Project 4

CSCI 140

Jin Y Choi
Email - jchoi101@student.mtsac.edu

December 3rd, 2018

Development Environment
System - PC Ubuntu 18.04
Compiler: CLion

Table of Contents

Notes/Extra Credit

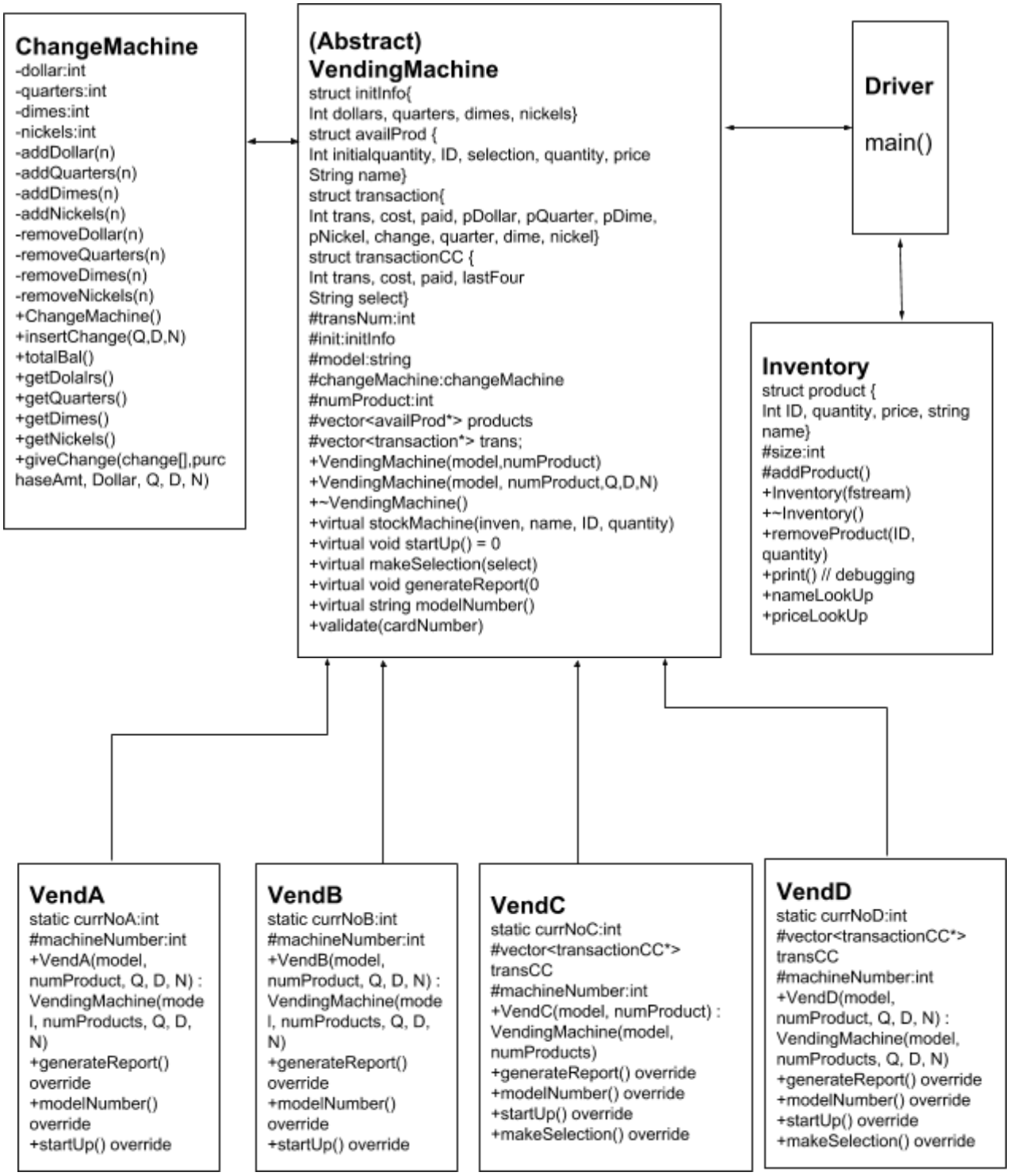
The status of the program is complete and fully implemented. One issue with the program is that the model A and B actually function the same way, the only difference has been that when model A is selected, the inserted money is typed in as 100 0. If you were to input change values, the machine would still work, although that is not what was originally intended for model A.

Thorough exception handling and input validation is not complete. Although I have implemented quite a lot of validation and handling, it is most likely buggy for mis inputs that would load the buffer with bad input.

This project was an incredibly helpful learning experience, the number of errors and bugs I ran into while coding, and all the original ideas that failed to work served as a good applied learning experience in C++. The main issue I originally ran into was the issue of creating an object and attempting to instantiate vectors of actual objects. The compiler threw all kinds of bugs, the most common of which were bad_alloc and SIGSEGV. Creating vectors of pointers, utilizing dynamic memory for cases and doing proper deletes in deconstructors fixed this issue.

I've chosen to implement model 100B and 100D as extra credit. Test Case 1 below shows model B and D in operation.

UML Diagram


ChangeMachine
-dollar:int
-quarters:int
-dimes:int
-nickels:int
-addDollar(n)
-addQuarters(n)
-addDimes(n)
-addNickels(n)
-removeDollar(n)
-removeQuarters(n)
-removeDimes(n)
-removeNickels(n)
+ChangeMachine()
+insertChange(Q,D,N)
+totalBal()
+getDolalrs()
+getQuarters()
+getDimes()
+getNickels()
+giveChange(change[],purchaseAmt, Dollar, Q, D, N)
(Abstract)
VendingMachine
struct initInfo{
Int dollars, quarters, dimes, nickels}
struct availProd {
Int initialquantity, ID, selection, quantity, price
String name}
struct transaction{
Int trans, cost, paid, pDollar, pQuarter, pDime, pNickel, change, quarter, dime, nickel}
struct transactionCC {
Int trans, cost, paid, lastFour
String select}
#transNum:int
#init:initInfo
#model:string
#changeMachine:changeMachine
#numProduct:int
#vector<availProd*> products
#vector<transaction*> trans;
+VendingMachine(model,numProduct)
+VendingMachine(model, numProduct,Q,D,N)
+~VendingMachine()
+virtual stockMachine(inven, name, ID, quantity)
+virtual void startUp() = 0
+virtual makeSelection(select)
+virtual void generateReport(0
+virtual string modelNumber()
+validate(cardNumber)
Driver
main()
Inventory
struct product {
Int ID, quantity, price, string name}
#size:int
#addProduct()
+Inventory(fstream)
+~Inventory()
+removeProduct(ID, quantity)
+print() // debugging
+nameLookUp
+priceLookUp
VendA
static currNoA:int
#machineNumber:int
+VendA(model, numProduct, Q, D, N) : VendingMachine(model, numProducts, Q, D, N)
+generateReport() override
+modelNumber() override
+startUp() override
VendB
static currNoB:int
#machineNumber:int
+VendB(model, numProduct, Q, D, N) : VendingMachine(model, numProducts, Q, D, N)
+generateReport() override
+modelNumber() override
+startUp() override
VendC
static currNoC:int
#vector<transactionCC*> transCC
#machineNumber:int
+VendC(model, numProduct) : VendingMachine(model, numProducts)
+generateReport() override
+modelNumber() override
+startUp() override
+makeSelection() override
VendD
static currNoD:int
#vector<transactionCC*> transCC
#machineNumber:int
+VendD(model, numProduct, Q, D, N) : VendingMachine(model, numProducts, Q, D, N)
+generateReport() override
+modelNumber() override
+startUp() override
+makeSelection() override

