',
    '0010101': 'D|A',
    '1110000': 'M',
    '1110001': '!M',
    '1110011': '-M',
    '1110111': 'M+1',
    '1110010': 'M-1',
    '1111111': 'D+M',
    '1111010': 'D-M',
    '1110011': 'M-D',
    '1110000': 'D&M',
```

```
    '1110101': 'D|M'}
# Dictionary #2 - dest
dest_dict = {
    '000': 'null',
    '001': 'M',
    '010': 'D',
    '011': 'MD',
    '100': 'A',
    '101': 'AM',
    '110': 'AD',
    '111': 'AMD'
}
# Dictionary #3 - jump
jump_dict = {
    '000': 'null',
    '001': 'JGT',
    '010': 'JEQ',
    '011': 'JGE',
    '100': 'JLT',
    '101': 'JNE',
    '110': 'JLE',
    '111': 'JMP'
}
# main method
def main():
```

```
    print("DATA ENTRY")
    filename = input("Enter the name of a .hack file: ")
    instructions = read_hack_file(filename)
    # Final print statements
    print("\nDISASSEMBLED OUTPUT")
    for instruction in instructions:
        print(instruction, disassemble_instruction(instruction))
if __name__ == "__main__":
    main()
```

Problem #2

Subject: Proper mortgage payment

Filename: hw02_02.py

Allowed modules: none

Write a program that asks the user for a loan amount, an APR, and a term (in years) and produces a report (to the console) stating the loan terms, including the monthly payment and the amount of the final payment. It should also provide the total amount that the borrow will pay over the life of the loan and the cost of credit (i.e., the total amount of interest paid).

Run your program for a 30-year loan of $388,000 at an APR of 2.5% and also at an APR of 7%.

WORK

After obtaining user input for the loan amount, APR, and loan term, the program calculates the monthly payment and final payment for the mortgage using the **mortgage_payment()** function. This function employs the formula for calculating the monthly payment for a fixed-rate mortgage, taking into account the loan amount, APR, and term. Additionally, a **mortgage_residual()** function calculates the residual amount due on the mortgage based on the original loan amount, APR, term, and monthly payment.

Inside the **main()** function, the calculated monthly payment and final payment are utilized to determine the total amount paid over the loan term and the cost of credit (i.e., total interest paid). Finally, the program prints a summary of the mortgage terms, including the loan amount, APR, loan term, monthly payment, total amount paid, and cost of credit. The execution of the program is encapsulated within the **main()** function, ensuring it runs only when the script is executed directly.

OUTPUT

```
DATA ENTRY

Enter loan amount ($):...... 1234567.89

Enter loan APR (%):......... 6.283

Enter loan term (yr):....... 34



MORTGAGE TERMS

Loan amount: $         1234567.89

Loan rate:           6.283 %
```

```
Loan term:           34 years
Monthly payment: $         7335.11
Final Payment: $         13799.10
Total paid: $         2992724.88
Cost of credit: $         1758156.99
--------------------------------------------------------------------------------
DATA ENTRY
Enter loan amount ($):...... 388000
Enter loan APR (%):......... 2.5
Enter loan term (yr):....... 30

MORTGAGE TERMS
Loan amount: $          388000.00
Loan rate:           2.500 %
Loan term:           30 years
Monthly payment: $         1533.07
Final Payment: $         2341.40
Total paid: $         551905.20
Cost of credit: $         163905.20
--------------------------------------------------------------------------------
DATA ENTRY
Enter loan amount ($):...... 388000
Enter loan APR (%):......... 7
Enter loan term (yr):....... 30
```

```
MORTGAGE TERMS

Loan amount: $         388000.00

Loan rate:           7.000 %

Loan term:           30 years

Monthly payment: $         2581.37

Final Payment: $         4844.71

Total paid: $         929293.20

Cost of credit: $         541293.20
```

CODE

```
'''

