Final Exam

Instruction

Time

Dec 11 (Thu) from 8:00 AM to Dec 12 (Fri) 7:59 AM (total 24 hours).

Submission

You will need to upload the following to Canvas:

1. **All source code files**
2. **Instructions** on how to compile and run your program
3. **A 3–5 minute demo video** that includes:
   (a) A brief explanation of your programming logic (key idea)
   (b) Compilation and execution of your program
   (c) Demonstration of test results using the provided example test code
   (d) **Your face must appear in the video** so we can verify your identity
   (You do **NOT** need to record your entire programming process.)

Grading (100 pts total)

- **50 pts** — Source code of all required classes/functions
- **10 pts** — Runnable program
- **10 pts** — Correctness (correct output for the provided test code)
- **30 pts** — Demo video:
  - 10 pts: clarity
  - 10 pts: following the video instructions
  - 10 pts: showing your face

Problem 1: Simple Library Borrowing System

In this problem, you will design a C++ class hierarchy for a simple library borrowing system. The goal is to apply and demonstrate core object-oriented and STL concepts, including classes, inheritance, virtual functions and polymorphism, and the use of an STL container.

Requirements

1. Base Class: `Item`

Create the base `Item` with the following components:

- **Members:**
  - the title of the item
  - a flag indicating whether the item is currently borrowed or not
  - the number of days the item is borrowed for
- **Methods:**
  - `available()`: returns whether the item is not currently borrowed
  - `borrow(days)`: marks the item as borrowed and records the number of days
  - `give_back()`: resets the item to an available state
  - `getTitle()`: returns the title of the item
  - `info()`: A pure virtual function which returns a string describing the item.

2. Derived Class: `Book`

The `Book` class should inherit from `Item` and include:

- the author's name
- the publication year
- an overridden `info()` method that returns the string:

```
Book: <title> by <author> (<year>)
```

3. Derived Class: `Magazine`

The `Magazine` class should inherit from `Item` and include:

- the issue number
- an overridden `info()` method that returns the string:

```
Magazine: <title>, Issue <number>
```

4. Library Class

Create a `Library` class that stores items (books and magazines) using an STL container of your choice (e.g.,`std::map`, `std::unordered_map`).

The container must store **polymorphic** objects (e.g., via `std::unordered_map<std::string, Item*>`).

The `Library` class must provide the following functions:

- `addItem(item)`: adds an item to the library
- `checkAvailability(title)`: returns whether the item with the given title is available
- `borrowItem(title, days)`: marks the item as borrowed for the specified number of days
- `returnItem(title)`: returns the item and resets its borrowing status
- `listItems()`: prints each item's `info()` along with its status ("Available" or "Borrowed for X days")

5. Test Code

The test code is provided on Canvas at `Files/Exam/Final/final_1.cpp`. You will need to complete the `TODO` sections, which correspond to the required classes and functions described above.

Problem 2: Managing Store Inventory Using STL

You are helping a small grocery store manage its inventory using C++ and the STL. Each product in the store has:

- a product name,
- a quantity in stock,
- a unit price.

You may define a simple `struct Product` to store these fields.

The initial inventory contains the following products:

Name	Quantity	Price
Apple	50	1.29
Milk	8	2.99
Bread	12	2.49
Eggs	24	3.99
Cheese	5	4.59
Chicken	15	7.99
Orange	10	1.19
Banana	30	0.79

Requirements

1. Initialization

Use `std::vector<Product>` to store the list of products. Construct an `std::unordered_map<std::string, int>` that maps each product name to its index in the vector (using iterators). Implement `initializeInventory()` to initialize the containers.

2. Fast item lookup

Implement a function `Product* findProduct()`, that uses the `unordered_map` to perform $O(1)$ name lookup and returns a pointer to the corresponding `Product` inside the vector (or `nullptr` if it does not exist).

3. Low-stock products.

Implement a function `filter_lower()` that takes the `vector<Product>` and a threshold, and returns a `std::vector` containing all products whose quantity is lower than the given threshold.

4. Sorting by quantity.

Implement the function `sortAndRefresh()` to sort the `vector<Product>` by quantity. Use `std::sort` with a lambda expression. Since `std::sort` is an in-place operation, you must also update the `unordered_map` to keep the name–index mapping consistent.

Hint: You may use a lambda expression as the comparator. A lambda has the general form:

```
[](T& x, T& y) { /* return a boolean comparison */ };
```

Replace the placeholder type `T` and define the comparison based on product quantity in the lambda expression body.

5. Increase stock.

Implement the `restockProducts()` function to increase the stock of products in the low-stock list returned by `filter_lower()` in (3). You may use your `findProduct()` function to quickly locate each product and update its quantity.

6. Test Code

The test code is provided on Canvas at `Files/Exam/Final/final_2.cpp`. You will need to complete the `TODO` sections, which correspond to the required classes and functions described above.