Project 5 — Online Groceries

CS 2370

Background

With the COVID-19 quarantine, many people have ordered their groceries online for home delivery. This project will print a report of online orders from a fictional grocery chain. There are 3 input files to process:

- *customers.txt*
- *items.txt*
- *orders.txt*

The first file holds customer data in a single text line the following:

```
810003,Kai Antonikov,31 Prairie Rose Street,Philadelphia,PA,19196,215-975-7421,kantonikov0@4shared.com
```

The data fields are the customer id, name, street, city, state, zip, phone, and email.

The file *items.txt* has fields item_id, description, and price:

```
57464,Almonds Ground Blanched,2.99
```

Orders.txt holds **2 lines per order**. The first line contains the customer id, order number, order date, and then a variable-length list of `item_id-quantity` pairs:

```
762212,1,2020-03-15,10951-3,64612-2,57544-1,80145-1,27515-2,16736-1,79758-2,29286-2,51822-3,39096-1,32641-3, …
```

In the line above 3 items of product #10951 were ordered, 2 of product #64612, etc.

The second line contains payment information in the form of payment code (explained below), and payment information. There are 3 types of payments:

- 1 = credit card (card number and expiration date)
- 2 = PayPal (paypal_id only)
- 3 = wire transfer (bank_id and account_id)

Here are respective examples:

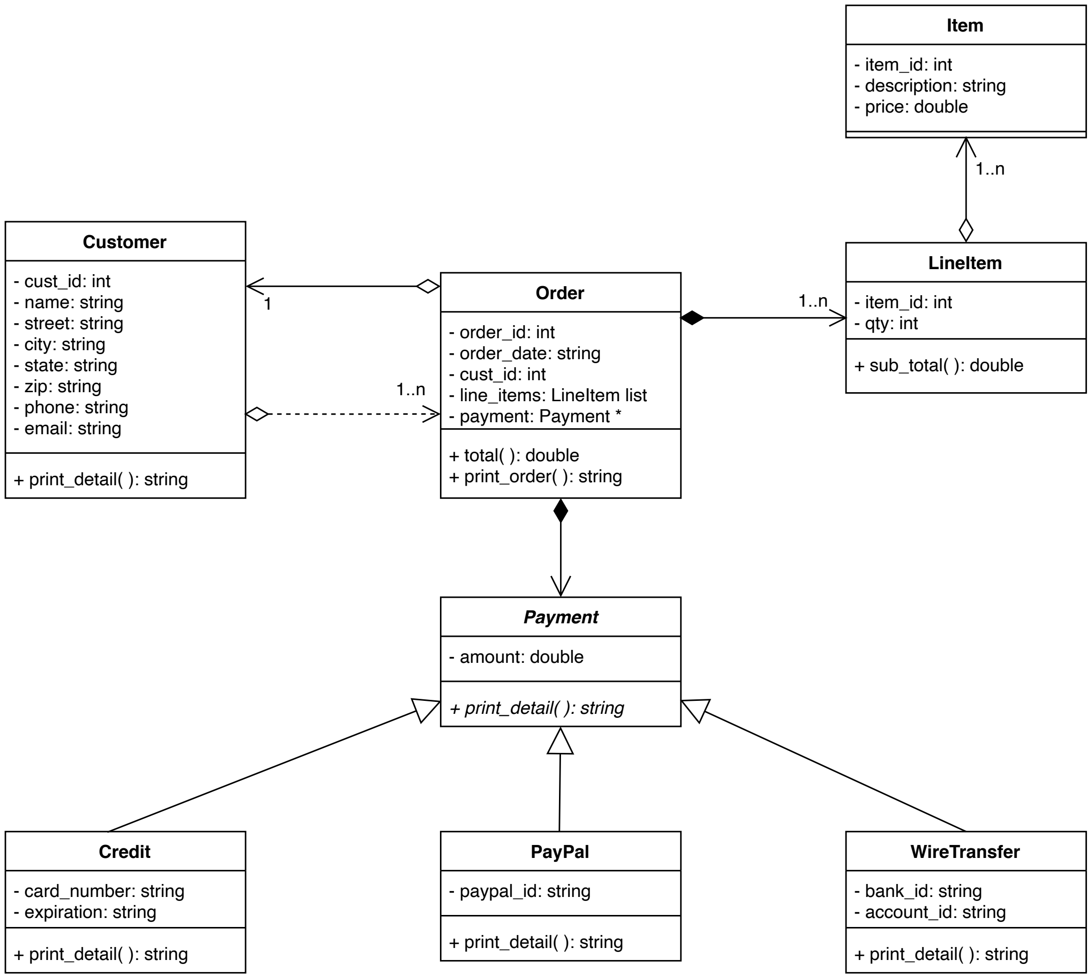
```
1,373975319551257,12-2023
2,cfeeham3s
3,72-2201515,68196-140
```

Note that all fields in all files are comma-separated, and that the `item_id-quantity` pairs in *orders.txt* are separated by a dash.

