UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science
Faculty of Engineering, Built Environment & IT
University of Pretoria

COS110 - Program Design: Introduction

Practical 2 Specifications

Release Date: 05-08-2024 at 06:00

Due Date: 09-08-2024 at 23:59

Total Marks: 85

Contents

1 General Instructions

- ***Read the entire assignment thoroughly before you start coding.***
- This assignment should be completed individually, no group effort is allowed.
- **To prevent plagiarism, every submission will be inspected with the help of dedicated software.**
- Be ready to upload your assignment well before the deadline, as **no extension will be granted.**
- If your code does not compile, you will be awarded a mark of 0. The output of your program will be primarily considered for marks, although internal structure may also be tested (eg. the presence/absence of certain functions or classes).
- Failure of your program to successfully exit will result in a mark of 0.
- Note that plagiarism is considered a very serious offence. Plagiarism will not be tolerated, and disciplinary action will be taken against offending students. Please refer to the University of Pretoria's plagiarism page at `https://portal.cs.up.ac.za/files/departmental-guide/`.
- Unless otherwise stated, the usage of c++11 or additional libraries outside of those indicated in the assignment, will **not** be allowed. Some of the appropriate files that you have submit will be overwritten during marking to ensure compliance to these requirements. **Please ensure you use c++98**
- All functions should be implemented in the corresponding `cpp` file. No inline implementation in the header file apart from the provided functions.
- The usage of ChatGPT and other AI-Related software is strictly forbidden and will be considered as plagiarism.
- Note, UML notation for variable and function signatures are used in this assignment.

2 Overview

In this practical you will implement an item and an inventory class. The goal of the practical is to get a basic understanding of classes and parameterised constructors as well as getting more comfortable with dynamic memory.

3 Your Task:

You are required to implement the following class diagram illustrated in Figure 1. Pay close attention to the function signatures as the `h` files will be overwritten, thus failure to comply with the UML, will result in a mark of 0.

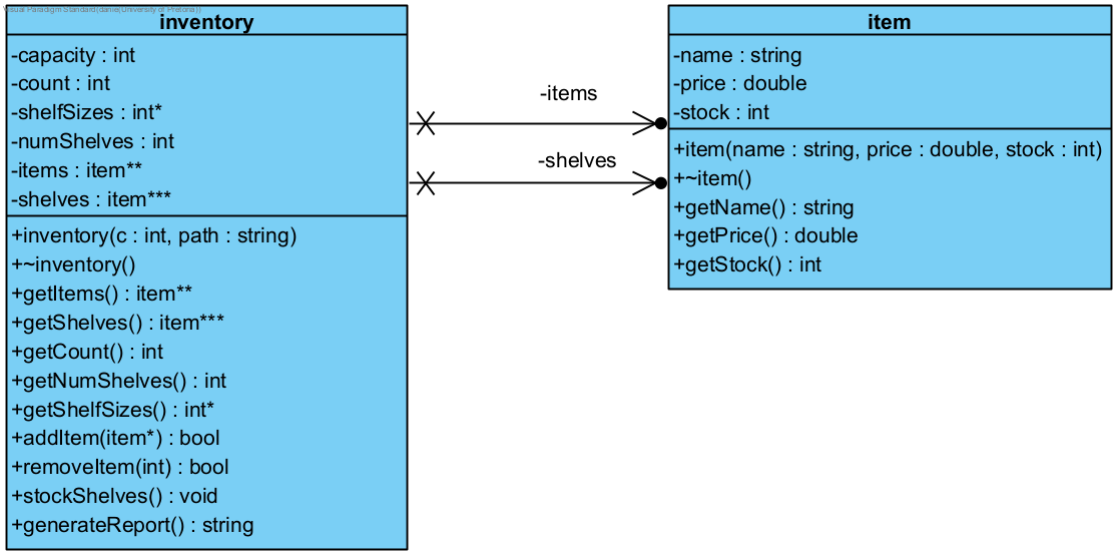

Figure 1: Class diagrams

Note, the importance of the arrows between the classes are not important for COS 110 and will be expanded on in COS 214.

3.1 item

The item class is used to represent the items that get stored in the inventory.

3.1.1 Members

- -name: string
  - This member variable contains the name of the item like: milk, eggs, pen, etc.
- -price: double
  - This member variable contains the price of the item.
- -stock: int
  - This member variable contains the number of items in stock.

3.1.2 Functions

- +item(name : std.string, price : double, stock : int)
  - This is the constructor for the item class.
  - It should assign the paramaters to the corresponding member variables.
- +~item()
  - This is the destructor for the item class.
  - It should deallocate all dynamic memory allocated to the current object, however since there are no dynamic memory in the item class, you can leave the implementation empty.

