TRIBHUVAN UNIVERSITY

INSTITUTE OF SCIENCE AND TECHNOLOGY

Central Department of Computer Science and Information Technology

Kirtipur, Kathmandu

Lab No.: 1

A Lab Report on *Implementation of Crisp Sets and Operations*

Submitted by:

Name: Brihat Ratna Bajracharya

Roll No.: 19/075

Submitted to:

Mr. Jagdish Bhatta

Central Department of Computer Science and Information Technology

Date of submission: *2077 Ashwin 3*

LAB 1

Implementation of Crisp Sets and Operations

Implement crisp sets as discussed in the class. Your program should ask user to input the set elements and corresponding membership values. The program should accept the valid membership values only. For the set, use appropriate data structure to store fuzzy element with its corresponding membership value. Also write functions to implement Union, Intersection, Complement and Subset operations over the sets.

Program Code:

```
'''
    Fuzzy Systems Lab 1
    Brihat Ratna Bajracharya (CRN: 19/075)
    CDCSIT
'''

def get_tuple():
    """ Takes element and value from user to form a tuple """
    element = input('   Enter Element: ')
    value = int(input('   Enter Characteristic Functional Value [0,1]: '))
    if value > 1:
        print("    Invalid Value, this element will be discarded.")
        return
    return (element, value)

def get_element(other_set, element):
    """ returns element from other_set having same first value as element """
    for elem in other_set:
        if elem[0] == element[0]:
            return elem
    return None

def get_union(seta, setb):
    """ returns union of seta and setb """
    union_set_list = []

    for elem in seta:
        union_set_list.append([*elem])

    for elem in setb:
        present = False
        if elem[0] in [e[0] for e in seta]:
            present = True
        if present is True:
            old_element = None
            for element in union_set_list:
                if element[0] == elem[0]:
                    old_element = element
                    union_set_list.remove(element)
                    break
```

```
            union_set_list.append([elem[0], max(old_element[1], elem[1])])
        else:
            union_set_list.append([*elem])

    union_set = set()
    for elem in union_set_list:
        union_set.add(tuple(elem))

    return union_set

def get_intersection(seta, setb):
    """ returns intersection of seta and setb """

    intersection_set_list = []

    for elem in seta:
        intersection_set_list.append([*elem])

    for elem in setb:
        present = False
        if elem[0] in [e[0] for e in intersection_set_list]:
            present = True
        if present is True:
            old_element = None
            for element in intersection_set_list:
                if element[0] == elem[0]:
                    old_element = element
                    intersection_set_list.remove(element)
                    break
            intersection_set_list.append([elem[0],
                                          min(old_element[1], elem[1])])
        else:
            intersection_set_list.append([elem[0], 0])

    for elem in intersection_set_list:
        if elem[0] not in [e[0] for e in setb]:
            elem[1] = 0

    intersection_set = set()
    for elem in intersection_set_list:
        intersection_set.add(tuple(elem))

    return intersection_set

def get_complement(seta):
    """ returns complement of a set """
    complement_set = set()

    for elem in seta:
        new_elem_tuple = (elem[0], 1 - elem[1])
        complement_set.add(new_elem_tuple)

    return complement_set

def is_subset(seta, setb):
```

```
    """ returns if seta is subset of setb """
    res = True

    for elema in seta:
        all_element_setb = [elem[0] for elem in setb]
        if elema[0] not in all_element_setb:
            res = False
        else:
            corresponding_element = get_element(setb, elema)
            if elema[1] > corresponding_element[1]:
                res = False

    return res

def main():
    """ Main Function """

    print("\nFS LAB 1 (Crisp Set Basic Operations)")
    print("Brihat Ratna Bajracharya\n19/075")
    print("---------\n")

    print("For Set A")
    set_a_num = int(input(" Enter number of elements: "))

    set_A = set()
    for i in range(set_a_num):
        print("  For Element " + str(i+1))
        new_element = get_tuple()
        if new_element is not None:
            set_A.add(new_element)
        print()

    print("For Set B")
    set_b_num = int(input(" Enter number of elements: "))

    set_B = set()
    for i in range(set_b_num):
        print("  For Element " + str(i+1))
        new_element = get_tuple()
        if new_element is not None:
            set_B.add(new_element)
        print()

    print("\n-----------------------------------\n")

    print("Set A: " + str(set_A) + "\n")
    print("Set B: " + str(set_B) + "\n")

    unionab = get_union(set_A, set_B)
    # print(set_A | set_B)
    print(" Set A union B: " + str(unionab) + "\n")

    intersectionab = get_intersection(set_A, set_B)
    # print(set_A & set_B)
    print(" Set A intersection B: " + str(intersectionab) + "\n")
```

```
    complementa = get_complement(set_A)
    print(" Set A complement: " + str(complementa) + "\n")

    complementb = get_complement(set_B)
    print(" Set B complement: " + str(complementb) + "\n")

    is_seta_subset_of_setb = is_subset(set_A, set_B)
    print(" Set A subset of set B: " + str(is_seta_subset_of_setb) + "\n")

    is_setb_subset_of_seta = is_subset(set_B, set_A)
    print(" Set B subset of set A: " + str(is_setb_subset_of_seta) + "\n")

if __name__ == "__main__":
    main()
    print("DONE.")
```

