COM2108: Functional Programming

Assignment 1: *Bags*

Test Report

In this report, there are tests to show that all functions & algorithms are working properly and are implemented properly. Each test case are logically different to test for all possible outcomes for the functions.

Parameters: **bagX:** First bag
bagY: Second bag
Item: A polymorphic element

Functions

1. listToBag - *converts a list of items into a bag type*

- **Test Cases**
  1. Converting an **empty list**
  2. Passing in **one item**
  3. Passing in **multiple items**
  4. Passing in **multiple items** with **multiple occurences**
  5. Passing in an **integer**
  6. Passing in **multiple integers**

Test Case	Input	Results	Status
1	listToBag[]	[]	Pass
2	listToBag['A']	[('A',1)]	Pass
3	listToBag['A','B','C']	[('A',1),('B',1),('C',1)]	Pass
4	listToBag['A','A','B','C','A','C','B']	[('A',3),('B',2),('C',2)]	Pass
5	listToBag[1]	[(1,1)]	Pass
6	listToBag[1,1,2,3,1]	[(1,3),(2,1),(3,1)]	Pass
	listToBag[123,1,2,3,22,68]	[(123,1),(1,1),(2,1),(3,1),(22,1),(68,1)]	Pass

Figure 1.1: Test Results for listToBag function

All the tests that were given were passed which proves that the ***listToBag*** function is polymorphic and is working properly.

2. bagEqual - *takes in two bags and returns true if they are exactly the same*

For the bagEquals test case, in order to prevent from having to convert listToBag for every test case, I have generalised them & created multiple bags to be used for testing as follows:

```
*Bags> emptyBag= listToBag[]
*Bags> emptyBag
[]

*Bags> bagABC = listToBag['A','B','C']
*Bags> bagABC
[('A',1),('B',1),('C',1)]

*Bags> bagABC1 = listToBag['A','B','C']
*Bags> bagABC1
[('A',1),('B',1),('C',1)]
```

```
*Bags> bagCBA = listToBag['C','B','A']
*Bags> bagCBA
[('C',1),('B',1),('A',1)]

*Bags> bagABCDE = listToBag['A','B','C','D','E']
*Bags> bagABCDE
[('A',1),('B',1),('C',1),('D',1),('E',1)]

*Bags> bagMultiple = listToBag['A','B','B','C','C','C']
*Bags> bagMultiple
[('A',1),('B',2),('C',3)]
```

Figure 2.1: Code for generalising the bags used for testing

- **Test Case**
    1. **Both** bags are **empty**
    2. One of the bags are **empty**
    3. Both bags are **exactly identical**
    4. Both bags have same elements but in **different order**
    5. One bag is a **subset** of the other bag
    6. Both bags have same elements but **different number of occurrences**

Test Case	**bagX**	**bagY**	**Expected Results**	**Actual Result**	**Status**
1	emptyBag	emptyBag	True	True	**Pass**
2	emptyBag	bagABC	False	False	**Pass**
3	bagABC	bagABC1	True	True	**Pass**
4	bagABC	bagCBA	True	True	**Pass**
5	bagABC	bagABCDE	False	False	**Pass**
6	bagABC	bagMultiple	False	False	**Pass**

Figure 2.1 - Test Results for bagEqual function

Since the test results for bagEquals are all passed, it shows that the bagEquals functions works perfectly and returns true when the bags have same elements, same number of occurrences, despite having a different order.

3. bagSum - *takes two bags and returns a bag which is the sum of both bags*

```
*Bags> emptyBag= listToBag[]
*Bags> emptyBag
[]

*Bags> bagA = listToBag['A']
*Bags> bagA
[('A',1)]

*Bags> bagABC = listToBag['A','B','C']
*Bags> bagABC
[('A',1),('B',1),('C',1)]
```

```
*Bags> bag123 = listToBag['1','2','3']
*Bags> bag123
[('1',1),('2',1),('3’,1)]

*Bags> bag2345= listToBag['2’,’3’,’4’,’5”]
*Bags> bag2345
[('2',1),('3',1),('4',1),('5',1)]
```

Figure 3.1 - Code for generalising the bags that are to be used for the test cases

- **Test Case**
  1. Both bags are **empty**
  2. One bags is **empty**
  3. Both bags contain same element but **different number of occurrences**
  4. Both bags contain **different elements**

Test Case	bagX	bagY	Result	Status
1	emptyBag	emptyBag	[]	**Pass**
2	emptyBag	bagA	[‘A’]	**Pass**
3	bagA	bagABC	[(‘A’,2),(‘B’,1),(‘C’,1)]	**Pass**
	bag123	bag2345	[(1,1),(2,2),(3,2),(4,1),(5,1)]	**Pass**
4	bag123	bagABC	[(1,1),(2,1),(3,1),(‘A’,1),(‘B’,1),(‘C’,1)]	**Pass**

Figure 3.2 - Test Results for bagSum function

The 5 test cases tests that the function sums up both bags regardless of whether it is of different type or the same type, and whether one or both of the bags are empty. Since all the tests have passed, it is proven that the bagSum function works properly as it should.