Requirements

To make this project manageable, observe the following:

- Write all code in a *single file*, *groceries.cpp*
- Define all classes with all functions *inline* (*in-situ* within each class)
- Use the design I provide in the following UML class diagram



All components of **Order** should be precomputed and passed to the **Order** constructor upon creation of the **Order** object. No setters!

Order::total computes the grand total for the order and sets the amount field in the **Payment** object directly.

Order::payment shouldn't be a raw pointer, use **unique_ptr**<Payment> and initialize it in **Order**'s initializer list.

Write the following global functions:

- **read_customers**( )
- **read_items**( )
- **read_orders**( )

Each of the above populate global vectors.

Figure 1 UML Class Diagram

Note that **Payment** is an *abstract class*. Its derived classes must override **print_detail** which returns a string which becomes part of printing an order as illustrated below. **Order::print_order** calls **Customer::print_detail** and **Payment::print_detail** to do its work.

I also provide the **main** function:

```
int main() {
    read_customers("customers.txt");
    read_items("items.txt");
    read_orders("orders.txt");

    for (const auto& order: orders)
        cout << order.print_order() << endl;
}
```

The first 3 functions are global functions you write. They create respectively global vectors of **Customer**, **Item**, and **Order** objects obtained from file input. I laid out my *groceries.cpp* as follows

```
[Code pertaining to Customers]

[Code pertaining to Items and LineItems]

[Code pertaining to Payments]

[Code pertaining to Orders]

[main function]
```

The order of the first two sections above are immaterial. The bulk of the work is done in **read_orders** and **Order::print_order**.