PseudoCode (Extra Credit Model B Special Case)

The Algorithm for giving back change utilizes a “greedy algorithm” for almost all cases. A greedy algorithm can be thought up as a series of steps for which each interval attempts to use the best possible option that gets you closest to your goal. Specifically for this project, when giving back change, the greedy algorithm would use as many quarters as possible first, followed by dimes, followed by nickels.
However, out of all the cases, when you have only quarters and dimes left, there exist cases where the greedy method does not work (ie. 55 cents with 2 Q and 4 D). For these cases the algorithm was designed this way:

```
Int smaller - change/25 < quarters ? change/25 : quarters;
If the change value is an even number (ending in 0 in our project)
        smaller must be even (even change - even quarters = number evenly divisible by 10)
Else if the change value is an odd number (ending in 5 in our project)
        Smaller must be odd *odd change - odd quarters = number even divisible by 10)
If (change - smaller * 25)/10 is less than or equal to the number of dimes in the machine
        Transaction is successful, remove smaller # of quarters, and rest dimes
Else
        Insufficient change
```

Source Code

```
/*  Program: VendingMachine.h
   Author: Jin Choi
   Class: CSCI 140
   Date: 11/12/2018
   Description: Header file for vending machine class.

   I certify that the code below is my own work.

  Exception(s): N/A

*/
#ifndef PROJECT_4_VENDINGMACHINE_H
#define PROJECT_4_VENDINGMACHINE_H

#include <string>
#include <vector>
#include <iomanip>
#include "ChangeMachine.cpp"
#include "Inventory.cpp"

using namespace std;

struct initInfo{
   int dollars, quarters, dimes, nickels;
};

struct availProd{
   int initialQuantity;
   int ID = 0;
   string selection;
   int quantity;
   int price;
   string name;
};

struct transaction{
   int trans, cost, paid, pDollar, pQuarter, pDime, pNickel, change, quarter, dime, nickel;
   string select;
};

struct transactionCC{
   int trans;
   string select;
   int cost, paid, lastFour;
};

class VendingMachine {
protected:
```

```
  int transNum = 1;
  initInfo init = {0,0,0,0};
  string model;
  ChangeMachine changeMachine;
  int numProduct;
  vector<availProd*> products;
  vector<transaction*> trans;
public:
  VendingMachine(string model, int numProducts);
  VendingMachine(string model, int numProducts, int Q, int D, int N);
  ~VendingMachine();
  virtual void stockMachine(Inventory&, string, int, int);
  virtual void startUp() = 0;
  virtual void makeSelection(string select);
  virtual void generateReport();
  virtual string modelNumber(){ return model; };
  void printItemList();
  bool validate(string cardNumber);
};


#endif //PROJECT_4_VENDINGMACHINE_H

/*  Program: VendingMachine.cpp
  Author: Jin Choi
  Class: CSCI 140
  Date: 12/03/2018
  Description: The main abstract class for the Vending Machine. Contains implementation to initialize various shared features for each different model. Implements pure virtual functions that are required to be overrided in derived classes.

  I certify that the code below is my own work.

 Exception(s): N/A

*/

#include "VendingMachine.h"

VendingMachine::VendingMachine(string model, int numProducts) {
  this->numProduct = numProducts;
  this->model = model;
}


VendingMachine::VendingMachine(string model, int numProducts, int Q, int D, int N) {
  this->numProduct = numProducts;
  this->model = model;
  changeMachine.insertChange(Q,D,N);
  init.quarters = Q;
```

```
  init.dimes = D;
  init.nickels = N;
}

VendingMachine::~VendingMachine() {
  for (auto &p : products){
    delete p;
    p = nullptr;
  }
}

void VendingMachine::stockMachine(Inventory &inven, string selection, int ID, int quantity) {
  string name = "";
  int price = 0;
  int availQuantity = inven.removeProduct(ID, quantity);
  name = inven.nameLookUp(ID);
  price = inven.priceLookUp(ID);
  availProd *temp = new availProd;
  temp->ID = ID;
  temp->initialQuantity = quantity;
  temp->selection = selection;
  temp->quantity = quantity;
  temp->price = price;
  temp->name = name;
  products.push_back(temp);
}

void VendingMachine::printItemList() {
  cout << "Available items:" << endl;
  for (int i = 0; i < products.size(); i++) {
    cout << right << setw(6) << products[i]->selection
      << setw(4) << products[i]->price << left << products[i]->name << endl;
  }
}

void VendingMachine::makeSelection(string select) {
  bool valid = false;
  availProd *pCurr;
  for (auto &p : products){
    if (toupper(p->selection.c_str()[1]) == toupper(select.c_str()[1])){
      pCurr = p;
      valid = true;
      cout << "You selected \"" << p->name << "\"." << endl;
      cout << fixed << setprecision(2) << "The cost of this item is " << p->price << " cents." << endl;
      break;
    }
  }
  if (valid){
    int insertedAmount = 0;
    cout << "Insert your money -->";
```

```
int curr = 0;
int inserted[4] = {0};
while(cin >> curr){
    if (curr == 0) { break; }
    if (curr == 100) {
        inserted[0]++;
        insertedAmount += 100;
    }
    else if(curr == 25){
        inserted[1]++;
        insertedAmount += 25;
    }else if(curr == 10){
        inserted[2]++;
        insertedAmount += 10;
    }
    else if (curr == 5){
        inserted[3]++;
        insertedAmount += 5;
    }
}
if (insertedAmount == 0){
    cout << "You chose to cancel your transaction." << endl;
}
else if (insertedAmount < pCurr->price){
    cout << "Not enough money inserted." << endl;
}
else if ((insertedAmount - pCurr->price) > changeMachine.totalBal()){
    cout << "Insufficient change." << endl;
}
else{
    int costOfItem = pCurr->price;
    int initQ, initD, initN;
    int change[3] = {0};
    if (pCurr-> quantity == 0){
        cout << "Product is out of stock." << endl;
    }else{
        if (changeMachine.giveChange(change, costOfItem, inserted[0], inserted[1],inserted[2],inserted[3])){
            pCurr->quantity -= 1;
            transaction *tPtr = new transaction;
            tPtr->trans = transNum;
            transNum++;
            tPtr->select = select;
            tPtr->cost = costOfItem;
            tPtr->paid = insertedAmount;
            tPtr->pDollar = inserted[0];
            tPtr->pQuarter = inserted[1];
            tPtr->pDime = inserted[2];
            tPtr->pNickel = inserted[3];
            tPtr->change = insertedAmount - costOfItem;
            tPtr->quarter = change[0];
```

```
                tPtr->dime = change[1];
                tPtr-> nickel = change[2];
                trans.push_back(tPtr);
                cout << "Your change of " << fixed << setprecision(2) <<insertedAmount - costOfItem << " is given
as:" << endl;
                cout << '\t' << left << setw(12) << "quarter(s): " << tPtr->quarter << endl;
