#### **ASSIGNMENT**

#### **NAME: KADAMBALA MEGHANA**

#### **ADMISSION NUMBER:19JE0411**

#### **1)**

### **//code**

### **#include <bits/stdc++.h>**

### **using namespace std;**

### **class Customer**

### **{**

### **private:**

### **string address;**

### **string ph;**

### **string lic\_no;**

### **long c\_n;**

### **long aP;**

### **public:**

### **Customer(string res, string tel, string license, long custno);**

### **bool operator<(Customer cust);**

### **void updateaP(long amt);**

### **void resetaP();**

### **long getaP();**

### **long gcN();**

### **};**

### **Customer::Customer(string res, string tel, string license, long custno)**

### **{**

### **address = res;**

### **ph = tel;**

### **lic\_no = license;**

### **c\_n = custno;**

### **aP = 0;**

### **}**

### **bool Customer::operator<(Customer cust)**

### **{**

### **return aP < cust.aP;**

### **}**

### **void Customer::updateaP(long amt)**

### **{**

### **aP += amt;**

### **cout << "Check-out successful\n";**

### **cout << "Annual purchase amt as of now: " << aP << "\n";**

### **}**

### **void Customer::resetaP()**

### **{**

### **aP = 0;**

### **}**

### **long Customer::getaP()**

### **{**

### **return aP;**

### **}**

### **long Customer::gcN()**

### **{**

### **return c\_n;**

### **}**

### **class sm**

### **{**

### **private:**

### **vector<Customer> cus;**

### **public:**

### **void reg\_cus();**

### **void Check\_out();**

### **void prep\_winners();**

### **};**

### **void sm::reg\_cus()**

### **{**

### **string res, tel, license;**

### **fflush(stdin);**

### **cout << "Enter address:";**

### **getline(cin, res);**

### **cout << "Enter phone number:";**

### **cin >> tel;**

### **cout << "Enter license number:";**

### **cin >> license;**

### **Customer customer(res, tel, license, cus.size() + 1);**

### **cus.push\_back(customer);**

### **cout << "Customer registered successfully.\n";**

### **cout << "c\_n : " << customer.gcN() << "\n\n";**

### **}**

### **void sm::Check\_out()**

### **{**

### **long custno, amt;**

### **cout << "Enter customer number (c\_n):";**

### **cin >> custno;**

### **int n = cus.size(), idx = -1;**

### **for (int i = 0; i < n; i++)**

### **{**

### **if (cus[i].gcN() == custno)**

### **{**

### **idx = i;**

### **break;**

### **}**

### **}**

### **if (idx == -1)**

### **{**

### **cout << "Invalid customer number. Please try again\n\n";**

### **return;**

### **}**

### **cout << "Enter purchase amt:";**

### **cin >> amt;**

### **cus[idx].updateaP(amt);**

### **}**

### **void sm::prep\_winners()**

### **{**

### **sort(cus.rbegin(), cus.rend());**

### **int n = cus.size(), winers = min(10, n);**

### **cout << "Surprise gift winers:\n";**

### **for (int i = 0; i < winers; i++)**

### **{**

### **cout << cus[i].gcN() << "\n";**

### **}**

### **cout << "\nGold Coin winers:\n";**

### **for (int i = 0; i < n; i++)**

### **{**

### **if (cus[i].getaP() > 10000)**

### **{**

### **cout << cus[i].gcN() << "\n";**

### **}**

### **}**

### **cout << "\n";**

### **for (int i = 0; i < n; i++)**

### **cus[i].resetaP();**

### **}**

### **int main()**

### **{**

### **sm sm;**

### **int c;**

### **cout << "Welcome to sm!";**

### **while (true)**

### **{**

### **cout << "Options:\n";**

### **cout << "1. Register as a new customer\n";**

### **cout << "2. Check-out a purchase\n";**

### **cout << "3. Prepare prize winers' list\n";**

### **cout << "4. Exit\n";**

### **cout << "Enter your option:";**

### **cin >> c;**

### **switch (c)**

### **{**

### **case 1:**

### **sm.reg\_cus();**

### **break;**

### **case 2:**

### **sm.Check\_out();**

### **break;**

### **case 3:**

### **sm.prep\_winners();**

### **break;**

### **case 4:**

### **exit(0);**

### **break;**