Your output should be a report in a file named *order_report.txt* that looks like the following:

```
===========================
Order #1, Date: 2020-03-15
Amount: $100.23, Paid by Credit card 373975319551257, exp. 12-2023

Customer ID #762212:
Yolanda McAlarney, ph. 505-136-7715, email: ymcalarney2u@wordpress.com
705 Corscot Hill
Albuquerque, NM 87190

Order Detail:
	Item 10951: "Syrup - Monin Swiss Choclate", 3 @ 3.00
	Item 16736: "Wine - Red Cooking", 1 @ 2.00
	Item 16974: "Pastry - Lemon Danish - Mini", 3 @ 1.19
	Item 23022: "Beef - Short Ribs", 1 @ 5.00
	Item 26461: "Bread - Crumbs Bulk", 1 @ 3.00
	Item 26860: "Waffle Stix", 2 @ 2.99
	Item 27515: "Longos - Burritos", 2 @ 4.00
	Item 29286: "Lemonade - Strawberry 591 Ml", 2 @ 2.25
	Item 32641: "Pie Filling - Cherry", 3 @ 0.89
	Item 39096: "Oil - Peanut", 1 @ 0.95
	Item 51822: "Pasta - Rotini Colour Dry", 3 @ 1.19
	Item 57544: "Peach - Halves", 1 @ 0.79
	Item 63725: "Black Currants", 3 @ 0.79
```

```
       Item 64007: "Juice - Propel Sport", 2 @ 1.99
       Item 64612: "Pie Shells 10", 2 @ 7.00
       Item 75536: "Quail - Whole Boneless", 2 @ 8.79
       Item 79758: "Beef Flat Iron Steak", 2 @ 5.49
       Item 80145: "Bread - Country Roll", 1 @ 2.29

===========================
Order #2, Date: 2020-03-15
Amount: $74.48, Paid by Credit card 201741963232463, exp. 02-2022

Customer ID #258572:
Garey Baraja, ph. 260-560-6065, email: gbaraja5r@fda.gov
42 Kenwood Parkway
Fort Wayne, IN 46862

Order Detail:
       Item 18329: "Scallops - In Shell", 1 @ 4.49
       Item 21316: "Tomatoes - Roma", 3 @ 2.50
       Item 22248: "Basil - Seedlings Cookstown", 2 @ 3.89
       Item 30346: "Pasta - Orzo Dry", 1 @ 0.99
       Item 37948: "Shrimp - Black Tiger 16/20", 3 @ 7.60
       Item 47680: "Sandwich Wrap", 3 @ 2.00
       Item 56107: "Numi - Assorted Teas", 1 @ 5.99
       Item 56833: "Jolt Cola", 1 @ 1.29
       Item 59695: "Dr. Pepper - 355ml", 1 @ 1.59
       Item 63498: "Vinegar - Raspberry", 1 @ 1.79
       Item 66295: "Bread - Dark Rye Loaf", 1 @ 2.99
       Item 70616: "Lemonade - Black Cherry 591 Ml", 1 @ 2.89
       Item 80237: "Juice - Orange", 2 @ 2.19
       Item 96497: "Scampi Tail", 1 @ 4.00

…
===========================
Order #8, Date: 2020-03-18
Amount: $113.63, Paid by Wire transfer from Bank ID 72-2201515, Account# 68196-140

Customer ID #217686:
Marisa Gossipin, ph. 239-305-6322, email: mgossipin1b@fema.gov
4 Charing Cross Lane
Fort Myers, FL 33994

Order Detail:
       Item 11770: "Rice - Jasmine Sented", 1 @ 2.69
       Item 12946: "Hagen Daza - Dk Choocolate", 1 @ 5.99
       Item 17714: "Flour Pastry Super Fine", 3 @ 4.95
       Item 21801: "Coffee - Ristretto Coffee Capsule", 2 @ 3.69
       Item 23276: "Juice Peach Nectar", 1 @ 2.45
       Item 24314: "Snapple - Mango Maddness", 1 @ 2.00
       Item 39096: "Oil - Peanut", 2 @ 0.95
       Item 39140: "Whmis Spray Bottle Graduated", 3 @ 1.99
       Item 49318: "Green Scrubbie Pad H.duty", 3 @ 3.99
       Item 54452: "Bananas", 2 @ 0.79
       Item 55222: "Artichoke - Fresh", 2 @ 2.39
       Item 59695: "Dr. Pepper - 355ml", 2 @ 1.59
       Item 67408: "Pasta - Penne Primavera Single", 3 @ 1.49
       Item 70461: "Pasta - Ravioli", 1 @ 1.25
       Item 79084: "Red Cod Fillets - 225g", 2 @ 5.59
       Item 79298: "Shiratamako - Rice Flour", 3 @ 3.49
       Item 89924: "Towel - Roll White", 2 @ 2.19
       Item 94416: "Shrimp - Baby Cold Water", 1 @ 4.99
       Item 98317: "Stainless Steel Cleaner Vision", 3 @ 4.05

===========================
Order #9, Date: 2020-03-18
Amount: $111.05, Paid by Paypal ID: tsantello5c
```

```
Customer ID #196547:
Alison Threader, ph. 703-698-2694, email: athreader5c@zimbio.com
50 Shopko Plaza
Washington, DC 20041

Order Detail:
       Item 20755: "Sansho Powder", 2 @ 2.05
       Item 21809: "Salmon - Atlantic Fresh Whole", 3 @ 4.75
       Item 23022: "Beef - Short Ribs", 2 @ 5.00
       Item 25956: "Energy Drink - Franks Pineapple", 3 @ 2.00
       Item 37019: "Instant Coffee", 3 @ 6.00
       Item 44214: "Wakami Seaweed", 1 @ 4.00
       Item 47680: "Sandwich Wrap", 3 @ 2.00
       Item 48704: "Beans - French", 1 @ 1.05
       Item 51822: "Pasta - Rotini Colour Dry", 1 @ 1.19
       Item 54044: "Sole - Fillet", 2 @ 3.99
       Item 57544: "Peach - Halves", 1 @ 0.79
       Item 64612: "Pie Shells 10", 2 @ 7.00
       Item 67193: "Cheese - Cheddar Old White", 2 @ 1.88
       Item 84418: "Soup - Knorr Country Bean", 2 @ 1.69
       Item 90349: "Soup - Campbells Beef Noodle", 3 @ 1.19
       Item 90475: "Chicken - Whole", 2 @ 6.49
…
===========================
Order #609, Date: 2020-12-30
Amount: $101.69, Paid by Credit card 3538527630422753, exp. 04-2024

Customer ID #409485:
Jaye Martinho, ph. 361-111-4183, email: jmartinho7x@dion.ne.jp
2339 Magdeline Plaza
Corpus Christi, TX 78410

Order Detail:
       Item 12527: "Bagelers - Cinn / Brown Sugar", 3 @ 3.99
       Item 12568: "Cabbage - Red", 2 @ 0.69
       Item 16724: "Blueberries", 3 @ 2.99
       Item 17981: "Halibut - Fletches", 1 @ 8.00
       Item 32641: "Pie Filling - Cherry", 3 @ 0.89
       Item 41142: "Vinegar - Red Wine", 3 @ 1.25
       Item 44214: "Wakami Seaweed", 1 @ 4.00
       Item 56826: "Flour - Bread", 2 @ 4.00
       Item 57464: "Almonds Ground Blanched", 2 @ 2.99
       Item 58524: "Thyme - Lemon Fresh", 1 @ 2.19
       Item 64067: "Magnotta Bel Paese Red", 2 @ 3.39
       Item 67408: "Pasta - Penne Primavera Single", 3 @ 1.49
       Item 78265: "Syrup - Monin Irish Cream", 3 @ 3.00
       Item 80145: "Bread - Country Roll", 2 @ 2.29
       Item 81169: "Table Cloth 90x90 White", 1 @ 3.99
       Item 86456: "Rice - Long Grain", 1 @ 2.29
       Item 86494: "Crackers - Trio", 2 @ 2.35
       Item 95662: "Garlic Powder", 3 @ 2.99
```