                cout << '\t' << left << setw(12) << "dimes(s): " << tPtr->dime << endl;
                cout << '\t' << left << setw(12) << "nickels(s): " << tPtr->nickel << endl;
                cout << "Thank you! Please take your item." << endl << endl;
            }
            else{
                cout << "Insufficient Change." << endl;
                cout << "Your transaction cannot be processed." << endl;
                cout << "Please take back your money." << endl << endl;
            }
        }
    }
  }
  else{
    cout << "Invalid Selection." << endl;
  }
}

bool VendingMachine::validate(string cardNumber) {
  if (cardNumber.length() < 13 || cardNumber.length() > 16){
    return false;
  }else{
    int totalEven = 0;
    int totalOdd = 0;
    int totalSum = 0;
    for (int i = 1; i < cardNumber.length(); i+=2){
      int currOdd = stoi(cardNumber.substr(cardNumber.length()-i,1));
      int curr = stoi(cardNumber.substr(cardNumber.length()-1-i,1));
      curr *= 2;
      if (curr > 9){
        string temp = to_string(curr);
        curr = stoi(temp.substr(0,1)) + stoi(temp.substr(1,1));
      }
      totalEven += curr;
      totalOdd += currOdd;
    }
    totalSum = totalEven + totalOdd;
    return totalSum%10 == 0;
  }
}

void VendingMachine::generateReport() {
  cout << "Current Balance: $" << static_cast<double>(changeMachine.totalBal())/100
  << " (" << changeMachine.getDollars() << " $, " << changeMachine.getQuarters() << " Q "
  << changeMachine.getDimes() << " D " << changeMachine.getNickels() << " N)" << endl << endl;
```

```
  cout << "Code    ID    Description        Initial    Current" << endl;
  for (auto &p : products){
     cout << setw(3) << p->selection << setw(9) << p->ID << "   " << setw(20) << left << p->name
     << setw(4) << right << p->initialQuantity << setw(10) << p->quantity << endl;
  }
  cout << endl;
}


/*  Program: VendingMachineDriver.cpp
   Author: Jin Choi
   Class: CSCI 140
   Date: 12/03/2018
   Description: Main driver class that reads from the input files and runs the program.

   I certify that the code below is my own work.

  Exception(s): N/A

*/

#include <string>
#include <iostream>
#include <fstream>
#include <vector>
#include "VendingMachine.h"
#include "VendA.cpp"
#include "VendB.cpp"
#include "VendC.cpp"
#include "VendD.cpp"

using namespace std;

static int NUM_MODEL = 4;


int main() {
  fstream machines;
  fstream products;
  fstream report;
  string code;
  cout << "Please enter a startup code --> ";
  cin >> code;
  if (code == "csci140"){
     cout << "Initializing machines. Pleast wait ... " << endl;
     machines.open("../machines.txt");
     products.open("../products.txt");
     vector<VendingMachine*> list[NUM_MODEL];
     Inventory stockList(products);
     VendingMachine *vPtr = nullptr;
```

```
    string model = "";
    int numberMachines = 0;
    int Q, D, N, num, ID, quantity;
    string selection;

    while (!machines.eof()){
        char c = machines.peek();
        machines >> model;
        machines >> numberMachines;
        for (int i = 0; i < numberMachines; i++){
            machines >> Q >> D >> N >> num;
            char option = model.substr(3,1).c_str()[0];
            switch(option){
                case 'A': {vPtr = new VendA(model,num,Q,D,N);list[0].push_back(dynamic_cast<VendA*>(vPtr)); }
break;
                case 'B': {vPtr = new VendB(model,num,Q,D,N); list[1].push_back(dynamic_cast<VendB*>(vPtr));}
break;
                case 'C': {vPtr = new VendC(model,num); list[2].push_back(dynamic_cast<VendC*>(vPtr));} break;
                case 'D': {vPtr = new VendD(model,num,Q,D,N); list[3].push_back(dynamic_cast<VendD*>(vPtr));}
break;
                default: cout << "Invalid Input."; break;
            }
            for (int j = 0; j < num; j++){
                machines >> selection >> ID >> quantity;
                vPtr->stockMachine(stockList, selection, ID, quantity);
            }
        }
        machines.ignore(2);
        vPtr = nullptr;
    }

    delete vPtr;
    cout << "Machines are ready.\nAvailable machines: ";
    string s;
    for (int i = 0; i < NUM_MODEL; i++){
        if (list[i].size() > 0){
            for (auto &v : list[i]){
                s += v->modelNumber();
                s += ", ";
            }
        }
    }
    s.erase(s.size()-2,s.size()-1);
    cout << s << endl << endl;
    while(true){
        string choice;
        char option;
        cout << "Select a machine --> ";
        cin >> choice;
        if (choice == "csci140"){
```

```
            for (int i = 0; i < NUM_MODEL; i++){
                for (int j = 0; j < list[i].size(); j++){
                    VendingMachine *vPtr  = list[i][j];
                    vPtr->generateReport();
                }
            }
            break;
        }
        option = toupper(choice.c_str()[3]);
        int machNum = choice.c_str()[4]-48;
        if (option < 'A' || option > 'D'){
            cout << "Model does not exist." << endl;
        }else if (machNum > list[option-65].size()) {
            cout << "Instance of Machine does not exist." << endl;
        }else{
            VendingMachine *vCurr = list[option-65][machNum-1];
            vCurr->startUp();
            vCurr->printItemList();

            string selection;
            cout << "Select an item --> ";
            cin >> selection;
            vCurr->makeSelection(selection);

        }
        cin.clear();
        cin.ignore(INTMAX_MAX,'\n');
    }


  }
  else{
     cout << "Code Incorrect, Program Terminating.";
  }
  return 0;
}

/*  Program: Inventory.cpp
   Author: Jin Choi
   Class: CSCI 140
   Date: 11/21/2018
   Description: The class that handles the main stock inventory where products are pulled from.

   I certify that the code below is my own work.

  Exception(s): N/A

*/

#include <iostream>
```

```cpp
#include <string>
#include <vector>
#include <iomanip>
#include <fstream>

using namespace std;



class Inventory{
protected:
  struct product{
     int ID;
     int quantity;
     int price;
     string name;
  };
  int size;
  vector<product*> list;
  void addProduct(int ID, int quantity, int price, string name){
     product *prod = new product;
     prod->ID = ID;
     prod->price = price;
     prod->quantity = quantity;
     prod->name = name;
     list.push_back(prod);
  }

public:
  Inventory(fstream &in){
     while (!in.eof()){
        char c = in.peek();
        if (c == -1){
           break;
        }
        int ID, quantity, price;
        string name;
        in >> ID >> quantity >> price;
        getline(in, name, '\r');
        addProduct(ID,quantity,price,name);
        in.ignore(1);
     }
     size = list.size();
  }

  ~Inventory(){
     for (auto &p :list){
        delete p;
        p = nullptr;
     }
```

```
    }