### **default:**

### **cout << "Invalid\n";**

### **break;**

### **}**

### **}**

### **}**

##### **2)**

##### **#include <bits/stdc++.h>**

##### **using namespace std;**

##### **class cus**

##### **{**

##### **private:**

##### **string ra;**

##### **long c\_n;**

##### **long ords;**

##### **long tP;**

##### **public:**

##### **cus(string res, long cus\_No)**

##### **{**

##### **ra = res;**

##### **c\_n = cus\_No;**

##### **ords = 0;**

##### **tP = 0;**

##### **}**

##### **string gettingAdd()**

##### **{**

##### **return ra;**

##### **}**

##### **long gettingcu\_No()**

##### **{**

##### **return c\_n;**

##### **}**

##### **bool Credit()**

##### **{**

##### **if (ords == 0)**

##### **return true;**

##### **return (tP \* 100) / ords >= 50;**

##### **}**

##### **void incrementOrder()**

##### **{**

##### **ords++;**

##### **}**

##### **void incrementPayment()**

##### **{**

##### **tP++;**

##### **}**

##### **};**

##### **class Commo**

##### **{**

##### **private:**

##### **long CommoId;**

##### **long unitPrice;**

##### **string CommoName;**

##### **long dealno;**

##### **public:**

##### **Commo(long id, long price, string name, long deal)**

##### **{**

##### **CommoId = id;**

##### **unitPrice = price;**

##### **CommoName = name;**

##### **dealno = deal;**

##### **}**

##### **long gettingCommoId()**

##### **{**

##### **return CommoId;**

##### **}**

##### **long gettingUnitPrice()**

##### **{**

##### **return unitPrice;**

##### **}**

##### **string gettingCommoName()**

##### **{**

##### **return CommoName;**

##### **}**

##### **long gettingdealno()**

##### **{**

##### **return dealno;**

##### **}**

##### **};**

##### **class Item**

##### **{**

##### **private:**

##### **long CommoId;**

##### **long quantity;**

##### **public:**

##### **Item(long id, long qty)**

##### **{**

##### **CommoId = id;**

##### **quantity = qty;**

##### **}**

##### **long gettingCommoId()**

##### **{**

##### **return CommoId;**

##### **}**

##### **long gettingQuantity()**

##### **{**

##### **return quantity;**

##### **}**

##### **void setQuantity(long newQty)**

##### **{**

##### **quantity = newQty;**

##### **}**

##### **};**

##### **class Order**

##### **{**

##### **private:**

##### **vector<Item> items;**

##### **long c\_n;**

##### **long orderId;**

##### **time\_t placedAt;**

##### **public:**

##### **Order(long cus\_No)**

##### **{**

##### **c\_n = cus\_No;**

##### **}**

##### **long gettingcu\_No()**

##### **{**

##### **return c\_n;**

##### **}**

##### **long gettingOrderId()**

##### **{**

##### **return orderId;**

##### **}**

##### **void setOrderId(long oId)**

##### **{**

##### **orderId = oId;**

##### **}**

##### **time\_t gettingPlacementTime()**

##### **{**

##### **return placedAt;**

##### **}**

##### **void setPlacementTime()**

##### **{**

##### **placedAt = time(0);**

##### **}**

##### **vector<Item> gettingItems()**

##### **{**

##### **return items;**

##### **}**

##### **bool isItemPresent(long itemId)**

##### **{**

##### **for (int i = 0; i < items.size(); i++)**

##### **{**

##### **if (items[i].gettingCommoId() == itemId)**

##### **{**

##### **return true;**

##### **}**

##### **}**

##### **return false;**

##### **}**

##### **bool isEmpty()**

##### **{**

##### **return items.size() == 0;**

##### **}**

##### **void addItem(Item item)**

##### **{**

##### **items.push\_back(item);**

##### **}**

##### **void updateItem(long itemId, long qty)**

##### **{**

##### **for (int i = 0; i < items.size(); i++)**

##### **{**

##### **if (items[i].gettingCommoId() == itemId)**

##### **{**

##### **items[i].setQuantity(qty);**

##### **return;**

##### **}**

##### **}**

##### **}**

##### **};**

##### **class Dealer**

##### **{**

