**Assignment 1**

**Name** : Shlok Zanwar

**Batch** : C-3

**Roll No** : 323071

**Gr No.** : 21910163

Files Required for the programs (Emot, class, Input files etc.)

  

  

Programs And Output

Week 1 - Program 1

Code :

import re

str = "MOVER       ,- AREG,=‘5’"

str = re.sub(',', ' ', str)

str = re.sub('\s+', ' ', str)

print(str)

print(str.split(" "))

Output :

MOVER - AREG =‘5’

['MOVER', '-', 'AREG', '=‘5’']

Week 1 - Program 2

Code :

file1 = open('input1.txt', 'r')

Lines = file1.readlines()

count = 0

for line in Lines:

    count += 1

    print(line.strip())

Week 1 - Program 3

Code :

import json

import re

from tabulate import tabulate

file = open("EmotJSON.json", "r+")

emotJson = json.loads(file.read())

file = open("EmotArray.json", "r+")

emotArray = json.loads(file.read())

# print("yes" if "STdOP" in emotJson else "no")

file = open('input1.txt', 'r')

fileLines = file.readlines()

lines = []

for line in fileLines:

    line = line.strip()

    line = re.sub(',', ' ', line)

    line = re.sub('\s+', ' ', line)

    # print(line)

    lines.append(line.split(" "))

symbolTable = []

# format = {

#     "symbol": "something",

#     "address": "something"

# }

for line in lines:

    if line[0] not in emotJson:

        symbolTable.append({

            "symbol": line[0],

            "address": None

        })

print(symbolTable)

headers = {"symbol": "Symbol", "address": "Address"}

print(tabulate(symbolTable, headers=headers, tablefmt="pretty"))

Output :

+--------+---------+

| Symbol | Address |

+--------+---------+

|   L1   |         |

|  LAST  |         |

|   X    |         |

|  ONE   |         |

|  TEN   |         |

+--------+---------+

Week 1 - Program 4

Code :

import json

import re

from tabulate import tabulate

file = open("EmotJSON.json", "r+")

emotJson = json.loads(file.read())

file = open("EmotArray.json", "r+")

emotArray = json.loads(file.read())

# print("yes" if "STdOP" in emotJson else "no")

file = open('input1.txt', 'r')

fileLines = file.readlines()

lines = []

for line in fileLines:

    line = line.strip()

    line = re.sub(',', ' ', line)

    line = re.sub('\s+', ' ', line)

    # print(line)

    lines.append(line.split(" "))

lc = 0

symbolTable = []

# format = {

#     "symbol": "something",

#     "address": "something"

# }

for line in lines:

    if line[0] == "START":

        lc = int(line[1])

        continue

    if line[0] not in emotJson:

        symbolTable.append({

            "symbol": line[0],

            "address": lc,

            "size": 1

        })

    lc += 1

# print(symbolTable)

headers = {"symbol": "Symbol", "size": "Size", "address": "Address"}

print(tabulate(symbolTable, headers=headers, tablefmt="pretty"))

Output :

+--------+---------+------+

| Symbol | Address | Size |

+--------+---------+------+

|   L1   |   101   |  1   |

|  LAST  |   106   |  1   |

|   X    |   107   |  1   |

|  ONE   |   108   |  1   |

|  TEN   |   109   |  1   |

+--------+---------+------+

Week 2 - Program 1, 2

Code :

import json

import re

from tabulate import tabulate

def giveLCToLiterals(literalTable, lc):

    for literal in literalTable:

        literal['address'] = lc

        lc += 1

    return literalTable, lc

file = open("EmotJSON.json", "r+")

emotJson = json.loads(file.read())

file = open("EmotArray.json", "r+")

emotArray = json.loads(file.read())

# print("yes" if "STdOP" in emotJson else "no")

file = open('input2.txt', 'r')

fileLines = file.readlines()

lines = []

for line in fileLines:

    line = line.strip()

    line = re.sub(',', ' ', line)

    line = re.sub('\s+', ' ', line)

    # print(line)

    lines.append(line.split(" "))

lc = 0

symbolTable = []

# format = {

#     "symbol": "something",

#     "lc": "something"

# }

literalTable = []

# format = {

#     "literal": "somthing",

#     "lc": "something"

# }

for line in lines:

    if line[0] == "START":

        lc = int(line[1])

        continue

    if line[0] == "END":

        literalTable, lc = giveLCToLiterals(literalTable, lc)

    if line[0] not in emotJson:

        symbolTable.append({

            "symbol": line[0],

            "address": lc,

        })

    if re.fullmatch("='([0-9]+)'", line[-1]):

        literalTable.append({

            "literal": line[-1],

            "newLiteral": int(line[-1][2:-1]),

            "address": None

        })

    lc += 1

# print(symbolTable)

headers = {"symbol": "Symbol", "address": "Address"}

print(tabulate(symbolTable, headers=headers, tablefmt="pretty"))

headers = {"literal": "Literal", "address": "Address"}

print(tabulate(literalTable, headers=headers, tablefmt="pretty"))

