EAST WEST UNIVERSITY

Mini Project

CSE-366

Sec-02

Submitted to:

Al Imran
Lecturer,
Department of Computer Science & Engineering

Submitted by:

Name: Rashik Buksh Rafsan
ID: 2018-3-60-088

Problem-1: Income Tax Calculation

```
def normalpeople(n):
    if 300 >= n > 0:
        print('Income = ' + str(n) + 'k\nTax = ' + str(n*0))
    elif 400 >= n > 300:
        print('Income = ' + str(n) + 'k\nTax = ' + str(n*5))
    elif 700 >= n > 400:
        print('Income = ' + str(n) + 'k\nTax = ' + str(n*10))
    elif 1100 >= n > 700:
        print('Income = ' + str(n) + 'k\nTax = ' + str(n*15))
    elif 1600 >= n > 1100:
        print('Income = ' + str(n) + 'k\nTax = ' + str(n*20))
    else:
        print('Income = ' + str(n) + 'k\nTax = ' + str(n*25))


def womenand65people(n):
    if 350 >= n > 0:
        print('Income = ' + str(n) + 'k\nTax = ' + str(n*0))
    elif 450 >= n > 350:
        print('Income = ' + str(n) + 'k\nTax = ' + str(n*5))
    elif 750 >= n > 450:
        print('Income = ' + str(n) + 'k\nTax = ' + str(n*10))
    elif 1150 >= n > 750:
        print('Income = ' + str(n) + 'k\nTax = ' + str(n*15))
    elif 1650 >= n > 1150:
        print('Income = ' + str(n) + 'k\nTax = ' + str(n*20))
    else:
        print('Income = ' + str(n) + 'k\nTax = ' + str(n*25))


def disabledpeople(n):
    if 450 >= n > 0:
        print('Income = ' + str(n) + 'k\nTax = ' + str(n*0))
    elif 550 >= n > 450:
        print('Income = ' + str(n) + 'k\nTax = ' + str(n*5))
    elif 850 >= n > 550:
        print('Income = ' + str(n) + 'k\nTax = ' + str(n*10))
    elif 1250 >= n > 850:
        print('Income = ' + str(n) + 'k\nTax = ' + str(n*15))
    elif 1750 >= n > 1250:
        print('Income = ' + str(n) + 'k\nTax = ' + str(n*20))
    else:
        print('Income = ' + str(n) + 'k\nTax = ' + str(n*25))
```

```python
def parentofdisabledpeople(n):
    if 350 >= n > 0:
        print('Income = ' + str(n) + 'k\nTax = ' + str(n*0.0))
    elif 450 >= n > 350:
        print('Income = ' + str(n) + 'k\nTax = ' + str(n*5.0))
    elif 750 >= n > 450:
        print('Income = ' + str(n) + 'k\nTax = ' + str(n*10))
    elif 1150 >= n > 750:
        print('Income = ' + str(n) + 'k\nTax = ' + str(n*15))
    elif 1650 >= n > 1150:
        print('Income = ' + str(n) + 'k\nTax = ' + str(n*20))
    else:
        print('Income = ' + str(n) + 'k\nTax = ' + str(n*25))


def freedomfighterpeople(n):
    if 475 >= n > 0:
        print('Income = ' + str(n) + 'k\nTax = ' + str(n*0))
    elif 575 >= n > 475:
        print('Income = ' + str(n) + 'k\nTax = ' + str(n*5))
    elif 875 >= n > 575:
        print('Income = ' + str(n) + 'k\nTax = ' + str(n*10))
    elif 1275 >= n > 875:
        print('Income = ' + str(n) + 'k\nTax = ' + str(n*15))
    elif 1775 >= n > 1275:
        print('Income = ' + str(n) + 'k\nTax = ' + str(n*20))
    else:
        print('Income = ' + str(n) + 'k\nTax = ' + str(n*25))


def takeInput():
    print('Enter Income(k): ')
    income = int(input())
    return income
```

```
while True:
    print('\n\033[1m Note: You must insert the values in "k" if you input 100, then The system will automatically count it as 100k.\033[0m\n')
    print('1.General')
    print('2.Woman and age greater than 65')
    print('3.Disabled')
    print('4.Freedom Fighters')
    print('5.Parent of Disabled')
    print('0.Exit')

    a = int(input())
    if a == 1:
        income = takeInput()
        normalpeople(income)
    elif a == 2:
        income = takeInput()
        womenand65people(income)
    elif a == 3:
        income = takeInput()
        disabledpeople(income)
    elif a == 4:
        income = takeInput()
        freedomfighterpeople(income)
    elif a == 5:
        income = takeInput()
        parentofdisabledpeople(income)
    elif a == 0:
        break
    else:
        print('Wrong Input')
```

