University of Exeter

College of Engineering, Mathematics

and Physical Sciences

ECM2433

The C Family

Continuous Assessment

Date Set:	22nd February 2021
Date Due:	18th March 2021
Return Date:	26th April 2021

This CA comprises 30% of the overall module assessment.

This is an **individual** exercise and your attention is drawn to the College and University guidelines on collaboration and plagiarism, which are available from the College website.

This is the **second** coursework for this module and tests your skills and understanding of programming in C++.

Please Turn Over

1 Problem Statement

You are working for the ACME Vaccine Shipping Company and have been given the task of creating a C++ program to process customer orders. The company only ships one product: packs of the Oxford-AstraZeneca Covid-19 vaccine.

Each customer places multiple orders during each day. At the end of each day the orders are collated and each customer's total order quantity is sent to them as a single shipment. If a customer places an "express" order, then it and all its outstanding orders for the day are sent immediately (i.e. not waiting for the end of the day).

When a shipment is triggered for a customer, they are notified that a shipment is on its way, and an invoice is sent to them.

2 The Task

Your task is to write a well-structured, C++11 standard C++ program (with executable called `ordering`) to process customer orders. To simulate the arrival of orders, a data file containing the orders will be passed to the program at runtime (see section 2.1 for details). The name of this data file will be passed to the program as the first parameter on the command line, like this:

```
ordering orderFile.txt
```

The file contains records of three different formats: *new customer* records, *sales order* records, and *end-of-day* records. *Sales order* records are either *normal* orders or *express* orders. Each record in this order file is processed in turn and results in one of four actions:

1. Add a new customer. Each customer is identified by a unique *customer number*.
2. Add a new *normal order*. The order quantity is added to the customer's current order total.
3. Add a new *express order*. The order quantity is added to the customer's current order total and that total quantity is then shipped to the customer. The customer's order quantity is then reset to zero.
4. End-of-day processing. For each customer in the system whose order quantity is not zero, that quantity is shipped to the customer and the customer's order quantity is then reset to zero.

When a shipment is made for a customer, they are notified by being sent an invoice. Invoice numbers start at 1000 and the number is incremented by 1 each time a new invoice is generated. An invoice consists of the following information: customer number, customer name, invoice number, shipment date, shipment quantity.

The output from the `ordering` program is a series of messages sent to the standard output stream. The order processing system will generate messages that are all prefixed with "OP:"; messages generated by customers will all be prefixed with "SC:". One order processing message will be generated for each of the following actions:

- A new customer is added.
- A new *normal order* is processed.
- A new *express order* is processed.
- A shipment is sent to a customer.
- The end of a day has been reached.

One customer message will be generated each time the customer receives an invoice.

In the following subsections, section 2.1 defines the format of the customer orders file and provides an example (which is also available on ELE), section 2.2 describes the actions that the program must take in response to that example file, and section 2.3 describes the messages that the program must output when those actions are completed.

2.1 Input file format

An example of the format of this file is as follows (this example file is uploaded to ELE):

line	contents
1	C0001Royal Devon & Exeter Hospital
2	C0002Derriford Hospital
3	C0003Torbay Hospital
4	S20210201N0001040
5	S20210201N0001050
6	E20210201
7	S20210202N0001040
8	S20210202N0001060
9	S20210202N0002050
10	S20210202N0002170
11	E20210202
12	S20210203N0001050
13	S20210203N0002065
14	S20210203N0003150
15	S20210203X0001190
16	S20210203N0002110
17	E20210203

The file contains records of three different types, indicated by the value in the first column: a 'C' indicates that this is a *new customer* record, while an 'S' indicates a *sales order*, and an 'E' an *end-of-day* record.

2.1.1 New customer record

The format of a *new customer* record is as follows:

column	datatype	description
1	character	always 'C'
2-5	integer	the customer number; a four digit, zero-padded integer
6-45	string	the customer's name

2.1.2 Sales order record

The format of a *sales order* record is as follows:

column	datatype	description
1	character	always 'S'
2-9	integer	the order date, in the format YYYYMMDD
10	character	'N' for a normal order; 'X' for an express order
11-14	integer	the customer number of the customer raising the order
15-17	integer	the order quantity; a three-digit, zero-padded integer

2.1.3 End-of-day record

The format of an *end-of-day* record is as follows:

column	datatype	description
1	character	always 'E'
2-9	integer	the date, in the format YYYYMMDD, for which this is the end of day