##### **private:**

##### **string dealerName;**

##### **public:**

##### **Dealer(string name)**

##### **{**

##### **dealerName = name;**

##### **}**

##### **string gettingDealerName()**

##### **{**

##### **return dealerName;**

##### **}**

##### **};**

##### **vector<Dealer> dealers;**

##### **void prepDealerData()**

##### **{**

##### **dealers.push\_back(Dealer("Pro Matchstick Dealers"));**

##### **dealers.push\_back(Dealer("Kiran Edibles"));**

##### **dealers.push\_back(Dealer("Saleem Healthcare"));**

##### **}**

##### **vector<Commo> commoties;**

##### **void prepOverallcommolist()**

##### **{**

##### **prepDealerData();**

##### **commoties.push\_back(Commo(1, 20, "Matchsticks", 0));**

##### **commoties.push\_back(Commo(2, 60, "Sunflower Oil", 1));**

##### **commoties.push\_back(Commo(3, 100, "Detergent", 2));**

##### **commoties.push\_back(Commo(4, 70, "Bathing Soap", 2));**

##### **commoties.push\_back(Commo(5, 35, "Brown Bread", 1));**

##### **}**

##### **Commo searchCommo(long CommoId)**

##### **{**

##### **Commo Commo(-1, -1, "ABCD", 0);**

##### **for (int i = 0; i < commoties.size(); i++)**

##### **{**

##### **if (commoties[i].gettingCommoId() == CommoId)**

##### **return commoties[i];**

##### **}**

##### **return Commo;**

##### **}**

##### **class tradeHouse**

##### **{**

##### **private:**

##### **const double TIME\_THRESHOLD = 120;**

##### **vector<cus> cuss;**

##### **vector<Order> ords;**

##### **vector<bool> paymentCompleted;**

##### **vector<Item> inventory;**

##### **vector<Order> pendingords;**

##### **vector<Item> records;**

##### **public:**

##### **tradeHouse();**

##### **void prepInventory();**

##### **void registercus();**

##### **void prepOrderList();**

##### **void processOrder(Order order, int cusIdx);**

##### **void handleIndentreq();**

##### **void manaQuery();**

##### **void issuePayment();**

##### **bool isCommoDealt(long CommoId)**

##### **{**

##### **return gettingInventoryIndex(CommoId) != -1;**

##### **}**

##### **long gettingInventoryIndex(long CommoId)**

##### **{**

##### **for (int i = 0; i < inventory.size(); i++)**

##### **{**

##### **if (inventory[i].gettingCommoId() == CommoId)**

##### **return i;**

##### **}**

##### **return -1;**

##### **}**

##### **long gettingcusIndex(long cus\_No)**

##### **{**

##### **for (int i = 0; i < cuss.size(); i++)**

##### **{**

##### **if (cuss[i].gettingcu\_No() == cus\_No)**

##### **{**

##### **return i;**

##### **}**

##### **}**

##### **return -1;**

##### **}**

##### **long gettingRecordsIndex(long CommoId)**

##### **{**

##### **for (int i = 0; i < records.size(); i++)**

##### **{**

##### **if (records[i].gettingCommoId() == CommoId)**

##### **{**

##### **return i;**

##### **}**

##### **}**

##### **return -1;**

##### **}**

##### **long gettingOrderIndex(long orderId)**

##### **{**

##### **for (int i = 0; i < ords.size(); i++)**

##### **{**

##### **if (ords[i].gettingOrderId() == orderId)**

##### **{**

##### **return i;**

##### **}**

##### **}**

##### **return -1;**

##### **}**

##### **void clearRecords()**

##### **{**

##### **records.clear();**

##### **records = vector<Item>(inventory);**

##### **for (int i = 0; i < records.size(); i++)**

##### **records[i].setQuantity(0);**

##### **}**

##### **};**

##### **tradeHouse::tradeHouse()**

##### **{**

##### **prepInventory();**

##### **}**

##### **void tradeHouse::prepInventory()**

##### **{**

##### **inventory.push\_back(Item(2, 5));**

##### **inventory.push\_back(Item(4, 5));**

##### **inventory.push\_back(Item(5, 5));**

##### **}**

##### **void tradeHouse::registercus()**

##### **{**

##### **int newCustId = cuss.size() + 1;**

##### **string res;**

##### **fflush(stdin);**

##### **cout << "\nEnter cus address:";**

##### **getline(cin, res);**

##### **cus cus(res, newCustId);**

##### **cuss.push\_back(cus);**

##### **cout << "cus registered successfully.