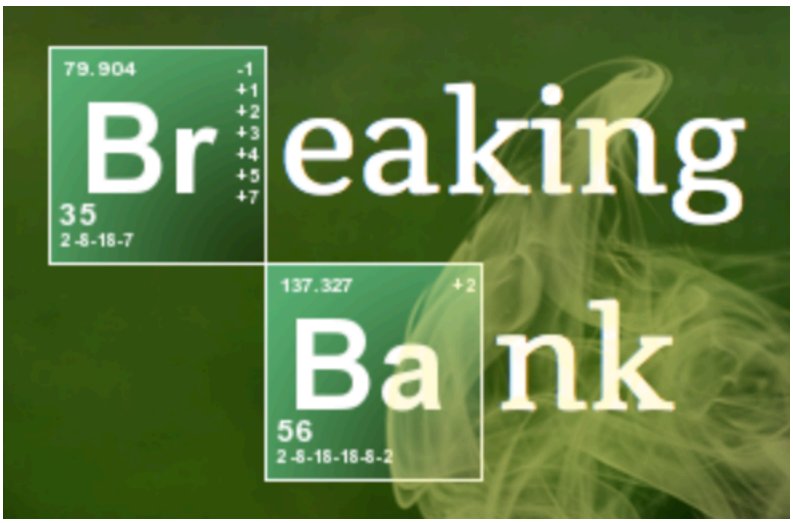


CISC 327 Assignment #5

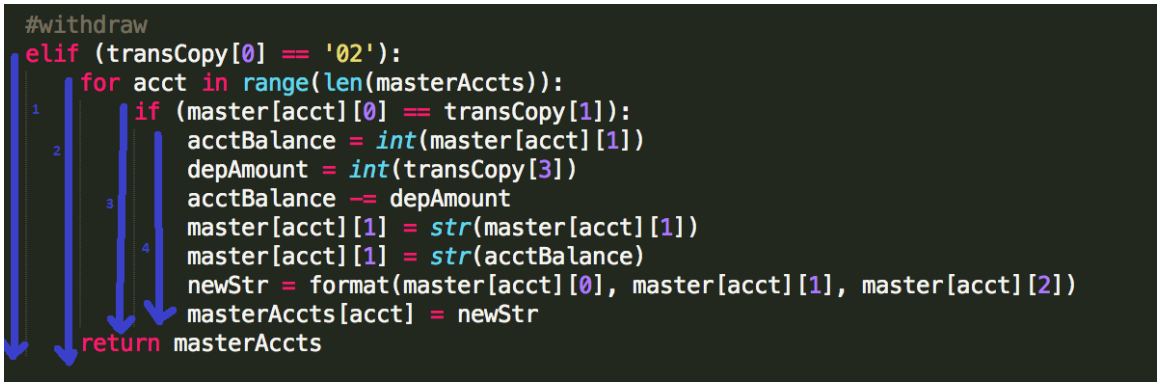
Scott Wallace 10051890
Brad Guner 10059112

Withdraw

Source Code

```
elif (transCopy[0] == '02'):
       for acct in range(len(masterAccts)):
              if (master[acct][0] == transCopy[1]):
                     acctBalance = int(master[acct][1])
                     depAmount = int(transCopy[3])
                     acctBalance -= depAmount
                     master[acct][1] = str(master[acct][1])
                     master[acct][1] = str(acctBalance)
                     newStr = format(master[acct][0], master[acct][1],
master[acct][2])
                     masterAccts[acct] = newStr
return masterAccts
```

Analysis of Test Cases for Basic Block Testing



We have 4 basic blocks to cover for our Withdrawals in the back end.

