Data Structure: inputFileName.txt

Access Programs: object.code, object.description, object.cost, object.quantity, object.used
Notation to indicate the various data stored in each object (line in data file).

Implementation:

Uses: none

Contstants: none

Variables:

recorded_min_cost: LONG recorded_max_cost: LONG
Represents values that create the domain for getAllItemsInCostRange()

code: STRING description: STRING cost: LONG quantity: INT
Represents the data of the stored objects

object_used: BOOLEAN
Represents whether the object has been iterated through by getAllItemsInCostRange()

Format: (X is positive integer)

Line Number	Object data
1	recorded_min_cost '\t' recorded_max_cost '\n'
X+1	code '\t' description '\t' cost '\t' quantity '\t' object_used

Decision Process:

Our group decided that it was best to contain the data structure for the program within the text file to keep all of the data localized as the scale of the data being used is very small and does not need to be modulated.

The objects in the data file are not sorted in any particular order to keep the complexity of creating the files very low as the file is rewritten often by various modules. This decision was made because even though searching functions suffer higher run times, those functions also contain Boolean values which require the whole file to be searched regardless of what order the objects are in to be obtained.

<u>Module:</u> setItemData

<u>Access Programs:</u> none

<u>Implementation:</u>

<u>Uses:</u> input.txt

<u>Variables</u>

<u>Input:</u>
code: STRING; description: STRING; cost:LONG; quantity: INT;
Represents the information of the new Item to be added to the data file.

<u>Output:</u>
code “\t” description “\t” cost “\t” quantity “\n”
Outputs to input.txt the information of the new Item.

<u>State:</u>
spotFound: BOOLEAN
Represents whether an empty line is found in input.txt.

currentLine: STRING
Stores the current linebeing examine in input.txt

Constants: none

Psudo Code:

```
IF (input.txt not found) DO
    PRINT (“Can’t open input file”)
IF(output.txt not found) DO
    PRINT (“Can’t open output file”)
WHILE (NOT input.txt end of file) DO
    IF (currentLine is empty) DO
        spotFound = true
        PRINT TO outputFile (code “\t” description “\t” cost “\t” quantity “\n”)
    ELSE DO
        PRINT TO outputFIle(currentLine “\n”)
IFNOT spotFound)
    PRINT TO outputFile (code “\t” description “\t” cost “\t” quantity “\n”)
CLOSE inputFIle
CLOSE outputFIle
DELETE input.txt
RENAME output.txt to input.txt
```

Test Report:

TEST CASE (item: code, descritption,cost, quantity)input.txt	code:String descrption: String cost: LONG quantity:INT	spotFound: BOOLEAN	currentLine: STRING	Input.txt’	Result
code1, des1,123,1 code2,des2, 45, 2 code3,des3, 6,3	code4, des4, 101,4	FALSE	“ ”	code1, des1,123,1 code2,des2, 45, 2 code3,des3, 6,3 code4,des4,101,4	PASS
code1, des1,123,1 code2,des2, 45, 2 \n code3,des3, 6,3	code4, des4, 101,4	TRUE	\n	code1, des1,123,1 code2,des2, 45, 2 code4,des4,101,4 code3,des3, 6,3	PASS
EMPTY	Code1, description,1,1	FALSE	“ ”	Code1,description1,1	PASS

Module: setRemoveItem()

Access Programs: none

Implementation:

Uses: inputFileName.txt,

Variables

Input:
code: STRING
Represents a short description of the item being searched for

Output: none

State:
code2
Represents the code of an object in the data file, used to compare with the input description.

Constants: inputFileName: CHAR[]
Represents the name of the text file used by this module.

Psudo code:

```
FOR (each object in data file) DO
    read the object
    IF (code == code2)
        remove object
        RETURN
RETURN
```

Test Report:

TEST CASE (object.code)	code IN	Data file after call	Result
item1 item2 item3 item4	item1	item2 item3 item4	pass
	item2	item1 item3 item4	pass
	item5	item1 item2 item3	pass

		item4	

<u>Module:</u> getItemCode()

<u>Access Programs:</u> none

<u>Implementation:</u>

<u>Uses:</u> inputFileName.txt

<u>Variables</u>

<u>Input:</u>
description: STRING
Represents a short description of the item being searched for

<u>Output:</u>
code: STRING
Represents the ID code of the item being searched for.