    int removeProduct(int ID, int quantity){
        product *product = nullptr;
        for (auto &a : list){
            if (a->ID == ID){
                product = a;
            }
        }
        if (product != nullptr){
            if (quantity == 0){
                return 0;
            }
            if (quantity > product->quantity){
                int temp = product->quantity;
                product->quantity = 0;
                return temp;
            }
            else{
                product->quantity -= quantity;
                return quantity;
            }
        }
    }
    void print(){
        cout << "ID   Quantity  Price  Name" << endl;
        for (auto &p : list){
            cout << left << setw(5) << p->ID << setw(10) << p->quantity << setw(7) << p->price << p->name << endl;
        }
    }

    string nameLookUp(int ID){
        for (auto &p : list){
            if (p->ID == ID){
                return p->name;
            }
        }
        return "";
    }

    int priceLookUp(int ID){
        for (auto &p : list){
            if (p->ID == ID){
                return p->price;
            }
        }
        return 0;
    }
};
```

```
/*  Program: ChangeMachine.cpp
   Author: Jin Choi
   Class: CSCI 140
   Date: 11/12/2018
   Description: This class is instantiated with each instance of VendingMachine, handles change.

   I certify that the code below is my own work.

  Exception(s): N/A

*/

#include <iostream>

/*const double QUARTER = 0.25;
const double DIME = 0.10;
const double NICKEL = 0.05;*/

class ChangeMachine {
private:
   int Dollars;
   int Quarters;
   int Dimes;
   int Nickels;
   void addDollars(int n){Dollars += n; }
   void addQuarters(int n){ Quarters += n; }
   void addDimes(int n){ Dimes += n; }
   void addNickels(int n) { Nickels += n; }
   int removeQuarters(int n){
      int result = Quarters;
      if (Quarters >= n){
         Quarters -= n;
         result = n;
      }
      else{
         Quarters = 0;
      }
      return result;
   }
   int removeDimes(int n){
      int result = Dimes;
      if (Dimes >= n){
         Dimes -= n;
         result = n;
      }
      else{
         Dimes = 0;
      }
      return result;
```

```
  }
  int removeNickels(int n){
     int result = Nickels;
     if (Nickels >= n){
        Nickels -= n;
        result = n;
     }
     else{
        Nickels = 0;
     }
     return result;
  }
  int removeDollars(int n){
     int result = Dollars;
     if (Dollars >= n){
        Dollars -= n;
        result = n;
     }
     else{
        Dollars = 0;
     }
     return result;
  }

public:
  ChangeMachine(){
     Dollars = 0;
     Quarters = 0;
     Dimes = 0;
     Nickels = 0;
  }
  void insertChange(int Q, int D, int N){
     Quarters = Q;
     Dimes = D;
     Nickels = N;
  }
  int totalBal(){return 100*Dollars + 25*Quarters + 10*Dimes + 5*Nickels;}
  int getDollars() const{return Dollars; }
  int getQuarters() const{ return Quarters; }
  int getDimes() const{ return Dimes; }
  int getNickels() const {return Nickels; }
  bool giveChange(int changeRemoved[], int purchaseAmount, int Dollar, int Q = 0, int D = 0, int N = 0){
     bool success = false;
     int tempDollar = Dollars + Dollar;
     int tempQ = Quarters + Q;
     int tempD = Dimes + D;
     int tempN = Nickels + N;
     int total = tempDollar *100 + tempQ*25 + tempD*10 + tempN*5;
     int inserted = Dollar * 100 + Q*25 + D*10 + N*5;
     int change = inserted - purchaseAmount;
```

```
if (total >= change){
   int removeTempDollar = 0; change/100 < tempDollar ? change/100 : tempDollar;
   if (change > 100 && removeTempDollar>0){
      change -= removeTempDollar*100;
   }
   if (tempN == 0 && tempQ > 0 && tempD > 0 && change > 25) { // Exception Handling Q D
      int smaller = change/25 < tempQ ? change/25 : tempQ;
      if (change%2 == 0){ //change is even
         if (smaller%2 == 1){
            smaller -= 1;
         }
      }
      else{ // change is odd
          if (smaller%2 == 0){
             smaller -= 1;
          }
      }
      if ((change - smaller*25)/10 <= tempD){ // you got enough dimes
         addDollars(Dollar);
         addQuarters(Q);
         addDimes(D);
         addNickels(N);
         removeDollars(removeTempDollar);
         changeRemoved[0] = removeQuarters(smaller);
         changeRemoved[1] = removeDimes((change - smaller*25)/10);
         success = true;
      }
   }
   else{ // GREEDY METHOD
      int rmvQ, rmvD, rmvN;
      rmvQ = rmvD = rmvN = 0;
      if ((change/25) > 0 && tempQ > 0){
         rmvQ = change/25 < tempQ ? change/25 : tempQ;
         change -= rmvQ * 25;
      }
      if ((change/10) > 0 && tempD > 0){
         rmvD = change/10 < tempD ? change/10 : tempD;
         change -= rmvD * 10;
      }
      if ((change/5) > 0 && tempN > 0){
         rmvN = change/5 < tempN ? change/5 : tempN;
         change -= rmvN * 5;
      }
      if (change == 0){
         addDollars(Dollar);
         addQuarters(Q);
         addDimes(D);
         addNickels(N);
         removeDollars(removeTempDollar);
         changeRemoved[0] = removeQuarters(rmvQ);
```

```
            changeRemoved[1] = removeDimes(rmvD);
            changeRemoved[2] = removeNickels(rmvN);
            success = true;
          }
        }
      }
      return success;
  }
};

/*  Program: VendA.cpp
   Author: Jin Choi
   Class: CSCI 140
   Date: 11/12/2018
   Description: Derived class of VendingMachine, accepts only dollar bills as method of payment.

   I certify that the code below is my own work.

  Exception(s): N/A

*/

#include "VendingMachine.h"

static int currNoA = 1;

class VendA : public VendingMachine{
protected:
  int machineNumber;
public:

  VendA(string model, int numProducts, int Q, int D, int N) : VendingMachine(model, numProducts,Q,D,N){
     machineNumber = currNoA;
     currNoA++;
  }

  void generateReport() override {
     if (!trans.empty()){
        cout << "Machine: " << modelNumber() << endl;
        int total = 25*init.quarters + 10*init.dimes + 5*init.nickels;
        int totalCost = 0;
        cout << "Initial Balance: $" << fixed << setprecision(2) <<static_cast<double>(total)/100 <<
            " (" << init.dollars << " $, "
            << init.quarters << " Q, " << init.dimes << " D, "
            << init.nickels << " N)" << endl << endl;
        cout << "Trans Item     Cost     Paid ($, Q, D, N) Changes(Q, D, N)" << endl;
        for (auto &t : trans){
           totalCost += t->cost;
           cout << right << setw(3) << t->trans << setw(7) << t->select << setw(8)
               << t->cost << setw(9) << t->paid << "  (" << t->pDollar << setw(3)
```

```
                << t->pQuarter << setw(3) << t->pDime << setw(3) << t->pNickel << ")"
                << setw(7) << t->change << "  (" << t->quarter << setw(3) << t->dime