Output Screenshots

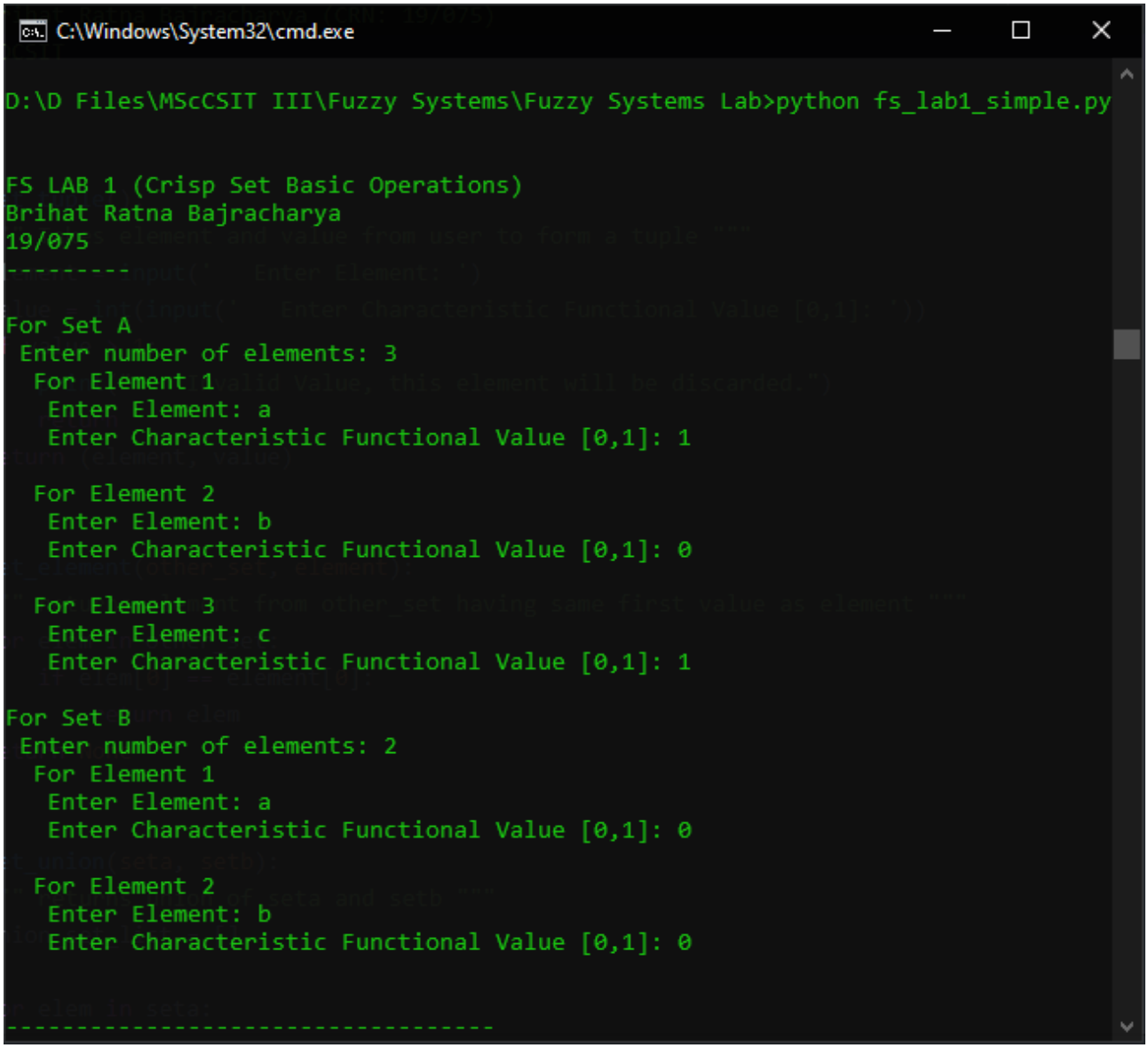

Figure 1: Implementation of Crisp Sets and Operations (1/2)