Each order begins with a dashed line and has 3 parts, each followed by an empty line. The orders appear in the order they are read from the file, but the items in each order are sorted by **item_id**.

Submit your *groceries.cpp* and *order_report.txt* files in Canvas by the due date.

Implementation Notes

I have provided a file, *split.h*, which contains a **split** function that returns all fields that were separated by some character as a vector of strings:

```
#include "split.h"
```

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;

int main() {
    string s = "715608,Vergil Heelis,61070 Marcy Park,Fort Worth,TX,76115,682-583-
7160,vheelis4@blogger.com";
    auto fields = split(s, ',');
    for (const auto& fld: fields)
        cout << fld << endl;
    cout << endl;
}

/* Output:
715608
Vergil Heelis
61070 Marcy Park
Fort Worth
TX
76115
682-583-7160
vheelis4@blogger.com
*/
```

You can also use this function to separate the `item_id-quantity` pairs.

To call **std::sort** to sort the line items of an order, add the following to your **LineItem** class:

```
   friend bool operator<(const LineItem& item1, const LineItem& item2) {
        return item1.item_id < item2.item_id;
   }
```

Inside **Order** you need to add a new keyword that we haven't discussed when declaring the **LineItem**s the **Order** objects hold:

```
  mutable vector<LineItem> line_items;
```

The **mutable** keyword allows **line_items** to be modified even inside of **const** member functions. This needs to be done on occasion, so the **mutable** keyword exists for this very purpose.

Note that to achieve runtime polymorphism with the **Payment** objects, the payment member of **Order** must be a *pointer***.** In our case, the concrete **Payment** instances (**Credit**, **PayPal**, **WireTransfer**) are allocated on the heap using the **new** operator. To avoid memory-management problems, we declare **Order::payment** as `unique_ptr<Payment> payment;`. Your **read_orders** function needs to check the payment code (1, 2, or 3) and create the appropriate payment subtype, storing it in a `Payment*`. It is then passed to the `unique_ptr<Payment>` member in the **Order** constructor, which is then added to your global vector of orders. When you add entries to a vector, use **emplace_back** instead of **push_back** so you don't have to create a temporary object.

This is the most challenging program of the semester. My *groceries.cpp* has about 250 lines of code, and it's pretty succinct. Start early and ask questions.

Here is a suggested development sequence for you to follow:

Start with **main**, as written.
Comment out the loop that does the **print_order**
Create empty-body functions the other three.
Run it and it will do nothing (just to have it compile correctly)

Write the **Customer** class
Declare the global **Customer** vector
Write **read_customers()** to fill in the vector
Write **Customer::print_detail()**. Use it to verify you read in the data correctly.

Follow similar procedure as above for **Items**

Do the **Payment** classes, and generate different kinds of payments. Verify that they print correctly

Do the **Orders** class

Do **read_orders**

Do **Orders::print_order**

FAQs

Q. How do I find the Customer or Item entries in the global arrays given a cust_id or item_id?
A. You'll have to search for them. You can write a loop, but you might find it easier to use **std::find_if** from <**algorithm**>. Write a function to do the job, regardless. I added the following functions to my solution:

```
int find_cust_idx(int cust_id);
int find_item_idx(int item_id);
```

Q. I heard global data is "evil".
A. It can be problematic, but it makes this project doable in the time allotted by keeping things simple. In production we would probably have a Grocery class and the global data from our project would be static data in that class, but that's an implementation detail not worth worrying about here. Static data is stored the same way as global data—only the access is different.

Q. Why didn't you make all the data of type **std::string**?
A. Well, prices must be **double**s so you can do arithmetic. Also, I want you to use **std::stoi** and **std::stod** for practice.

Q. Why is this project harder than the others?
A. Because you can't see the benefit of object-oriented programming on tiny projects. OOP was invented to organize larger projects. This is the smallest one I could think of that had virtual functions and would help you see the practical, organizational power of OOP.

Q. How do I read the variable number of item-quantity pairs?
A. **split** did the work for you when you first called it. Then split each pair on a dash.