                << setw(3) << t->nickel << ")" <<  endl;
        }
        cout << "Total Cost: " << totalCost << endl << endl;
        VendingMachine::generateReport();
    }

  }

  string modelNumber() override {
    string result = VendingMachine::modelNumber();
    result += to_string(machineNumber);
    return result;
  }

  void startUp() override {
    cout << "This machine accepts one-dollar bills only." << endl;
  }
};

/*  Program: VendB.cpp
  Author: Jin Choi
  Class: CSCI 140
  Date: 11/13/2018
  Description: Derived class of VendingMachine, able to accept coins and dollar bills as method of payment.

  I certify that the code below is my own work.

 Exception(s): N/A

*/
#include "VendingMachine.h"

static int currNoB = 1;

class VendB : public VendingMachine {

protected:
  int machineNumber;
public:
  VendB(string model, int numProducts, int Q, int D, int N) : VendingMachine(model,numProducts,Q,D,N){
    machineNumber = currNoB;
    currNoB++;
  }

  void generateReport() override {
    if (!trans.empty()){
      cout << "Machine: " << modelNumber() << endl;
      int total = 25*init.quarters + 10*init.dimes + 5*init.nickels;
```

```
        int totalCost = 0;
        cout << "Initial Balance: $" <<fixed << setprecision(2) << static_cast<double>(total)/100 <<
            " (" << init.dollars << " $, "
            << init.quarters << " Q, " << init.dimes << " D, "
            << init.nickels << " N)" << endl << endl;
        cout << "Trans Item     Cost     Paid ($, Q, D, N) Changes(Q, D, N)" << endl;
        for (auto &t : trans){
           totalCost += t->cost;
           cout << right << setw(3) << t->trans << setw(7) << t->select << setw(8)
               << t->cost << setw(9) << t->paid << "  (" << t->pDollar << setw(3)
               << t->pQuarter << setw(3) << t->pDime << setw(3) << t->pNickel << ")"
               << setw(7) << t->change << "  (" << t->quarter << setw(3) << t->dime << setw(3) << t->nickel << ")" <<
endl;
        }
        cout << "Total Cost: " << totalCost << endl << endl;
        VendingMachine::generateReport();
     }
  }

  string modelNumber() override {
     string result = VendingMachine::modelNumber();
     result += to_string(machineNumber);
     return result;
  }

  void startUp() override {
     cout << "This machine accepts one-dollar bills and coins." << endl;
  }
};

/*  Program: VendC.cpp
   Author: Jin Choi
   Class: CSCI 140
   Date: 11/13/2018
   Description: Derived class of VendingMachine, able to accept credit card as payment.

   I certify that the code below is my own work.

  Exception(s): N/A

*/

#include <string>
#include "VendingMachine.h"

static int currNoC = 1;

class VendC : public VendingMachine{
protected:
   vector<transactionCC*> transCC;
```

```
  int machineNumber;
public:

  VendC(string model, int numProducts) : VendingMachine(model, numProducts){
     machineNumber = currNoC;
     currNoC++;
  }

  ~VendC(){
     for (auto &t : transCC){
        delete t;
        t = nullptr;
     }
  }
  void generateReport() override {
     if (!transCC.empty()){
        cout << "Machine: " << modelNumber() << endl;
        int total = 25*init.quarters + 10*init.dimes + 5*init.nickels;
        int totalCost = 0;
        cout << "Initial Balance: $" << fixed << setprecision(2) << static_cast<double>(total)/100 <<
        " (" << init.dollars << " $, "
        << init.quarters << " Q, " << init.dimes << " D, "
        << init.nickels << " N)" << endl << endl;
        cout << "Trans Item     Cost     Paid Last 4 digits of credit card" << endl;
        for (auto &t : transCC){
           totalCost += t->cost;
           cout << right << setw(3) << t->trans << setw(7) << t->select << setw(8)
           << t->cost << setw(9) << t->paid << setw(6) << t->lastFour << endl;
        }
        cout << "Total Cost: " << totalCost << endl << endl;
        VendingMachine::generateReport();
     }
  }

  string modelNumber() override {
     string result = VendingMachine::modelNumber();
     result += to_string(machineNumber);
     return result;
  }

  void startUp() override {
     cout << "This machine accepts credit card only." << endl;
  }

  void makeSelection(string select) override {
     bool valid = false;
     availProd *pCurr;
     for (auto &p : products){
        if (toupper(p->selection.c_str()[1]) == toupper(select.c_str()[1])){
           pCurr = p;
```

```
            valid = true;
            cout << "You selected \"" << p->name << "\"." << endl;
            cout << "The cost of this item is " << p->price << " cents." << endl;
            break;
         }
      }
      if (valid){
         string cardNumber;
         int failure = 0;
         while (failure < 2){
            cout << "Enter your credit card number -->";
            cin >> cardNumber;
            if (validate(cardNumber)){
               break;
            }
            else{
               cout << "Invalid credit card number was entered." << endl;
               failure++;
            }
         }
         if (failure == 2){
            cout << "Too many invalid card numbers." << endl;
         }else{
            //purchase
            if (pCurr-> quantity == 0){
               cout << "Product is out of stock." << endl;
            }else{
               pCurr->quantity -= 1;
               transactionCC *tPtr = new transactionCC;
               tPtr->trans = transNum;
               transNum++;
               tPtr->select = select;
               tPtr->cost = pCurr->price;
               tPtr->paid = pCurr->price;
               tPtr->lastFour = stoi(cardNumber.substr(12,4));
               transCC.push_back(tPtr);
               cout << "Your credit card was successfully charged for $" << fixed << setprecision(2) <<
static_cast<double>(pCurr->price)/100 << "." << endl;
               cout << "Thank you! Please take your item." << endl;
            }
         }
      }
      else{
         cout << "Invalid Selection." << endl;
      }
   }
};

/*  Program: VendD.cpp
   Author: Jin Choi
```

```
  Class: CSCI 140
  Date: 11/13/2018
  Description: Derived class of VendingMachine class that is able to take both credit card and dollar bills as a method
of payment.

  I certify that the code below is my own work.

 Exception(s): N/A

*/

#include <string>
#include "VendingMachine.h"

static int currNoD = 1;

class VendD : public VendingMachine{
protected:
  vector<transactionCC*> transCC;
  int machineNumber;
public:
  VendD(string model, int numProducts, int Q, int D, int N) : VendingMachine(model,numProducts,Q,D,N){
     machineNumber = currNoD;
     currNoD++;
  }
  ~VendD(){
     for (auto &t : transCC){
        delete t;
        t = nullptr;
     }
  }

  void generateReport() override {
     cout << "Machine: " << modelNumber() << endl;
     int total = 25*init.quarters + 10* init.dimes + 5*init.nickels;