\n";**

##### **cout << "cus Identification Number : " << newCustId << "\n\n";**

##### **}**

##### **void tradeHouse::prepOrderList()**

##### **{**

##### **long cus\_No, itemId, qty, idx;**

##### **cout << "\nEnter cus number:";**

##### **cin >> cus\_No;**

##### **idx = gettingcusIndex(cus\_No);**

##### **if (idx == -1)**

##### **{**

##### **cout << "Invalid cus identification number. Please try again.\n\n";**

##### **return;**

##### **}**

##### **if (cuss[idx].Credit() == false)**

##### **{**

##### **cout << "Sorry, our organization rules do not allow accepting ords from you.\n\n";**

##### **return;**

##### **}**

##### **Order order(cus\_No);**

##### **while (true)**

##### **{**

##### **cout << "\n0.\tCheck-out\n";**

##### **for (int i = 0; i < commoties.size(); i++)**

##### **{**

##### **cout << commoties[i].gettingCommoId() << "\t" << commoties[i].gettingCommoName() << "\n";**

##### **}**

##### **cout << "\nEnter your choice:";**

##### **cin >> itemId;**

##### **if (itemId == 0)**

##### **{**

##### **if (order.isEmpty())**

##### **{**

##### **cout << "Enter at least one item.\n";**

##### **continue;**

##### **}**

##### **else**

##### **{**

##### **break;**

##### **}**

##### **}**

##### **if (order.isItemPresent(itemId))**

##### **{**

##### **cout << "Enter updated quantity:";**

##### **cin >> qty;**

##### **order.updateItem(itemId, qty);**

##### **cout << "Item updated.\n";**

##### **}**

##### **else**

##### **{**

##### **cout << "Enter quantity:";**

##### **cin >> qty;**

##### **order.addItem(Item(itemId, qty));**

##### **cout << "Item added.\n";**

##### **}**

##### **}**

##### **processOrder(order, idx);**

##### **}**

##### **void tradeHouse::processOrder(Order order, int cusIdx)**

##### **{**

##### **vector<long> regret;**

##### **vector<Item> orderItems = order.gettingItems();**

##### **Order finalOrder(order.gettingcu\_No());**

##### **Order pendingOrder(order.gettingcu\_No());**

##### **for (int i = 0; i < orderItems.size(); i++)**

##### **{**

##### **if (!isCommoDealt(orderItems[i].gettingCommoId()))**

##### **{**

##### **regret.push\_back(orderItems[i].gettingCommoId());**

##### **}**

##### **}**

##### **if (!regret.empty())**

##### **{**

##### **cout << "\nUnfortunately, we do not deal in the following items.\n";**

##### **for (int i = 0; i < regret.size(); i++)**

##### **{**

##### **cout << searchCommo(regret[i]).gettingCommoName() << "\n";**

##### **}**

##### **}**

##### **cout << "\n";**

##### **if (regret.size() == orderItems.size())**

##### **{**

##### **return;**

##### **}**

##### **//Finalize the quantities and update the stocks.**

##### **for (int i = 0; i < orderItems.size(); i++)**

##### **{**

##### **long CommoId = orderItems[i].gettingCommoId();**

##### **if (!isCommoDealt(CommoId))**

##### **continue;**

##### **//Find index of stock**

##### **long stockIdx = gettingInventoryIndex(CommoId);**

##### **long stockQty, orderQty;**

##### **orderQty = orderItems[i].gettingQuantity();**

##### **stockQty = inventory[stockIdx].gettingQuantity();**

##### **if (stockQty < orderQty)**

##### **{**

##### **pendingOrder.addItem(orderItems[i]);**

##### **}**

##### **else**

##### **{**

##### **finalOrder.addItem(orderItems[i]);**

##### **inventory[stockIdx].setQuantity(stockQty - orderQty);**

##### **}**

##### **}**

##### **if (!pendingOrder.isEmpty())**

##### **{**

##### **pendingords.push\_back(pendingOrder);**

##### **}**

##### **if (finalOrder.isEmpty())**

##### **{**

##### **cout << "Sorry, none of your reqed items were in stock.\nOrder will be placed as and when items become available.\n\n";**