4. bagInsert - *Takes in an item and inserts it into the bag*

```
*Bags> emptyBag= listToBag[]
*Bags> emptyBag
[]

*Bags> bagA = listToBag['A']
*Bags> bagA
[('A',1)]

*Bags> bagABC = listToBag['A','B','C']
*Bags> bagABC
[('A',1),('B',1),('C',1)]
```

```
*Bags> bag123 = listToBag['1','2','3']
*Bags> bag123
[('1',1),('2',1),('3',1)]

*Bags> bag2345= listToBag['2','3','4','5"]
*Bags> bag2345
[('2',1),('3',1),('4',1),('5',1)]
```

Figure 4.1 - Code for generalising the bags that are to be used for the test cases

- **Test Case**
  1. Insert **one item** into an **empty bag**
  2. Insert **multiple items** into an **empty bag**
  3. Insert **multiple items** into a **bag with multiple items**
  4. Insert **multiple items** into a bag with **different types of items**
  5. Insert **multiple integers** into a bag of **integers**

Test Case	Item	Bag	Result	Status
1	(‘A’)	emptyBag	[(‘A’,1)]	**Pass**
2	(‘A’)	bagA	[(‘A,2’)]	**Pass**
3	(‘A’’)	bagABC	[('A',1),('B',2),('C',1)]	**Pass**
4	(‘A’)	bag123	[('1',1),('2',1),('3',1),('A',1)]	**Pass**
5	(1)	bag123	[('1',2),('2',1),('3',1)]	**Pass**

Figure 4.2 - Test cases for bagInsert function

Since all test cases are passed, it shows that the bagInsert function works properly for all cases, regardless if the bag has different types or items, is empty or has the same item.

5. bagIntersection - *returns a bag with items that are present in both bags*

```
*Bags> emptyBag = listToBag[]
*Bags> emptyBag
[]

*Bags> bagA = listToBag['A']
*Bags> bagA
[('A',1)]

*Bags> bagAA = listToBag ['A','A']
*Bags> bagAA
[('A',2)]
```

```
*Bags> bagAAA = listToBag['A','A','A']
*Bags> bagAAA
[('A',3)]

*Bags> bagABC = listToBag['A','B','C']
*Bags> bagABC
[('A',1),('B',1),('C',1)]
```

Figure 5.1 - Code for generalising the bags that are to be used for the test cases

- **Test Case**
  1. Two **empty bags**
  2. One **empty bag**
  3. Two bags with **different items**
  4. Two bags with **same items**, but **different number of occurrences**

Test Case	bagX	bagY	Actual Result	Status
1	emptyBag	emptyBag	[]	Pass
2	bagA	emptyBag	[]	Pass
3	bagA	bagAAA	[‘A’,1]	Pass
4	bagA	bagABC	[‘A’,1]	Pass
	bagAA	bagAAA	[‘A’,2]	Pass

Figure 5.2 - Test cases for bagIntersection function

Since all the test cases pass, it is proven that bagIntersection works properly.

Test Results from the Command Prompt

```
*Bags> bagIntersection emptyBag emptyBag
[]

*Bags> bagIntersection bagA emptyBag
[]

*Bags> bagIntersection bagA bagAAA
[('A',1)]

*Bags> bagIntersection bagA bagABC
[('A',1)]

*Bags> bagIntersection bagAA bagAAA
[('A',2)]
```

```
*Bags> listToBag[]
[]

*Bags> listToBag['A']
[('A',1)]

*Bags> listToBag['A','B','C']
[('A',1),('B',1),('C',1)]

*Bags> listToBag['A','A','B','C','A','C','B']
[('A',3),('B',2),('C',2)]

*Bags> listToBag[1]
[(1,1)]

*Bags> listToBag[1,1,2,3,1]
[(1,3),(2,1),(3,1)]

*Bags> listToBag[123,1,2,3,22,68]
[(123,1),(1,1),(2,1),(3,1),(22,1),(68,1)]
```

```
*Bags> bagEqual emptyBag emptyBag
True

*Bags> bagEqual emptyBag bagABC
False

*Bags> bagEqual bagABC bagABC1
True

*Bags> bagEqual bagABC bagCBA
True

*Bags> bagEqual bagABC bagABCDE
```

```
False

*Bags> bagEqual bagABC bagMultiple
False
```

```
*Bags> bagSum emptyBag emptyBag
[]

*Bags> bagSum emptyBag bagA
[('A',1)]

*Bags> bagSum bagA bagABC
[('A',2),('B',1),('C',1)]

*Bags> bagSum bag123 bag2345
[('1',1),('2',2),('3',2),('4',1),('5',1)]

*Bags> bagSum bag123 bagABC
[('1',1),('2',1),('3',1),('A',1),('B',1),('C',1)]
```

```
*Bags> bagInsert ('A') emptyBag
[('A',1)]

*Bags> bagInsert ('A') bagA
[('A',2)]

*Bags> bagInsert ('A') bag123
[('1',1),('2',1),('3',1),('A',1)]

*Bags> bagInsert ('B') bagABC
[('A',1),('B',2),('C',1)]

*Bags> bagInsert ('1') bag123
[('1',2),('2',1),('3',1)]
```