2.2 System actions

The example file shown in section 2.1 will result in the program taking the following actions:

line	**message**
1	new customer added, with customer number 1 and current order quantity 0
2	new customer added, with customer number 2 and current order quantity 0
3	new customer added, with customer number 3 and current order quantity 0
4	customer 1: order quantity is incremented by 40
5	customer 1: order quantity is incremented by 50
6	end of day 20200201: quantity of 90 shipped to customer 1; customer 1 order quantity reset to 0
7	customer 1: order quantity is incremented by 40
8	customer 1: order quantity is incremented by 60
9	customer 2: order quantity is incremented by 50
10	customer 2: order quantity is incremented by 170
11	end of day 20200202: quantity of 100 shipped to customer 1; customer 1 order quantity reset to 0 quantity of 220 shipped to customer 2; customer 2 order quantity reset to 0
12	customer 1: order quantity is incremented by 50
13	customer 2: order quantity is incremented by 65
14	customer 3: order quantity is incremented by 150
15	customer 1: express order: order quantity is incremented by 190 quantity of 240 shipped to customer 1; customer 1 order quantity reset to 0
16	customer 2: order quantity is incremented by 110
17	end of day 20200203: quantity of 175 shipped to customer 2; customer 2 order quantity reset to 0 quantity of 150 shipped to customer 3; customer 3 order quantity reset to 0

2.3 System messages

The actions described in section 2.2 must result in the following messages being output by the system. The "SC" messages from the customers might be generated asynchronously from the "OP" messages from the order processing system, so the relative ordering of those might be different from that shown below.

line	**action**
1	OP: customer 0001 added
2	OP: customer 0002 added
3	OP: customer 0003 added
4	OP: customer 0001: normal order: quantity 40
5	OP: customer 0001: normal order: quantity 50
6	OP: end of day 20200201 OP: customer 0001: shipped quantity 90 SC: customer 0001: invoice 1000: date 20200201: quantity: 90
7	OP: customer 0001: normal order: quantity 40
8	OP: customer 0001: normal order: quantity 60
9	OP: customer 0002: normal order: quantity 50
10	OP: customer 0002: normal order: quantity 170
11	OP: end of day 20200202 OP: customer 0001: shipped quantity 100 SC: customer 0001: invoice 1001: date 20200202: quantity: 100 OP: customer 0002: shipped quantity 220 SC: customer 0002: invoice 1002: date 20200202: quantity: 220
12	OP: customer 0001: normal order: quantity 50
13	OP: customer 0002: normal order: quantity 65
14	OP: customer 0003: normal order: quantity 150
15	OP: customer 0001: EXPRESS order: quantity 190 OP: customer 0001: shipped quantity 240 SC: customer 0001: invoice 1003: date 20200203: quantity: 240
16	OP: customer 0002: normal order: quantity 110
17	OP: end of day 20200203 OP: customer 0002: shipped quantity 175 SC: customer 0002: invoice 1004: date 20200203: quantity: 175 OP: customer 0003: shipped quantity 150 SC: customer 0003: invoice 1005: date 20200203: quantity: 150

3 Deliverables

The deliverables for this coursework comprise the source code for your customer ordering system, instructions for compiling and linking it into an executable, and your report, zipped together and submitted via EBART. Your submission must contain the following:

- The source code must be written in well-structured, C++11 standard C++. Note that while the whole program could be written in C and compile as a "C++11 standard C++" program, marks will be awarded for the "C++"-ness of it.
- There must be a Linux shell script for compiling and linking your code, called `compileOP` and nothing else, and must compile your code to an executable called `ordering` and nothing else.
- A report, as detailed in section 3.1 below.

Your program must compile, link and execute on one of the two Linux servers we are using on this module (`emps-ugcs1` and `emps-ugcs2`) and will be tested on one of those servers using input files other than the example one supplied on the ELE page.

Submission must be made by 12pm (noon) on the date indicated on the front page of this document.

3.1 Report

As well as the code, you must submit a report (maximum 3 pages of A4). This must describe the overall design of your system, in particular how the class(es) you have implemented support the functionality, and any assumptions you have made. If your submitted code does not work fully as expected, then describe the details in the report, including any testing you have performed to narrow down what is causing the problem.

4 Marking Scheme

Criteria	Marks
Program basics *To what extent is your program a well-constructed and well-formatted C++11 standard C++ program? To what extent does your program use C++ constructs rather than C constructs? Does your program compile and link without errors and warnings? To what extent does it trap and report run-time errors? To what extent does your program correctly read in the command line parameters?* Processing of the order file *To what extent does your program correctly read the data from the file specified on the command line?*	40
Function of the system *To what extent does the program make a suitable representations of the entities in the system? To what extent does the program correctly and efficiently process the orders, and output the messages to the standard output stream?*	40
Report *To what extent does the report clearly and concisely describe the overall design of your system, in particular how the class(es) you have implemented support the functionality, and any assumptions you have made. If the code does not work fully, to what extent have the errors been investigated and described in the report? Are the spelling, grammar, language and presentation of the report clear, formal and professional and appropriate to the intended audience?*	20
Total	100

Penalties

You will be penalised 10 marks if the executable code is not called `ordering`, or the Linux shell script for compiling and linking it is not called `compileOP`.

If your report exceeds 3 pages, only the first 3 pages will be marked.