<u>State:</u>
description2
Represents the description of an object in the data file, used to compare with the input description.

<u>Constants:</u> inputFileName: CHAR[]
Represents the name of the text file used by this module.

<u>Psudo code:</u>

```
FOR (each object in data file) DO
        read the object
        IF (description == description2)
                RETURN object.code
RETURN DNE
```

<u>Function table:</u>

		code
description == description2		object.code
description != description2	object.description2 is last object	DNE
	object.description2 is not last object	NO CHANGE

Test Report:

TEST CASE (object.code, object.description)	Description IN	Code OUT	Result
item1, a car item2, a box item3, a car item4, a dog	a box	item2	pass
	a car	item1	pass
	a walrus	DNE	pass

Module: getItemData()

Access Programs: none

Implementation:

Uses: inputFileName.txt

Variables

Input:
code: STRING
Represents a short description of the item being searched for

Output:
send: ARRAY<STRING>
An array containing the description, cost, and quantity of an the object with matching code input.

State:
code2
Represents the code of an object in the data file, used to compare with the input code.

Constants: inputFileName: CHAR[]
Represents the name of the text file used by this module.

Psudo code:

```
FOR (each object in data file) DO
    read the object
    IF (code == code2)
        RETURN send[object.description, object.cost, object.quantity]
RETURN DNE
```

Function table:

		send
code == code2		[object.description, object.cost, object.quantity]
code != code2	object.code2 is last object	DNE
	object.code2 is not last object	NO CHANGE

Test Report:

TEST CASE (object.code, object.description, object.cost, object.quantity)	code IN	send OUT	Result
item1, aaaa, 3020, 67 item2, bbbb, 9999, 90 item3, cccc, 0001, 100 item4, dddd, 8763, 0	item1	[aaaa, 3020, 67]	pass
	item2	[bbbb, 9999, 90]	pass
	item5	DNE	pass

Module: getItemCost()

Access Programs: none

Implementation:

Uses: inputFileName.txt

Variables

Input:
code: STRING
Represents the ID code of the object being searched for

Output:
cost: LONG
Represents the cost of the object being searched for.

State:
code2
Represents the code of an object in the data file, used to compare with the input code.

Constants: inputFileName: CHAR[]
Represents the name of the text file used by this module.

Psudo code:

```
FOR (each object in data file) DO
        read the object
        IF (code == code2)
                RETURN object.cost
RETURN DNE
```

Function table:

		cost
code == code2		object.cost
code != code2	object.code2 is last object	DNE
	object.code2 is not last object	NO CHANGE

Test Report:

TEST CASE (object.code, object.cost)	code IN	cost OUT	Result
item1, 3020 item2, 9999 item3, 0001 item4, 8763	item1	3020	pass
	item2	9999	pass
	item5	DNE	pass

<u>Module:</u> setAddQuantity

<u>Access Programs:</u> none

<u>Implementation:</u>

<u>Uses:</u> input.txt

<u>Variables</u>

<u>Input:</u>
code: STRING; increment: INT;
Represents the code of the Item which will have its quantity value increased by increment.

<u>Output:</u>
code “\t” description “\t” cost “\t” (quantity+increment) “\n”
Outputs to input.txt the updated information of the Item.