PROGRAMMER: .... Jakob K. West

USERNAME: ...... jwest21

PROGRAM: ....... hw02_02.py

DESCRIPTION: Creating a mortgage_residual function

'',

def mortgage_residual(amount, rate, term, monthly_payment):

    monthly_interest_rate = rate / 100 /12

    total_payments = term * 12

    for i in range(total_payments):

        interest_charge = amount * monthly_interest_rate

        principal_payment = monthly_payment - interest_charge

        amount -= principal_payment

    return round(amount, 2)

def mortgage_payment(amount, rate, term):

    monthly_interest_rate = rate / 100 / 12

    total_payments = term * 12

    # Calculate monthly payment

    monthly_payment = amount * monthly_interest_rate / (1 - (1 +
monthly_interest_rate) ** -total_payments)

    final_payment = monthly_payment + (amount * monthly_interest_rate)

    return round(monthly_payment, 2), round(final_payment, 2)

def main():

    # Get input from user (loan_amount, apr, loan_term)

    print("DATA ENTRY")
```

```
    loan_amount = float(input("Enter loan amount ($):..... "))
    apr = float(input("Enter loan APR (%):..... "))
    loan_term = int(input("Enter loan term (yr):..... "))
    # Calculate mortgage terms
    monthly_payment, final_payment = mortgage_payment(loan_amount, apr, loan_term)
    residual = mortgage_residual(loan_amount, apr, loan_term, monthly_payment)
    total_paid = monthly_payment * loan_term * 12
    cost_of_credit = total_paid - loan_amount
    # Display mortgage terms (final print statements)
    print("\nMORTGAGE TERMS")
    print(f"Loan amount:.............. {'$':<10}{loan_amount:.2f}")
    print(f"Loan rate:................ {'':<10}{apr:.3f} %")
    print(f"Loan term:................ {'':<10}{loan_term} years")
    print(f"Monthly payment:.......... {'$':<10}{monthly_payment:.2f}")
    print(f"Final Payment:............ {'$':<10}{final_payment:.2f}")
    print(f"Total paid:............... {'$':<10}{total_paid:.2f}")
    print(f"Cost of credit:........... {'$':<10}{cost_of_credit:.2f}")
if __name__ == "__main__":
    main()
```

Problem #3

Subject: Extract assembly code from disassembler output.

Filename: hw02_03.py Allowed modules: re

Capture the output from your Problem #2 code (or have your script for that problem write it to a file, your choice) named ‘hackman.dis’.

Write a program that asks the user for a disassembler output file (w/o extension), reads the file (.dis extension), and produces an assembler file (.asm extension) with just the assembly code. The code should also be echoed to the console.

Run your program on your hackman.dis file.

WORK

Upon receiving user input for the disassembler output file name (without extension), the program calls the extract_assembly_code() function, passing the provided filename as an argument. This function reads the contents of the disassembler output file, extracts the assembly code lines using a regular expression pattern, and writes the extracted assembly code to a new file with a ".asm" extension. Inside the extract_assembly_code() function, the regular expression pattern is defined to match lines that do not start with a binary digit (0 or 1) and contain any characters until the end of the line. Each matching assembly code line is printed to the console and written to the output file. The execution of the program is encapsulated within the main() function, ensuring it runs only when the script is executed directly.

OUTPUT

DATA ENTRY

Enter a disassembler output file name (w/o extension): hackman

ASSEMBLY CODE

.. ..

CODE

```
'''
PROGRAMMER: .... Jakob K. West
USERNAME: ...... jwest21
PROGRAM: ....... hw02_03.py
DESCRIPTION: ...
'''
import re
def extract_assembly_code(input_filename):
    output_filename = input_filename + ".asm"
# Open the disassembler output file for reading
with open(input_filename + ".dis", "rt") as fp:
        data = fp.read()
    # Define a regular expression pattern to match assembly code
    regex = re.compile(r"^\s*(?![01])[^\n]*$", re.MULTILINE)
    results = regex.findall(data)
    # Output file for writing
    with open(output_filename, "wt") as fp:
        for item in results:
            print(item)
            fp.write(item + "\n")
def main():
    # Input disassembler output file name
    print("DATA ENTRY")
    filename = input("Enter a disassembler output file name (w/o extension): ")
```

```
    # print the assembly code
    print("\nASSEMBLY CODE")
    extract_assembly_code(filename)
if __name__ == "__main__":
    main()
```