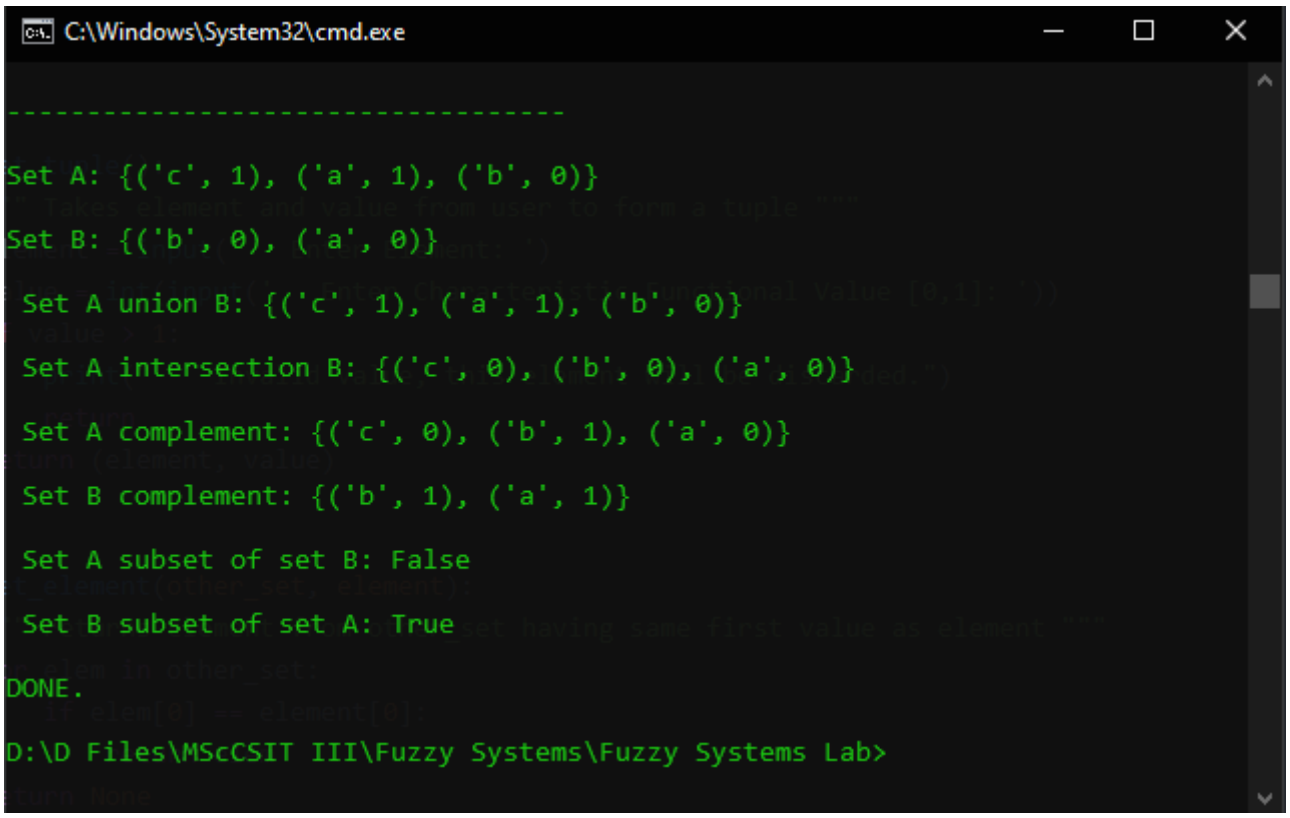

Figure 2: Implementation of Crisp Sets and Operations (2/2)