<u>State:</u>
tempCode:string
Stores the current code of the line which the program is reading from input.txt
Description:string
Stores the current description of the line which the program is reading from input.txt

cost: LONG
Stores the current cost of the line which the program is reading from input.txt

quantity: INT
Stores the current quantity of the line which the program is reading from input.txt

Constants: none

Psudo Code:

```
IF (input.txt not found) DO
    PRINT (“Can’t open input file”)
IF(output.txt not found) DO
    PRINT (“Can’t open output file”)
WHILE (NOT input.txt end of file) DO
    tempCode = GET FROM input.txt (code)
    IF (NOT(tempCode==”\n”)) DO
        Description = GET FROM input.txt (description)
        cost= GET FROM input.txt (cost)
        quantity= GET FROM input.txt (quantity)
            IF(Description NOT = “ “)DO
                PRINT TO output.txt (tempcode “\t” Description”\t” cost  “\t”
                IF(tempCode==Code) DO
                    quantity’ = ‘quantity + increment
            PRINT TO output.txt (quantity “\n”)
CLOSE input.txt
CLOSE output.txt
DELETE input.txt
RENAME output.txt to input.txt
```

<u>Test Report:</u>

TEST CASE (item: code, descritption,cost, quantity)input.txt	code:String increment:INT	Tempcode:String, Description:String, Cost:LONG,quantity’: INT	Input.txt’	Result
code1, des1,123,1 code2,des2, 45, 2 code3,des3, 6,3	code3,4	Code3,des3,6,7	code1, des1,123,1 code2,des2, 45, 2 code3,des3, 6,7	PASS
code1, des1,123,1 code2,des2, 45, 2 \n code3,des3, 6,3	code2, 10	Code2,des2,45, 12	code1, des1,123,1 code2,des2, 45, 12 \n code3,des3, 6,3	PASS
EMPTY	Code2,10	EMPTY	EMPTY	PASS

Module: setDeleteQuantity

Access Programs: none

Implementation:

Uses: input.txt

Variables

Input:
code: STRING; decrement: INT;
Represents the code of the Item which will have its quantity value decreased by decrement.

Output:
code "\t" description "\t" cost "\t" quantity "\n"
Outputs to input.txt the updated information of the Item.

State:
tempCode:string
Stores the current code of the line which the program is reading from input.txt
Description:string
Stores the current description of the line which the program is reading from input.txt

cost: LONG
Stores the current cost of the line which the program is reading from input.txt

quantity: INT
Stores the current quantity of the line which the program is reading from input.txt

Constants: none

Pseudo Code:

```
IF (input.txt not found) DO
    PRINT ("Can't open input file")
IF(output.txt not found) DO
    PRINT ("Can't open output file")
WHILE (NOT input.txt end of file) DO
    tempCode = GET FROM input.txt (code)
    IF (NOT(tempCode=="\n")) DO
        Description = GET FROM input.txt (description)
        cost= GET FROM input.txt (cost)
        quantity= GET FROM input.txt (quantity)
            IF(Description NOT = " ")DO
                PRINT TO output.txt (tempcode "\t" Description"\t" cost  "\t"
                IF(tempCode==Code) DO
                    quantity' = 'quantity - decrement
                    PRINT TO output.txt (quantity "\n")
CLOSE input.txt
CLOSE output.txt
DELETE input.txt
RENAME output.txt to input.txt
```

<u>Test Report:</u>

TEST CASE (item: code, descritption,cost, quantity)input.txt	code:String decrement:INT	Tempcode:String, Description:String, Cost:LONG,quantity': INT	Input.txt'	Result
code1, des1,123,1 code2,des2, 45, 2 code3,des3, 6,3	code3,2	Code3,des3,6,1	code1, des1,123,1 code2,des2, 45, 2 code3,des3, 6,1	PASS
code1, des1,123,1 code2,des2, 45, 2 \n code3,des3, 6,3	code2, 10	Code2,des2,45,-8	code1, des1,123,1 code2,des2, 45, 12 \n code3,des3, 6,-8	PASS
EMPTY	Code2,10	EMPTY	EMPTY	PASS
code1, des1,123,1 code2,des2, 45, 2 \n code3,des3, 6,3	Code7,des7,10,1	EMPTY/NULL	code1, des1,123,1 code2,des2, 45, 2 code3,des3, 6,3	PASS

Module: getAllItemsSorted()

Access Programs: none

Implementation:
Uses: inputFileName.txt

Variables

Input:
None.

Output:
ARRAY(<STRING8>)
Outputs a sorted array containing the codes of all items in the list.