Week 3 - Program 1, 2, 3

Code :

import re

import json

from tabulate import tabulate

def giveLCToLiterals(literalTable, lc, poolTable=None):

    if poolTable is None:

        poolTable = [0]

    if poolTable[-1] == len(literalTable):

        return literalTable, lc, poolTable

    for i in range(poolTable[-1], len(literalTable)):

        literalTable[i]['address'] = lc

        lc += 1

    poolTable.append(len(literalTable))

    return literalTable, lc, poolTable

def handleOrigin(symbolTable, addToSymbol, addNum):

    for symbol in symbolTable:

        if(symbol['symbol'] == addToSymbol):

            return symbol['address'] + addNum

def handleEqu(symbolTable, leftSymbol, rightSymbol):

    address = 0

    for symbol in symbolTable:

        if (symbol['symbol'] == rightSymbol):

            address = symbol['address']

    index = 0

    for symbol in symbolTable:

        if (symbol['symbol'] == leftSymbol):

            symbol['address'] = address

            break

        index += 1

    if index == len(symbolTable):

        symbolTable.append({

            "symbol": leftSymbol,

            "address": address,

        })

    return symbolTable

file = open("EmotJSON.json", "r+")

emotJson = json.loads(file.read())

file = open('input2.txt', 'r')

fileLines = file.readlines()

lines = []

for line in fileLines:

    line = line.strip()

    line = re.sub(',', ' ', line)

    line = re.sub('\s+', ' ', line)

    lines.append(line.split(" "))

poolTable = [0]

literalTable = []

symbolTable = []

lc = 0

for line in lines:

    if line[0] == "START":

        lc = int(line[1])

        continue

    if line[0] == "END":

        literalTable, lc, poolTable = giveLCToLiterals(literalTable, lc, poolTable)

        continue

    if line[0] == "LTORG":

        # print(poolTable, lc, literalTable)

        literalTable, lc, poolTable = giveLCToLiterals(literalTable, lc, poolTable)

        continue

    if line[0] == "ORIGIN":

        arg = line[1].split("+")

        addTo = arg[0]

        add = int(arg[1])

        lc = handleOrigin(symbolTable, addTo, add)

        continue

    if line[1] == "EQU":

        symbolTable = handleEqu(symbolTable, line[2], line[0])

        continue

    if line[0] not in emotJson:

        symbolTable.append({

            "symbol": line[0],

            "address": lc,

        })

    # print (line)

    if re.fullmatch("='([0-9]+)'", line[-1]):

        # print(line[-1])

        literalTable.append({

            "literal": line[-1],

            "newLiteral": int(line[-1][2:-1]),

            "address": None

        })

    elif line[-1] not in emotJson:

        symbolTable.append({

            "symbol": line[-1],

            "address": None,

        })

    # print(line, lc)

    lc += 1

headers = {"symbol": "Symbol", "address": "Address"}

print("Symbol Table :- \n", tabulate(symbolTable, headers=headers, tablefmt="pretty"))

headers = {"literal": "Literal", "address": "Address"}

print("\nLiteral Table :- \n", tabulate(literalTable, headers=headers, tablefmt="pretty"))

print("\nPool Table :- ", poolTable)

Week 4 Assignment 1 (Complete)

Code :

import re

import json

from tabulate import tabulate

def giveLCToLiterals(literalTable, lc, poolTable=None):

    if poolTable is None:

        poolTable = [0]

    if poolTable[-1] == len(literalTable):

        return literalTable, lc, poolTable

    for i in range(poolTable[-1], len(literalTable)):

        literalTable[i]['address'] = lc

        lc += 1

    poolTable.append(len(literalTable))

    return literalTable, lc, poolTable

def handleOrigin(symbolTable, addToSymbol, addNum):

    for symbol in symbolTable:

        if(symbol['symbol'] == addToSymbol):

            return symbol['address'] + addNum

def handleEqu(symbolTable, leftSymbol, rightSymbol):

    address = 0

    for symbol in symbolTable:

        if (symbol['symbol'] == rightSymbol):

            address = symbol['address']

    index = 0

    for symbol in symbolTable:

        if (symbol['symbol'] == leftSymbol):

            symbol['address'] = address

            break

        index += 1

    if index == len(symbolTable):

        symbolTable.append({

            "symbol": leftSymbol,

            "address": address,

        })

    return symbolTable, address

def isConstant(variable):

    try:

        int(variable)

        return True

    except ValueError:

        return False

file = open("EmotJSON.json", "r+")

emotJson = json.loads(file.read())

file = open("Class.json", "r+")

classJson = json.loads(file.read())

file = open('Input3.txt', 'r')

fileLines = file.readlines()

outputFile = open("Output.txt", "w+")

lines = []

for line in fileLines:

    line = line.strip()

    line = re.sub(',', ' ', line)

    line = re.sub('\s+', ' ', line)

    lines.append(line.split(" "))

poolTable = [0]

literalTable = []

symbolTable = []

lc = 0

for line in lines:

    newOutput = ""

    incrementLC = True

    for word in line:

        if word in emotJson:

            if line[0] == "LTORG":

                literalTable, lc, poolTable = giveLCToLiterals(literalTable, lc, poolTable)

                for i in range(poolTable[-2], poolTable[-1]):

                    newOutput += "(DL,02) (C," + str(literalTable[i]['newLiteral']) + ") "

                    if(i < poolTable[-1]-1):