Block	transCopy[0]	masterAccts	master[acct][0]	transCopy[1]	Test
1	02	empty	empty	000001	1
2	02	1 account	000001	000002	2
3	02	1 account	000001	000001	3
4	02	1 account	000001	000001	

Actual Test Inputs for each Case

Test 1
Merged Transactions file:
02_000001_BBBBBB_00000000_NNNNNNNNNNNNNN

Master Accounts file: Empty

Test 2
Merged Transactions file:
02_000002_BBBBBB_00000100_NNNNNNNNNNNNNN

Master Accounts file:
000001_00001000_ Bob

Test 3
Merged Transactions file:
02_000001_BBBBBB_00000100_Bob

Master Accounts file:
000001_00001000_ Bob

Test Report

TEST #	Results	Failure (Yes/No)	Analysis
1	Empty Master accounts, and empty valid accounts files	No	Test 1 gave the correct output for the input
2	Printed incorrect master accounts, and valid accounts	Yes	Cause: incorrect format function in backend, leading to the return of a hard-coded string
3	Printed incorrect master accounts, and valid accounts	Yes	Cause: incorrect format function in backend, leading to the return of a hard-coded string
2	Incorrect output	Yes	Incorrect merged transactions file
3	Correct output, and updated master and valid accounts	No	Test 3 gave the correct output for the input on the second run
2	Correct output and updated master and valid accounts	No	Test 2 gave the correct output for the input on the third run

How Withdraw tests were executed

Withdraw shell script

```
#!/bin/bash

cd testsuite1
python breakingbank-backend.py
```

cd ..
cd testsuite2
python breakingbank-backend.py

cd ..
cd testsuite3
python breakingbank-backend.py

The way we performed the tests on withdraw was running this shell script which went through each of our test suites and ran our back end. Each suite contained a master accounts file and a merged transactions file, as per the specified inputs.

Delete

Source Code

```
elif (transCopy[0] == '05'):  #delete _ do decision testing, need a
test case it evaluate every if both ways
            acctNum = str(transCopy[1])
            transAcctName = str(transCopy[4])
            for acct in range(len(master)):
                  if (acctNum == master[acct][0]):
                        acctBalance = master[acct][1]
                        if (acctBalance == '00000000'):
                              acctName = str(master[acct][2])
                              if (transAcctName == acctName):
                                    masterAccts =
masterAccts.remove(masterAccts[acct])
                              else:
                                    throwError()
                        else:
                              throwError()
            return masterAccts
```

Analysis of Test Cases for Decision Testing

```
elif (transCopy[0] == '05'):
    acctNum = int(trans[1])
    transAcctName = trans[4]
    for acct in range(master):
1.      if (acctNum == acct[0]):
            acctBalance = acct[1]
2.          if (acctBalance == 0):
                acctName = acct[2]
3.              if (transAcctName == acctName):
                    masterAccts.remove(masterAccts[acct])
                else:
                    throwError()
            else:
                throwError()
    return masterAccts
```

(Above image, contains the contents of the starting code)

There are 3 decisions made in the block, thus we have 3 decision test cases.

Decision	masterAccts	master[acct][0]	transCopy[1]	Test
1	1 account	000001	000002	1
2	1 account	000001	000001	2
3	1 account	000001	000001	3
1	1 account	000001	000001	4
2	1 account	000001	000001	5
3	1 account	000001	000001	6

Decision	transCopy[4]	acct[1]	acct[2]	Test
1	aaaaaaaaaaaaaaa	00000000	aaaaaaaaaaaaaaa	1
2	aaaaaaaaaaaaaaa	00000001	aaaaaaaaaaaaaaa	2
3	aaaaaaaaaaaaaab	00000000	aaaaaaaaaaaaaaa	3
1	aaaaaaaaaaaaaaa	00000000	aaaaaaaaaaaaaaa	4
2	aaaaaaaaaaaaaaa	00000000	aaaaaaaaaaaaaaa	5
3	aaaaaaaaaaaaaaa	00000000	aaaaaaaaaaaaaaa	6

Please read the 2 tables as if they were connected.