     int totalCost = 0;
     cout << "Initial Balance: $" << fixed << setprecision(2) <<static_cast<double>(total)/100 <<
        " (" << init.dollars << " $, "
        << init.quarters << " Q, " << init.dimes << " D, "
        << init.nickels << " N)" << endl << endl;
     if (!trans.empty()){
        cout << "Trans Item     Cost     Paid ($, Q, D, N) Changes(Q, D, N)" << endl;
        for (auto &t : trans){
           totalCost += t->cost;
           cout << right << setw(3) << t->trans << setw(7) << t->select << setw(8)
              << t->cost << setw(9) << t->paid << "  (" << t->pDollar << setw(3)
              << t->pQuarter << setw(3) << t->pDime << setw(3) << t->pNickel << ")"
              << setw(7) << t->change << "  (" << t->quarter << setw(3) << t->dime << setw(3) << t->nickel << ")"<<
endl;
        }
```

```
    }
    if (!transCC.empty()){

        cout << "Trans Item     Cost     Paid Last 4 digits of credit card" << endl;
        for (auto &t : transCC){
            totalCost += t->cost;
            cout << right << setw(3) << t->trans << setw(7) << t->select << setw(8)
                 << t->cost << setw(9) << t->paid << setw(6) << t->lastFour << endl;
        }
    }
    if (!transCC.empty() || !trans.empty()){
        cout << "Total Cost: " << totalCost << endl << endl;
        VendingMachine::generateReport();
    }

}

string modelNumber() override {
    string result = VendingMachine::modelNumber();
    result += to_string(machineNumber);
    return result;
}

void startUp() override {
    cout << "This machine accepts both dollar and credit card." << endl;
}

void makeSelection(string select) override {
    bool valid = false;
    availProd *pCurr;
    for (auto &p : products){
        if (toupper(p->selection.c_str()[1]) == toupper(select.c_str()[1])){
            pCurr = p;
            valid = true;
            cout << "You selected \"" << p->name << "\"." << endl;
            cout << "The cost of this item is " << p->price << " cents." << endl;
            break;
        }
    }
    if (valid){
        int DollarOrCC;
        while(true){
            cout << "Purchase options:\n\t1. Credit Card\n\t2. Dollar Bill\n\t'Q' to Quit-->";
            cin>>DollarOrCC;
            if (cin.fail()){
                cout << "Invalid option selected." << endl;
            }
            if (DollarOrCC == 1){
                cout << "Credit Card option selected." << endl;
                break;
```

```
            }
            else if (DollarOrCC == 2){
                cout << "Dollar Bill option selected." << endl;
                break;
            }
            else if (DollarOrCC == 'Q'){
                cout << "Cancelling transaction." << endl << endl;
                break;
            }
        }
        if (DollarOrCC == 1){
            string cardNumber;
            int failure = 0;
            while (failure < 2){
                cout << "Enter your credit card number -->";
                cin >> cardNumber;
                if (validate(cardNumber)){
                    break;
                }
                else{
                    cout << "Invalid credit card number was entered." << endl;
                    failure++;
                }
            }
            if (failure == 2){
                cout << "Too many invalid card numbers." << endl;
            }else{
                //purchase
                if (pCurr-> quantity == 0){
                    cout << "Product is out of stock." << endl;
                }
                else {
                    pCurr->quantity -= 1;
                    transactionCC *tPtr = new transactionCC;
                    tPtr->trans = transNum;
                    transNum++;
                    tPtr->select = select;
                    tPtr->cost = pCurr->price;
                    tPtr->paid = pCurr->price;
                    tPtr->lastFour = stoi(cardNumber.substr(12,4));
                    transCC.push_back(tPtr);
                    cout << "Your credit card was successfully charged for $" <<fixed << setprecision(2) <<
static_cast<double>(pCurr->price)/100 << "." << endl;
                    cout << "Thank you! Please take your item." << endl;
                }
            }
        }
        else if(DollarOrCC == 2) {
            int insertedAmount = 0;
            cout << "Insert your money -->";
```

```
int curr = 0;
int inserted[4] = {0};
while (cin >> curr) {
    if (curr == 0) { break; }
    if (curr == 100) {
        inserted[0]++;
        insertedAmount += 100;
    } else if (curr == 25) {
        inserted[1]++;
        insertedAmount += 25;
    } else if (curr == 10) {
        inserted[2]++;
        insertedAmount += 10;
    } else if (curr == 5) {
        inserted[3]++;
        insertedAmount += 5;
    }
}
if (insertedAmount == 0) {
    cout << "You chose to cancel your transaction." << endl;
} else if (insertedAmount < pCurr->price) {
    cout << "Not enough money inserted." << endl;
} else if ((insertedAmount - pCurr->price) > changeMachine.totalBal()) {
    cout << "Insufficient change." << endl;
} else {
    //purchase, give change
    int costOfItem = pCurr->price;
    int numDollars = inserted[0];
    int change[3] = {0};
    if (pCurr-> quantity == 0){
        cout << "Product is out of stock." << endl;
    }else{
        if (changeMachine.giveChange(change, costOfItem, numDollars, inserted[1], inserted[2],inserted[3])) {
            pCurr->quantity -= 1;
            transaction *tPtr = new transaction;
            tPtr->trans = transNum;
            transNum++;
            tPtr->select = select;
            tPtr->cost = costOfItem;
            tPtr->paid = insertedAmount;
            tPtr->pDollar = inserted[0];
            tPtr->pQuarter = inserted[1];
            tPtr->pDime = inserted[2];
            tPtr->pNickel = inserted[3];
            tPtr->change = insertedAmount - costOfItem;
            tPtr->quarter = change[0];
            tPtr->dime = change[1];
            tPtr->nickel = change[2];
            trans.push_back(tPtr);
            cout << "Your change of " << insertedAmount - costOfItem << " is given as:" << endl;
```

```
                cout << '\t' << left << setw(12) << "quarter(s): " << tPtr->quarter << endl;
                cout << '\t' << left << setw(12) << "dimes(s): " << tPtr->dime << endl;
                cout << '\t' << left << setw(12) << "nickels(s): " << tPtr->nickel << endl;
                cout << "Thank you! Please take your item." << endl << endl;
            } else {
                cout << "Insufficient Change." << endl;
                cout << "Your transaction cannot be processed." << endl;
                cout << "Please take back your money." << endl << endl;

            }
        }

    }
  }
  else{
    cout << "Invalid Selection." << endl;
  }
 }
};
```