                        newOutput += "\n"

                incrementLC = False

                break

            newOutput += "(" + classJson[emotJson[word]['class']] + "," + emotJson[word]['opcode'] + ") "

            if word == "START":

                lc = int(line[1])

                newOutput += "(C," + str(lc) + ")"

                incrementLC = False

                break

            if word == "STOP":

                # incrementLC = False

                break

            if word == "END":

                literalTable, lc, poolTable = giveLCToLiterals(literalTable, lc, poolTable)

                incrementLC = False

                break

            if word == "ORIGIN":

                arg = line[1].split("+")

                addTo = arg[0]

                add = int(arg[1])

                lc = handleOrigin(symbolTable, addTo, add)

                newOutput += "(C," + str(lc) + ")"

                incrementLC = False

                break

            if word == "EQU":

                symbolTable, address = handleEqu(symbolTable, line[0], line[2])

                newOutput += "(C," + str(address) + ")"

                incrementLC = False

                break

        else:

            if re.fullmatch("='([0-9]+)'", word):

                newOutput += "(L," + str(len(literalTable)) + ") "

                literalTable.append({

                    "literal": word,

                    "newLiteral": int(word[2:-1]),

                    "address": None

                })

            elif isConstant(word):

                newOutput += "(C," + str(word) + ") "

            else:

                if word == line[0]:

                    index = 0

                    for symbol in symbolTable:

                        if(symbol["symbol"] == word):

                            symbol["address"] = lc

                            break

                        index += 1

                    if index == len(symbolTable):

                        symbolTable.append({

                            "symbol": word,

                            "address": lc,

                        })

                else:

                    index = 0

                    for symbol in symbolTable:

                        if (symbol["symbol"] == word):

                            break

                        index += 1

                    if index == len(symbolTable):

                        symbolTable.append({

                            "symbol": word,

                            "address": None,

                        })

                    newOutput += "(S," + str(index) + ") "

    # print(line, lc)

    outputFile.write(newOutput + "\n")

    if incrementLC:

        lc += 1

headers = {"symbol": "Symbol", "address": "Address"}

print("Symbol Table :- \n", tabulate(symbolTable, headers=headers, tablefmt="pretty"))

headers = {"literal": "Literal", "address": "Address"}

print("\nLiteral Table :- \n", tabulate(literalTable, headers=headers, tablefmt="pretty"))

print("\nPool Table :- ", poolTable)

outputFile.close()

outputFile = open("Output.txt", "r+")

print("\nOutput File :-")

outputLines = outputFile.readlines()

for line in outputLines:

    print(line, end="")

Output :

Symbol Table :-

 +--------+---------+

| Symbol | Address |

+--------+---------+

|   X    |   228   |

|   Y    |   229   |

|  NEXT  |   205   |

|  BACK  |   205   |

+--------+---------+

Literal Table :-

 +---------+------------+---------+

| Literal | newLiteral | Address |

+---------+------------+---------+

|  ='5'   |     5      |   208   |

|  ='1'   |     1      |   209   |

|  ='1'   |     1      |   210   |

|  ='1'   |     1      |   216   |

|  ='2'   |     2      |   217   |

|  ='1'   |     1      |   230   |

+---------+------------+---------+

Pool Table :-  [0, 3, 5, 6]

Output File :-

(AD,01) (C,200)

(IS,09) (S,0)

(IS,09) (S,1)

(IS,04) (RG,01) (L,0)

(IS,05) (RG,01) (S,0)

(IS,04) (RG,03) (L,1)

(IS,04) (RG,02) (L,2)

(AD,03) (C,208)

(DL,02) (C,5)

(DL,02) (C,1)

(DL,02) (C,1)

(IS,01) (RG,01) (L,3)

(IS,02) (RG,02) (S,1)

(IS,03) (RG,03) (L,4)

(IS,06) (RG,01) (RG,02)

(AD,04) (C,205)

(IS,07) (CC,02) (S,3)

(DL,02) (C,1)

(DL,02) (C,2)

(AD,03) (C,225)

(IS,01) (RG,01) (L,5)

(IS,03) (RG,03) (S,0)

(IS,00)

(DL,01) (C,1)

(DL,01) (C,1)

(AD,02)

Thank You !