Actual Test Inputs for each Case

Test 1
Merged Transactions file:
05_000002_BBBBBB_00000000_aaaaaaaaaaaaaaa

Master Accounts file:
000001_00000000_aaaaaaaaaaaaaaa

Test 2
Merged Transactions file:
05_000001_BBBBBB_00000000_aaaaaaaaaaaaaaa

Master Accounts file:
000001_00000001_ aaaaaaaaaaaaaaa

Test 3
Merged Transactions file:
05_000001_BBBBBB_00000000_aaaaaaaaaaaaaab

Master Accounts file:
000001_00000001_aaaaaaaaaaaaaaa

Test 4
Merged Transactions file:
05_000001_BBBBBB_00000000_aaaaaaaaaaaaaaa

Master Accounts file:
000001_00000000_aaaaaaaaaaaaaaa

Test 5
Merged Transactions file:
05_000001_BBBBBB_00000000_aaaaaaaaaaaaaaa

Master Accounts file:
000001_00000000_aaaaaaaaaaaaaaa

Test 6
Merged Transactions file:
05_000001_BBBBBB_00000000_aaaaaaaaaaaaaaa

Master Accounts file:
000001_00000000_aaaaaaaaaaaaaaa

Test Report

TEST #	Results	Failure (Yes/No)	Analysis
1	Empty Master accounts, and empty valid accounts file	No	Gave correct output, but did not technically do anything, test case was altered after this result
2	1 account in Master accounts, account has a non-zero balance	Yes	Wrote to valid accounts file, should have thrown fatal error instead, but a comparison on mismatched types was occurring, bug in code was fixed
3	1 account in Master accounts, transaction account name and Master account name were different	Yes	Wrote to valid accounts file, did not throw fatal error because of a comparison on mismatched types, bug in code was fixed
1	Test case altered to contain different account number from transaction account number	No	Test 1 gave the correct output for the input
2	1 account in Master accounts, account has a non-zero balance	No	Test 2 gave a fatal error, passed test
3	1 account in Master accounts, account has a non-zero balance	No	Test 3 gave a fatal error, passed test
4	Added to test opposite outcome of decision 1. 1 account in Master accounts file, account number is same as transaction account number	No	Test 4 correctly deleted the account from master accounts file
5	Added to test opposite outcome of decision 2. 1 account in Master accounts file has zero balance	No	Test 5 correctly deleted the account from master accounts file
6	Added to test outcome of decision	No	Test 6 correctly deleted the account from master accounts file

	3. 1 account in Master accounts file has same name as transaction account name		

How delete tests were executed

Delete shell script

```
#!/bin/bash

cd testsuite1
echo "testing suite 1"
python breakingbank-backend.py

cd ..
cd testsuite2
echo "testing suite 2"
python breakingbank-backend.py

cd ..
cd testsuite3
echo "testing suite 3"
python breakingbank-backend.py

cd ..
cd testsuite4
echo "testing suite 4"
python breakingbank-backend.py

cd ..
cd testsuite5
echo "testing suite 5"
python breakingbank-backend.py

cd ..
cd testsuite6
echo "testing suite 6"
python breakingbank-backend.py
```

The way we performed the tests on delete was running this shell script which went through each of our test suites and ran our back end. Each suite contained a master accounts file and a merged transactions file, as per the specified inputs.