Pseudo code:

```
input =  fileStream (filename)
array = string array ()

while(!eof)
  if input.ReadLine() != blank
                        output = output + input.ReadLine()

//Perform QuickSort on array
if length(array) ≤ 1
return array
select and remove a pivot value 'pivot' from 'array'
create empty lists 'less' and 'greater'
for each 'x' in 'array'
if 'x' ≤ 'pivot' then append 'x' to 'less'
else append 'x' to 'greater'
return concatenate(quicksort('less'), 'pivot', quicksort('greater'))
```

Module: getAllItemsInCostRange()

Access Programs: none

Implementation:

Uses: inputFileName.txt, setRemoveItem(), setItemData()

Variables

Input:
min_cost: LONG, max_cost: LONG
Represents the domain of the objects to be output with respect to their cost. Assumes min_cost < max_cost.

Output:
product: ARRAY(<STRING>,<BOOLEAN>)
Outputs a string containing data pertaining to a single object within the domain min_cost < (object cost) max_cost and a Boolean value indicating if there are more objects within the domain remaining in the data file

State:
object.used: BOOLEAN
Represents whether the object has already been previously used in a prior call

recorded_min_cost: LONG, recorded_max_cost: LONG
Represents current boundaries being used, input values that vary from these values trigger a reset to all object.used values to FALSE.

Constants: inputFileName: CHAR[]
Represents the name of the text file used by this module

Psudo code:

```
IF (min_cost == recorded_min_cost and  max_cost == recorded_max_cost) DO
    FOR (each object in data file) DO
        read the object
        IF ((min_cost <= (current_object.cost) <= max_cost) AND (object.used == FALSE)) DO
            IF (product[0] contains an object) DO
                product[1] = FALSE
                RETURN product
            ELSE DO
                product[1] = TRUE
            current_object.used = TRUE
            set product[0] to object
        END FOR
```

```
RETURN product
ELSE DO
    recorded_min_cost = min_cost;  recorded_max_cost =  max_cost
    FOR (each object in data file) DO
        object used = FALSE
    FOR (each object in data file) DO
        read the object
        IF ((min_cost <= (object cost) <= max_cost) AND (object_used == FALSE)) DO
            IF (product[0] contains an object) DO
                product[1] = FALSE
                RETURN product
            ELSE DO
                product[1] = TRUE
            object_used = TRUE
            set product[0] to object
        END FOR
RETURN product
```

Function table:

		product[0]	product[1]
min_cost <= (current_object.cost) <= max_cost AND (current_object.used == FALSE	product[0] contains an object	NO CHANGE	FALSE
	product[0] does not contains an object	current_object	TRUE
ELSE		NO CHANGE	NO CHANGE
min_cost <= (current_object.cost) <= max_cost AND (current_object.used == FALSE	product[0] contains an object	NO CHANGE	FALSE
	product[0] does not contains an object	current_object	TRUE
ELSE		NO CHANGE	NO CHANGE

Test Report:

TEST CASE (object.code, object.cost, object.used	min_cost IN, max_cost IN	recorded_min_cost, recorded_max_cost	object.used (through iteration)	product OUT	Result
Item1, 3, FALSE Item2, 7, FALSE Item3, 5,	1, 9	0, 0	FALSE, FALSE, FALSE, FALSE, FALSE, FALSE	(Item1, FALSE)	pass

FALSE Item4, 50, FALSE Item5, 2, FALSE Item6, 10, FALSE	1, 9	1, 9	TRUE, FALSE, FALSE, FALSE, FALSE, FALSE	(Item2, FALSE)	pass
	1, 9	1, 9	TRUE, TRUE, FALSE, FALSE, FALSE, FALSE	(Item3,FALSE)	pass
	1, 9	1, 9	TRUE, TRUE, TRUE, FALSE, FALSE, FALSE	(Item5, TRUE)	pass
	1, 9	1, 9	TRUE, TRUE, TRUE, FALSE, TRUE, FALSE	error	Fail – exception when no possible objects remain not handled
	10, 100	1, 9	FALSE, FALSE, FALSE, FALSE, FALSE, FALSE	(Item4, FALSE)	pass