##### **return;**

##### **}**

##### **finalOrder.setPlacementTime();**

##### **finalOrder.setOrderId(ords.size() + 1);**

##### **ords.push\_back(finalOrder);**

##### **paymentCompleted.push\_back(false);**

##### **cuss[cusIdx].incrementOrder();**

##### **orderItems = finalOrder.gettingItems();**

##### **cout << "\nInvoice:\n";**

##### **cout << "Order ID:" << finalOrder.gettingOrderId() << "\n";**

##### **cout << "cus Identification Number:" << cuss[cusIdx].gettingcu\_No() << "\n";**

##### **cout << "Billing Address:" << cuss[cusIdx].gettingAdd() << "\n";**

##### **cout << "Items:\n";**

##### **long totalPrice = 0;**

##### **for (int i = 0; i < orderItems.size(); i++)**

##### **{**

##### **long CommoId = orderItems[i].gettingCommoId();**

##### **string itemName = searchCommo(CommoId).gettingCommoName();**

##### **cout << itemName << "\t" << orderItems[i].gettingQuantity() << "\n";**

##### **totalPrice += (orderItems[i].gettingQuantity() \* searchCommo(CommoId).gettingUnitPrice());**

##### **}**

##### **cout << "\nTotal billable amount:" << totalPrice << "\n\n";**

##### **}**

##### **void tradeHouse::handleIndentreq()**

##### **{**

##### **vector<vector<Item>> indent(dealers.size());**

##### **for (int orderIdx = 0; orderIdx < pendingords.size(); orderIdx++)**

##### **{**

##### **vector<Item> orderItems = pendingords[orderIdx].gettingItems();**

##### **for (int itemIdx = 0; itemIdx < orderItems.size(); itemIdx++)**

##### **{**

##### **Commo Commo = searchCommo(orderItems[itemIdx].gettingCommoId());**

##### **indent[Commo.gettingdealno()].push\_back(orderItems[itemIdx]);**

##### **}**

##### **}**

##### **cout << "\nIndents:\n";**

##### **for (int dealer = 0; dealer < dealers.size(); dealer++)**

##### **{**

##### **if (indent[dealer].empty())**

##### **continue;**

##### **cout << "Dealer Name:" << dealers[dealer].gettingDealerName() << "\n";**

##### **cout << "Items:\n";**

##### **for (int item = 0; item < indent[dealer].size(); item++)**

##### **{**

##### **cout << searchCommo(indent[dealer][item].gettingCommoId()).gettingCommoName()**

##### **<< "\t" << indent[dealer][item].gettingQuantity() << "\n";**

##### **}**

##### **cout << "\n";**

##### **}**

##### **//Replenish stock**

##### **for (int orderIdx = 0; orderIdx < pendingords.size(); orderIdx++)**

##### **{**

##### **vector<Item> orderItems = pendingords[orderIdx].gettingItems();**

##### **for (int itemIdx = 0; itemIdx < orderItems.size(); itemIdx++)**

##### **{**

##### **long inventoryIdx = gettingInventoryIndex(orderItems[itemIdx].gettingCommoId());**

##### **long newQty = inventory[inventoryIdx].gettingQuantity() + orderItems[itemIdx].gettingQuantity();**

##### **inventory[inventoryIdx].setQuantity(newQty);**

##### **}**

##### **long cusIdx = gettingcusIndex(pendingords[orderIdx].gettingcu\_No());**

##### **processOrder(pendingords[orderIdx], cusIdx);**

##### **}**

##### **pendingords.clear();**

##### **}**

##### **void tradeHouse::issuePayment()**

##### **{**

##### **long orderId;**

##### **cout << "\nEnter order number:";**

##### **cin >> orderId;**

##### **long orderIdx = gettingOrderIndex(orderId);**

##### **if (orderIdx == -1)**

##### **{**

##### **cout << "Invalid order ID. Please try again.