Output

Test Case 0

```
Please enter a startup code --> csci140
Initializing machines. Pleast wait ...
Machines are ready.
Available machines: 100A1, 100A2, 100B1, 100B2, 100B3, 100C1, 100C2, 100D1

Select a machine --> 100A1
This machine accepts one-dollar bills only.
Available items:
    1A  50 candy bar
    1B  35 chocolate chips
    1C  75 cookies
    1D  60 brownie
    1E 165 protein bar
Select an item --> 1B
You selected " chocolate chips".
The cost of this item is 35 cents.
Insert your money -->100 0
Your change of 65 is given as:
    quarter(s): 2
    dimes(s):   1
    nickels(s): 1
Thank you! Please take your item.

Select a machine --> 100A1
This machine accepts one-dollar bills only.
Available items:
    1A  50 candy bar
    1B  35 chocolate chips
    1C  75 cookies
    1D  60 brownie
    1E 165 protein bar
Select an item --> 1B
You selected " chocolate chips".
The cost of this item is 35 cents.
Insert your money -->100 0
Insufficient Change.
Your transaction cannot be processed.
Please take back your money.

Select a machine --> 100A1
This machine accepts one-dollar bills only.
Available items:
    1A  50 candy bar
    1B  35 chocolate chips
    1C  75 cookies
    1D  60 brownie
    1E 165 protein bar
Select an item --> 1C
You selected " cookies".
The cost of this item is 75 cents.
Insert your money -->100 0
Your change of 25 is given as:
    quarter(s): 0
    dimes(s):   2
    nickels(s): 1
Thank you! Please take your item.
```

```
Select a machine --> 100A1
This machine accepts one-dollar bills only.
Available items:
    1A  50 candy bar
    1B  35 chocolate chips
    1C  75 cookies
    1D  60 brownie
    1E 165 protein bar
Select an item --> 1B
You selected " chocolate chips".
The cost of this item is 35 cents.
Insert your money -->0
You chose to cancel your transaction.
Select a machine --> 100C1
This machine accepts credit card only.
Available items:
    1A 300 ham sandwich
    1B 275 egg sandwich
    1C 325 tuna sandwich
Select an item --> 1B
You selected " egg sandwich".
The cost of this item is 275 cents.
Enter your credit card number -->4388576018402625
Invalid credit card number was entered.
Enter your credit card number -->4388576018410707
Your credit card was successfully charged for $2.75.
Thank you! Please take your item.
Select a machine --> csci140
Machine: 100A1
Initial Balance: $1.10 (0 $, 2 Q, 4 D, 4 N)

Trans Item     Cost     Paid ($, Q, D, N) Changes(Q, D, N)
  1     1B      35      100  (1  0  0  0)     65  (2  1  1)
  2     1C      75      100  (1  0  0  0)     25  (0  2  1)
Total Cost: 110

Current Balance: $2.20 (2 $, 0 Q 1 D 2 N)

Code     ID     Description         Initial    Current
 1A     1034    candy bar             5         5
 1B     1000    chocolate chips      10         9
 1C     1100    cookies               1         0
 1D     1123    brownie              20        20
 1E     1210    protein bar           5         5

Machine: 100C1
Initial Balance: $0.00 (0 $, 0 Q, 0 D, 0 N)

Trans Item     Cost     Paid Last 4 digits of credit card
  1     1B     275      275   707
Total Cost: 275

Current Balance: $0.00 (0 $, 0 Q 0 D 0 N)

Code     ID     Description         Initial    Current
 1A     6774    ham sandwich          5         5
 1B     6869    egg sandwich          5         4
 1C     6879    tuna sandwich         2         2

Machine: 100D1
Initial Balance: $3.15 (0 $, 10 Q, 5 D, 3 N)
```