Full Back-end source code

```
import sys

def transaction(masterAccts,trans):
	#take trans, split by _ into list
	transCopy = trans.split('_') #[CC, AAAAAA, BBBBBB, MMMMMMMM, NNNNNNNNNNNNNNN]
	master = []
	for i in range(len(masterAccts)):
		master.append(masterAccts[i])
	for i in range(len(master)):
		master[i] = master[i].split('_')

	print transCopy[0]
	if (transCopy[0] == '01'):    #deposit
		for acct in range(len(masterAccts)):
			if (master[acct][0] == transCopy[1]):
				acctBalance = int(master[acct][1])
				depAmount = int(transCopy[3])
				acctBalance += depAmount
				master[acct][1] = str(master[acct][1])
				master[acct][1] = str(acctBalance)
				newStr = format(master[acct][0], master[acct][1],
master[acct][2])
				masterAccts[acct] = newStr
		return masterAccts

	#withdraw
	elif (transCopy[0] == '02'):
		for acct in range(len(masterAccts)):
			if (master[acct][0] == transCopy[1]):
				acctBalance = int(master[acct][1])
				depAmount = int(transCopy[3])
				acctBalance -= depAmount
				master[acct][1] = str(master[acct][1])
				master[acct][1] = str(acctBalance)
				newStr = format(master[acct][0], master[acct][1],
master[acct][2])
				masterAccts[acct] = newStr
		return masterAccts

	elif (transCopy[0] == '03'):  #transfer
		for acct in range(len(masterAccts)):
			if (master[acct][0] == transCopy[1]):
				for anotherAcct in range(len(masterAccts)):
					if (master[anotherAcct][0] == transCopy[2]):
						recAcctBalance = int(master[acct][1])
						transAcctBalance =
int(master[anotherAcct][1])
						transAmt = int(transCopy[3])
						recAcctBalance += transAmt
						transAcctBalance -= transAmt
						master[acct][1] = str(master[acct][1])
						master[anotherAcct][1] =
str(master[acct][1])
						newStrFirstAcct = format(master[acct][0],
master[acct][1], master[acct][2])
						masterAccts[acct] = newStr
						newStr = format(master[anotherAcct][0],
master[anotherAcct][1], master[anotherAcct][2])
						masterAccts[anotherAcct] = newStr
		return masterAccts

	elif (transCopy[0] == '04'):  #create
		temp = 0
		acctNum = int(transCopy[1])
		newStr = format(transCopy[1], transCopy[3], transCopy[4])
		for acct in range(len(master)):
			if (accct[0] != acctNum):
				first = int(master[acct][0])
```

```
                                if (acct + 1 <= range(master)):
                                        second = int(master[acct + 1][0])
                                else:
                                        second = 'None'
                                if (accNum < first):
                                        masterAccts.insert(acct - 1, newStr)
                                elif (accNum > first and accNum < second):
                                        masterAccts.insert(acct, newStr)
                                elif (accNum > first and second == 'None'):
                                        masterAccts.insert(acct, newStr)
                        else:
                                throwError()
                return masterAccts

        elif (transCopy[0] == '05'):  #delete _ do decision testing, need a test case it
evaluate every if both ways
                acctNum = str(transCopy[1])
                transAcctName = str(transCopy[4])
                for acct in range(len(master)):
                        if (acctNum == master[acct][0]):
                                acctBalance = master[acct][1]
                                if (acctBalance == '00000000'):
                                        acctName = str(master[acct][2])
                                        if (transAcctName == acctName):
                                                masterAccts =
masterAccts.remove(masterAccts[acct])
                                        else:
                                                throwError()
                                else:
                                        throwError()
                return masterAccts

        elif (transCopy[0] == '00'):
                return masterAccts

def format(num, balance, name):
        string = str(num) + "_" + str(balance) + "_" + str(name)
        return string

def writeNewMasterAccounts(list):
        f = open('./masteraccounts.txt','w')
        for i in list:
                f.write(i + "\n")
        f.close()
        return 0

def writeNewValidAccounts(list):
        f = open('./validaccounts.txt','w')
        for i in list:
                #wrong doesnt write it correctly
                f.write(i + "\n")
        f.close()
        return 0

def throwError():
        sys.exit('Fatal Error')

def main_program():
        #open master accounts
        masteraccts = []
        f = open('./masteraccounts.txt')
        masteraccts = f.readlines()
        for x in range(len(masteraccts)):
                masteraccts[x] = masteraccts[x].strip()
        f.close()
        #open merged transaction file
        mergedtrans= []
        f = open('./mergedtransactions.txt')
        mergedtrans = f.readlines()
        for x in range(len(mergedtrans)):
                mergedtrans[x] = mergedtrans[x].strip()
```

```
        f.close()
        #for all transactions update the master accounts file
        for i in mergedtrans:
                masteraccts = transaction(masteraccts,i)
        #writes output files
        writeNewValidAccounts(masteraccts)
        writeNewMasterAccounts(masteraccts)
        return 0

main_program()
```