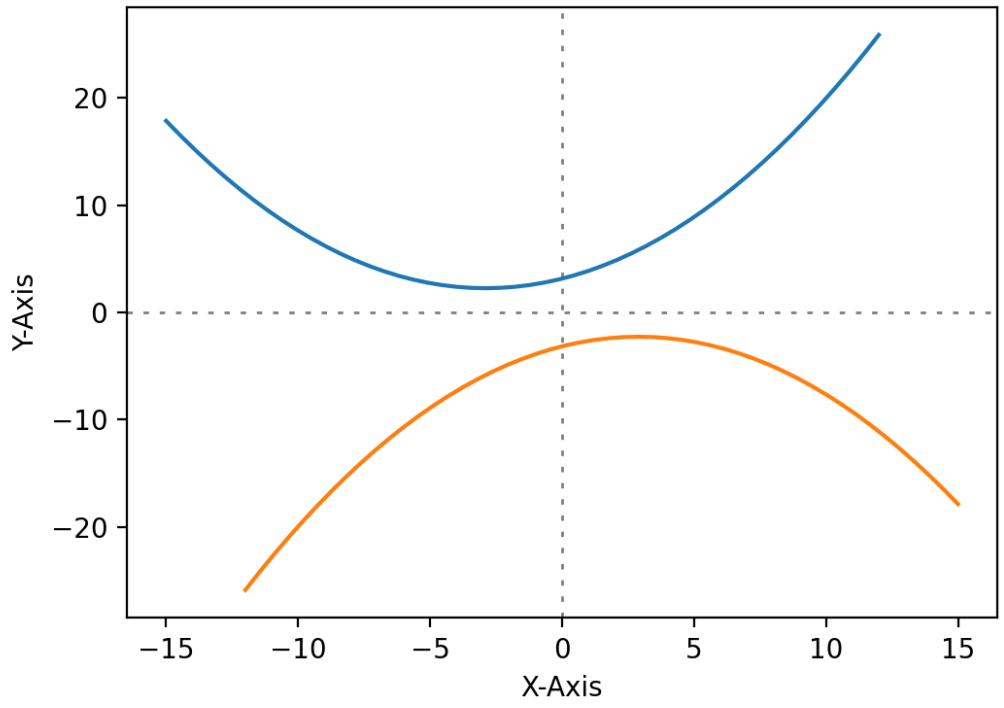
OUTPUT:

```
 Note: You must insert the values in "k" if you input 100, then The system will automatically count it as 100k.

1.General
2.Woman and age greater than 65
3.Disabled
4.Freedom Fighters
5.Parent of Disabled
0.Exit
1
Enter Income(k):
900
Income = 900k
Tax = 13500
```

Problem-2: Draw Graph

```
import matplotlib.pyplot as plt
import numpy as np

# Value of x within a range
valx = np.linspace(-15, 12, num=50)
valx1 = np.linspace(-12, 15, num=50)

# assigning y on the value of x
valy = 0.1065 * valx ** 2 + 0.6164 * valx + 3.1565
valy1 = -(0.1065 * valx1 ** 2) + 0.6164 * valx1 - 3.1565

# plotting graph with the values of x and y
plt.figure(num=0, dpi=200)
plt.axhline(0, color='black', alpha=0.5, dashes=[2, 4], linewidth=1)
plt.axvline(0, color='black', alpha=0.5, dashes=[2, 4], linewidth=1)
plt.xlabel('X-Axis')
plt.ylabel('Y-Axis')
plt.plot(valx, valy)
plt.plot(valx1, valy1)
plt.show()
```

OUTPUT:



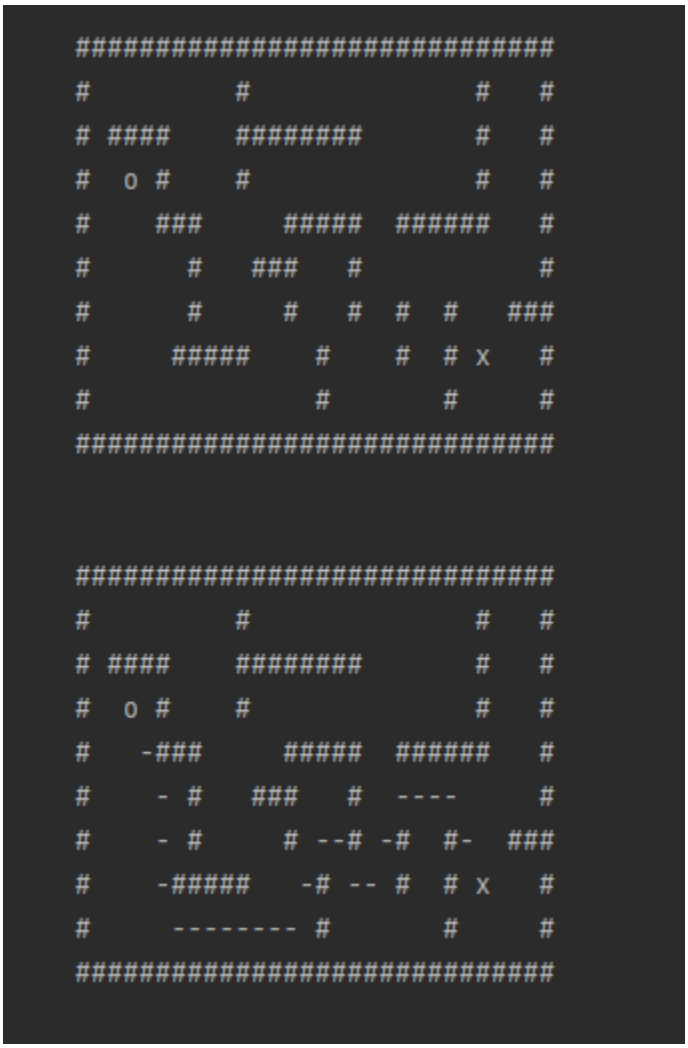
Problem-3: Maze Solver

```
import math
from simpleai.search import SearchProblem, astar


class MazeSolver(SearchProblem):
    def __init__(self, board):
        self.board = board
        self.goal = (0, 0)

        for y in range(len(self.board)):
            for x in range(len(self.board[y])):
                if self.board[y][x].lower() == "o":
                    self.initial = (x, y)
                elif self.board[y][x].lower() == "x":
                    self.goal = (x, y)

        super(MazeSolver, self).__init__(initial_state=self.initial)

    # Define the method that takes actions to arrive at the solution
    def actions(self, state):
        actions = []
        for action in COSTS.keys():
            newx, newy = self.result(state, action)
            if self.board[newy][newx] != "#":
                actions.append(action)

        return actions

    # Based on the action, updating state
    def result(self, state, action):
        x, y = state

        if action.count("up"):
            y -= 1
        if action.count("down"):
            y += 1
        if action.count("left"):
            x -= 1
        if action.count("right"):
            x += 1

        new_state = (x, y)

        return new_state
```

```
    # Check if goal is reached
    def is_goal(self, state):
        return state == self.goal

    # Cost Calculation
    def cost(self, state, action, state2):
        return COSTS[action]

    # Heuristic to arrive at the solution
    def heuristic(self, state):
        x, y = state
        gx, gy = self.goal

        return math.sqrt((x - gx) ** 2 + (y - gy) ** 2)


def mazeprint(maze):
    for y in range(len(maze)):
        for x in range(len(maze[y])):
            if (x, y) == problem.initial:
                print('o', end='')
            elif (x, y) == problem.goal:
                print('x', end='')
            elif (x, y) in path:
                print('-', end='')
            else:
                print(maze[y][x], end='')
        print()


if __name__ == "__main__":
    # maze input
    MAP = """
    ##############################
    #         #              #   #
    # ####    ########       #   #
    #  o #    #              #   #
    #    ###     #####  ######   #
    #      #   ###   #           #
    #      #     #   #  #  #   ###
    #     #####    #    #  # x   #
    #              #       #     #
    ##############################
    """
```

```
# Convert map to a list
print(MAP)
maze_conversion = []
lines = MAP.splitlines()
for line in lines:
    maze_conversion.append(list(line))
maze = maze_conversion
# Define initial cost
cost_regular = 1.0
cost_diagonal = 1.7

# Cost dictionary
COSTS = {
    "up": cost_regular,
    "down": cost_regular,
    "left": cost_regular,
    "right": cost_regular,
    "up left": cost_diagonal,
    "up right": cost_diagonal,
    "down left": cost_diagonal,
    "down right": cost_diagonal,
}

# Create maze solver object and getting problem
problem = MazeSolver(maze)

# For solving problem, running astar to get optimal path
result = astar(problem, graph_search=True)

# Getting path
path = []
y = []
for x in result.path():
    y.append(x[1])
path = y
# Printing maze
mazeprint(maze)
```

OUTPUT:



Problem-4: Puzzle Solve

```
from logpy import *
from logpy.core import lall

people = var()
rules = lall(
    # There are 4 people
    (eq, (var(), var(), var(), var()), people),
    # Steve's car is blue
    (membero, ('Steve', var(), 'blue', var()), people),
    # Person who has a cat lives in Canada
    (membero, (var(), 'cat', var(), 'Canada'), people),
    # Matthew lives in USA
    (membero, ('Matthew', var(), var(), 'USA'), people),
    # The person who has a black car lives in Australia
    (membero, (var(), var(), 'black', 'Australia'), people),
    # Jack has a cat
    (membero, ('Jack', 'cat', var(), var()), people),
    # Alfred lives in Australia
    (membero, ('Alfred', var(), var(), 'Australia'), people),
    # Person who owns the dog lives in France
    (membero, (var(), 'dog', var(), 'France'), people),
    # Who has a rabbit?
    (membero, (var(), 'rabbit', var(), var()), people)
)

solutions = run(0, people, rules)

output = ''
for house in solutions[0]:
    if 'rabbit' in house:
        output = house[0]


print('\nThe owner of the rabbit : ' + output)
attributes = ['Name', 'Pet', 'Color', 'Country']
print('\n' + '\t'.join(attributes))
print('_' * 40)
for item in solutions[0]:
    print('')
    print('\t'.join([str(x) for x in item]))
```

OUTPUT:

```
The owner of the rabbit : Matthew

Name    Pet Color   Country
---------------------------------------

Steve   dog blue    France

Jack    cat ~_9 Canada

Matthew rabbit  ~_11    USA

Alfred  ~_13    black   Australia
```