Test Case 1

```
Please enter a startup code --> csci140
Initializing machines. Pleast wait ...
Machines are ready.
Available machines: 100A1, 100A2, 100B1, 100B2, 100B3, 100C1, 100C2, 100D1

Select a machine --> 100b1
This machine accepts one-dollar bills and coins.
Available items:
    1A  80 coke bottle
    1B  60 coke can
    1C  80 diet coke bottle
    1D  95 12 oz orange juice
    1E  75 8 oz orange juice
    1F  65 apple juice
    1G  60 diet coke can
Select an item --> 1A
You selected " coke bottle".
The cost of this item is 80 cents.
Insert your money -->25 25 10 10 10 13 5 5 5 5 0
Your change of 20 is given as:
    quarter(s): 0
    dimes(s):   2
    nickels(s): 0
Thank you! Please take your item.

Select a machine --> 100b2
This machine accepts one-dollar bills and coins.
Available items:
    1A 125 cappuccino
    1B 125 latte
    1C  80 hot chocolate
    1D  60 decaff coffee
    1E  85 large coffee
    1F  60 regular coffee
Select an item --> 1b
You selected " latte".
The cost of this item is 125 cents.
Insert your money -->25 25 25 10 10 5 5 5 10 25 0
Your change of 20 is given as:
    quarter(s): 0
    dimes(s):   2
    nickels(s): 0
Thank you! Please take your item.

Select a machine --> 100B3
This machine accepts one-dollar bills and coins.
Available items:
    1A  50 candy bar
    1B  35 chocolate chips
    1C  75 cookies
    1D  60 brownie
    1E 165 protein bar
Select an item --> 1E
You selected " protein bar".
The cost of this item is 165 cents.
Insert your money -->100 25 10 5 5 10 25 10 5 0
Your change of 30 is given as:
    quarter(s): 1
    dimes(s):   0
    nickels(s): 1
Thank you! Please take your item.
```

```
Select a machine --> 100D1
This machine accepts both dollar and credit card.
Available items:
    1A  80 coke bottle
    1B  95 12 oz orange juice
    1C  65 apple juice
Select an item --> 1A
You selected " coke bottle".
The cost of this item is 80 cents.
Purchase options:
    1. Credit Card
    2. Dollar Bill
    'Q' to Quit-->1
Credit Card option selected.
Enter your credit card number -->4388576018410707
Your credit card was successfully charged for $0.80.
Thank you! Please take your item.
Select a machine --> 100D1
This machine accepts both dollar and credit card.
Available items:
    1A  80 coke bottle
    1B  95 12 oz orange juice
    1C  65 apple juice
Select an item --> 1B
You selected " 12 oz orange juice".
The cost of this item is 95 cents.
Purchase options:
    1. Credit Card
    2. Dollar Bill
    'Q' to Quit-->2
Dollar Bill option selected.
Insert your money -->100 0
Your change of 5 is given as:
    quarter(s): 0
    dimes(s):   0
    nickels(s): 1
Thank you! Please take your item.

Select a machine --> csci140
Machine: 100B1
Initial Balance: $1.10 (0 $, 2 Q, 4 D, 4 N)

Trans Item     Cost     Paid ($, Q, D, N) Changes(Q, D, N)
  1     1A      80      100  (0  2  3  4)     20  (0  2  0)
Total Cost: 80

Current Balance: $1.90 (0 $, 4 Q 5 D 8 N)

Code     ID     Description         Initial    Current
 1A     2180    coke bottle          10         9
 1B     1283    coke can             10        10
 1C     3629    diet coke bottle      5         5
 1D     3649    12 oz orange juice    3         3
 1E     4051    8 oz orange juice    15        15
 1F     4211    apple juice          10        10
 1G     3026    diet coke can         5         5
```

```
Machine: 100B2
Initial Balance: $2.00 (0 $, 5 Q, 6 D, 3 N)

Trans Item     Cost     Paid ($, Q, D, N) Changes(Q, D, N)
  1     1b     125      145  (0  4  3  3)     20  (0  2  0)
Total Cost: 125

Current Balance: $3.25 (0 $, 9 Q 7 D 6 N)

Code     ID     Description         Initial    Current
 1A     6626    cappuccino            5         5
 1B     6155    latte                 5         4
 1C     5982    hot chocolate        10        10
 1D     5573    decaff coffee         3         3
 1E     5454    large coffee         10        10
 1F     5336    regular coffee       50        50

Machine: 100B3
Initial Balance: $4.00 (0 $, 10 Q, 10 D, 10 N)

Trans Item     Cost     Paid ($, Q, D, N) Changes(Q, D, N)
  1     1E     165      195  (1  2  3  3)     30  (1  0  1)
Total Cost: 165

Current Balance: $5.65 (1 $, 11 Q 13 D 12 N)

Code     ID     Description         Initial    Current
 1A     1034    candy bar             5         5
 1B     1000    chocolate chips       5         5
 1C     1100    cookies               5         5
 1D     1123    brownie               5         5
 1E     1210    protein bar          12        11

Machine: 100D1
Initial Balance: $3.15 (0 $, 10 Q, 5 D, 3 N)

Trans Item     Cost     Paid ($, Q, D, N) Changes(Q, D, N)
  2     1B      95      100  (1  0  0  0)      5  (0  0  1)
Trans Item     Cost     Paid Last 4 digits of credit card
  1     1A      80       80   707
Total Cost: 175

Current Balance: $4.10 (1 $, 10 Q 5 D 2 N)

Code     ID     Description         Initial    Current
 1A     2180    coke bottle           5         4
 1B     3649    12 oz orange juice    2         1
 1C     4211    apple juice           3         3
```