Memory

- Purpose of this exercise is to give exposure to memory access patterns, and some practice examining how a program uses memory.
- Files
  - **Makefile**
    - heaploop: compile heaploop.c
    - matmul: compile matmul.c
    - traces: executes runit on compiled heaploop.c and matmul.c
    - clean: remove everything created by make
  - **heaploop.c**: file to trace
  - **matmul.c**: file to trace
  - **runit**: runs valgrind to trace program, pass it to refstring.py, output contents to tr-*.ref file
  - **refstring.py**: remove valgrind comments
- Trace file contains memory address in hexadecimal, and character in second field to determine type of access
  - I: instruction
  - Data
    - S: store
    - L: load
    - M: modify
- Translate Virtual Address -> Page Table -> Physical Address
  - Page: 4kb. Therefore 12 bits page offset. (Convert 4000 decimal to binary)
  - Example
    - Virtual address = 7ff0008e8
    - Convert virtual address to binary = 0111 1111 1111 0000 0000 0000 1000 1110 1000
    - Remove last 12 bits from virtual address binary = 0111 1111 1111 0000 0000 0000
    - Convert the binary from last step to hex = 7FF000
- **Excercise**: Write a program that will translate each memory reference into a page number assuming pages are 4096 bytes. Output a table of each unique intruction pages with a count of the number of accesses for each page, and a table of unique data pages with a count of the number of accesses for each page.
  - Refer to code 1

```
# Code 1
'''Translate memory reference into page number assuming pages are 4096
bytes'''

FILENAME = "tr-heaploop.ref"


def conversion(hexa):
    '''Given virtual page address convert it to physical page

    Givn a physical page address in hexadecimal, convert it to binary,
    remove number of bits from the end depending on bits needed to
```

```
represent
    size of page, conver this value back to hexadecimal.

    Args:
        hexa: string which holds an hexadecimal number representing
physical
                page address

    Returns: virtual page address in hexadecimal
    '''
    number_of_bits = 12

    binary = bin(int(hexa, 16))

    binary = binary[:len(binary) - number_of_bits]

    binary = int(binary, 2)

    return hex(binary)


def generate_data(filename):
    '''Given a filename in the current folder get all the page info

    Given a filename find convert the physical page address to virtual page
    address, identify the number of time the pages has been accessed,
seperate
    out the Instruction data accesses from the other accesses.

    Args:
        filename: string which holds the name of the file in the current
                    directory which holds all the information that needs to
                    be parsed.
                    Ex format: hexadeimal,access_type

    Returns: 2 parameters
        data_holder: dictionary which has virtual hexadecimal keys
associated
                        with number of page access. Ex: {str: int}
        instructions: set which holds all the virtual hexadecimal key with
                        instruction data accesses
    '''
    data_holder = {}
    instructions = set()

    with open(filename) as trace_file:
        for line in trace_file:
            line = line.strip()
            memory, instruction = line.split(",")

            converted_mem = conversion(memory)

            if converted_mem not in data_holder:
                data_holder[converted_mem] = 0
            data_holder[converted_mem] += 1

            if instruction == "I":
                instructions.add(converted_mem)
        return data_holder, instructions


def display_contents(data_holder, instructions):
    '''Displays the number of page accesses and page with instruction
access

    Displays the data in a specified format. Displays instructions type
    accesses with virtual page address followed by number of times called.
Then
    display all the accesses (other than instruction), followed by number
of
    times called

    Args:
        data_holder: dictionary which has virtual hexadecimal keys
associated
                        with number of page access. Ex: {str: int}
        instructions: set which holds all the virtual hexadecimal key with
                        instruction data accesses

    Returns: str with all the data organized in specified format
    '''
    contents = "Instructions\n"

    for i in instructions:
        contents += i + "," + str(data_holder[i]) + "\n"
        data_holder.pop(i)

    contents += "\nData:\n"
    for key, value in data_holder.items():
        contents += key + "," + str(value) + "\n"

    return contents.strip()


if __name__ == "__main__":
    DATA_HOLDER, INSTRUCTIONS = generate_data(FILENAME)
    CONTENTS = display_contents(DATA_HOLDER, INSTRUCTIONS)

    print(CONTENTS)
```