\n\n";**

##### **return;**

##### **}**

##### **long cus\_No = ords[orderIdx].gettingcu\_No();**

##### **long custIdx = gettingcusIndex(cus\_No);**

##### **time\_t curr = time(0);**

##### **if (paymentCompleted[orderIdx])**

##### **{**

##### **cout << "\nPayment has already been completed.\n";**

##### **return;**

##### **}**

##### **if (difftime(curr, ords[orderIdx].gettingPlacementTime()) < TIME\_THRESHOLD)**

##### **{**

##### **cuss[custIdx].incrementPayment();**

##### **}**

##### **paymentCompleted[orderIdx] = true;**

##### **cout << "Payment Completed\n\n";**

##### **}**

##### **void tradeHouse::manaQuery()**

##### **{**

##### **clearRecords();**

##### **for (int orderIdx = 0; orderIdx < ords.size(); orderIdx++)**

##### **{**

##### **vector<Item> orderItems = ords[orderIdx].gettingItems();**

##### **for (int itemIdx = 0; itemIdx < orderItems.size(); itemIdx++)**

##### **{**

##### **long recordIdx = gettingRecordsIndex(orderItems[itemIdx].gettingCommoId());**

##### **if (recordIdx == -1)**

##### **{**

##### **continue;**

##### **}**

##### **else**

##### **{**

##### **long newQty = records[recordIdx].gettingQuantity() + orderItems[itemIdx].gettingQuantity();**

##### **records[recordIdx].setQuantity(newQty);**

##### **}**

##### **}**

##### **}**

##### **cout << "Sales of all items dealt:\n";**

##### **for (int recordIdx = 0; recordIdx < records.size(); recordIdx++)**

##### **{**

##### **cout << searchCommo(records[recordIdx].gettingCommoId()).gettingCommoName() << "\t";**

##### **cout << records[recordIdx].gettingQuantity() << "\n";**

##### **}**

##### **cout << "\n";**

##### **}**

##### **int main()**

##### **{**

##### **prepOverallcommolist();**

##### **tradeHouse house;**

##### **int choice;**

##### **cout << "Welcome to trade House Automation System!\n";**

##### **while (true)**

##### **{**

##### **cout << "Options:\n";**

##### **cout << "1. Register cust\n";**

##### **cout << "2. Place order\n";**

##### **cout << "3. Issue payment\n";**

##### **cout << "4. Handle indents\n";**

##### **cout << "5. Managerial query\n";**

##### **cout << "6. Exit\n";**

##### **cout << "Enter your choice:";**

##### **fflush(stdin);**

##### **cin >> choice;**

##### **switch (choice)**

##### **{**

##### **case 1:**

##### **house.registercus();**

##### **break;**

##### **case 2:**

##### **house.prepOrderList();**

##### **break;**

##### **case 3:**

##### **house.issuePayment();**

##### **break;**

##### **case 4:**

##### **house.handleIndentreq();**

##### **break;**

##### **case 5:**

##### **house.manaQuery();**

##### **break;**

##### **case 6:**

##### **exit(0);**

##### **break;**

##### **default:**

##### **cout << "Invalid choice. Please try again.\n\n";**

##### **break;**

##### **}**

##### **}**

##### **}**