The C Family - CA2

ECM2433

How To Run?

(NOTE: ordering is compiled for a linux machine)

In a terminal...

- Unzip `**order-processing-system.zip**`
- navigate to the `**order-processing-system**` directory
- compile the program with command: `**./compileOP**`
- then run the program with command: `**bin/./ordering testInputFile.txt**`
- IF you have your own test file then replace `**testInputFile.txt**`

General Program Design

Methods were used to break down the programs steps into manageable and associated segments of code.

First **loadInputFileData** was used to load the **testInputFile.txt** data into a vector of strings, thus dealing with the opening and closing of the input stream within a single method.

runOrderProcessingSystem manages all the processing of Customers, Orders, and End-Of-Day, data entries. It also acts as a partial safety net for variable management as any local variables created will be removed unless declared as static or allocated in memory.

processCustomer and **processOrder** both deal with validation, instantiation and output of their respective records.

finaliseOrders deals with the deletion of order objects for a single customer, then outputs shipping and invoice messages.

processEndOfDay outputs the end-of-day message, then invokes finalisesOrders for each customer who has a order quantity greater than zero.

Class Design

Why there are no derived / sub classes...
Originally the Express class was to be derived from the Order class. The idea became redundant once it was realised the only comparable difference between orders was the type of 'X' or 'N', thus the variable "orderType" was added which removed the need to derive an Express class.

Customer Class (customer.hpp, customer.cpp)
Customer has been designed to hold a pointer for every **new** order created. Upon a customer being shipped

their orders the pointers are used to iterate through and **delete** every order associated with the customer. Note, the Customer class does not delete orders directly but instead is used to get all its orders.

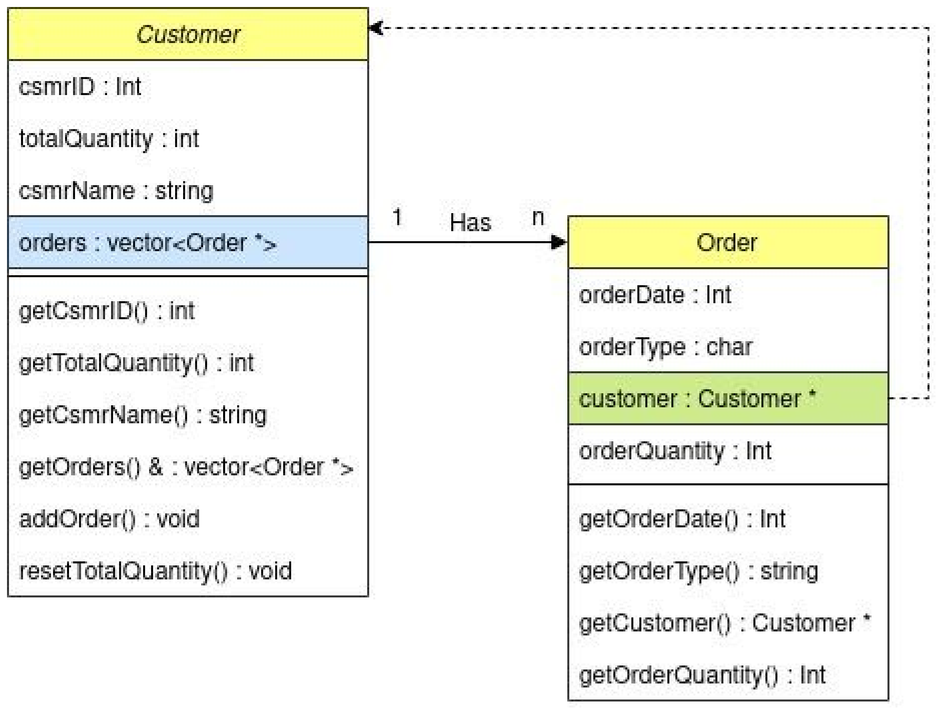
The design choice to store order object pointers was for two reasons:

1. It would model a design similar to a relational database, each order pointer mimics a forign key.
2. With all associated orders for a single given customer being held in a collection of pointers, managing the program's memory becomes far more streamlined. All orders allocated as "**new**" are stored as pointers within their customers.

Order Class (order.hpp, order.cpp)
Order stores all data from the input line as variables except the customer number. Instead the customer number is replaced with a pointer to the actual customer instance, since #Assumptions states "orders have a customer number for an already existing customer" and thus it makes more sense to have a pointer. Additionally by having an order point to its associated customer instance, all computations involving retrieving the order's customer details are removed as we have a direct reference to said customer.

To better demonstrate the design...

Class UML Diagram



Assumptions

No.	Assumption
1	A single input file populated with data will be the only source of data the program processes

2	The input file dates will correctly be in the format YYYYMMDD and nothing else.
3	Any extra characters after the initial input line will be whitespace and thus ignored.
4	An input line not starting with ‘C’, ‘S’, or ‘E’ should be considered invalid and halt the program.
5	There cannot be any duplicate customers and upon finding one an error should be given.
6	All orders have a customer number for an already existing customer otherwise an error should be given.
7	All orders are exclusively either ‘normal’ or ‘EXPRESS’ and any other type is considered invalid.
8	All order quantities will be greater than zero.

Known Issues

- **String Conversions Exceptions:** If **stoi()** or **to_string()** fail they will throw an exception and halt the program. I could not find a way to override these exceptions in order to report more information to identity the cause of the exception.
  i.e. print the customer number in the original input string, say the number was meant to be 0001 but 0X01 was given.

- **Odd or Incorrect Dates:** So long as a date is an integer then it will be considered correct (as mentioned in #Assumptions).
  Should an input date be 00010203 the program will print the date as 10203. Since the specification declared date as an integer I did not attempt to resolve this issue due to the assumption it will always be presented in the correct format, and it is not critical to the programs computational tasks as all dates are used for printing output.