- +getName(): string
  - This is the get method for the name member variable.
  - It should return the name of the item.
- +getPrice(): double
  - This is the get method for the price member variable.
  - It should return the price of the item.
- +getStock(): string
  - This is the get method for the stock member variable.
  - It should return the stock of the item.

3.2 inventory

3.2.1 Textfile Format

To populate the items in the inventory you will get input from a textfile of the following format:

```
Name#Price#Stock
```

Here is an example textfile:

```
Toothpaste#257.65#46.00
Chocolate#308.04#15.00
Notebook#521.33#39.00
Toothpaste#978.62#47.00
Lotion#217.67#29.00
Pen#847.72#31.00
Milk#118.83#50.00
Chocolate#864.70#11.00
Cheese#857.55#38.00
Toothpaste#224.88#42.00
```

3.2.2 Members

- -capacity: int
  - This member variable contains the maximum items that the inventory can hold.
- -count: int
  - This member variable contains the current number of items in the inventory.
- -shelfSizes: int*
  - This member variable is a dynamic array that contains the number of items on each shelf.
- -numShelves: int

  - This member variable contains the total shelves used to stock all the items.

- -items: item**
  - This member variable is a dynamic array of dynamic item objects.
  - It represents all the items in the inventory.

- -shelves: item***
  - This member variable is a dynamic 2-dimensional array of dynamic item objects.
  - It represents shelves on which the items need to be stored.

3.2.3 Functions

- +inventory(c: int, path: string)
  - This is the constructor for the inventory class.
  - It takes two parameters: The maximum capacity of the inventory and the name of the items text file.
  - Set the capacity member variable to the passed in parameter.
  - Initialize shelfSizes and shelves to NULL and numShelves to 0.
  - Lastly you should initialise the items array and populate it with items corresponding to the items in the text file. Each line of the text file should correspond to a single item object.
  - Remember to not exceed the capacity and to update the count member variable.

- +~inventory()
  - This is the destructor for the inventory class.
  - It should deallocate all dynamic memory allocated to the current object. Therefore it should deallocate shelfSizes, items and shelves.

- +getItems(): item**
  - This is the get method for the items member variable.
  - It should return the items in the inventory.

- +getShelves(): item***
  - This is the get method for the shelves member variable.
  - It should return the shelves in the inventory.

- +getCount(): int
  - This is the get method for the count member variable.
  - It should return the count of the items in the inventory.

- +getNumShelves(): int
  - This is the get method for the numShelves member variable.
  - It should return the number of shelves in the inventory.
- +getShelfSizes(): int*
  - This is the get method for the shelfSizes member variable.
  - It should return the shelfSizes in the inventory.
- +addItem(item*): bool
  - This is a method to add an item to the inventory.
  - It should return true if the item was added and false otherwise.
  - An item can not be added to a full inventory.
  - If an item can be added, the passed in item parameter should be added to the end of the items array. *Note: remember to make a deep copy.*
- +removeItem(int): bool
  - This is a method to remove an item from the inventory.
  - It should return true if the item was removed and false otherwise.
  - The passed in parameter is the index of the item that should be removed.
  - An item cannot be removed from a empty inventory or if the index is higher than the number of items in the inventory.
  - If an item can be removed, it should be deleted and all the other items in the array need to be shifted down. Therefore the NULL items will be at the end of the array.
- +stockShelves(): void
  - This method should ”pack” the items on shelves. It creates and populate the shelves array.
  - The array is a 2-dimensional dynamic jagged array, meaning that each shelf can have a different number of items on it.
  - It is important that the items on the shelves are deep copies of the items in the items array.
  - This method also initialise and populate the shelfSizes array and determine the numShelves member variable.
  - The rows of the array correspond to the number of shelves.
  - Items with the same name should be on the same shelf.
  - shelfSizes is used to keep track of the number of items on each shelf. Therefore it correspond to the columns of the array.
  - It is important to make deep copies.

- +generateReport(): string
  - This method generates a report for the shelves of the inventory and returns a string.
  - The format of the string is as follows: Shelf shelfNumber: Item: name Total Stock: total Average Price: avg Min Price: min Max Price: max
  - After each shelf you have to add a newline character, see the example below of a string returned from the function.
  - Total Stock is the sum of the stock of all the items on the shelf.
  - Average Price is the average price of the items on the shelf.
  - Min Price is the cheapest item on the shelf.
  - Max Price is the most expensive item on the shelf.
  - The following is an example of a report for an inventory with two shelves:

```
Shelf 0: Item: Notebook Total Stock: 211 Average Price: 444.404 Min Price: 21.19 Max Price: 906.49
Shelf 1: Item: Soap Total Stock: 311 Average Price: 381.403 Min Price: 67 Max Price: 829.8
```

4 Memory Management

As memory management is a core part of COS110 and C++, each task on FitchFork will allocate approximately 10% of the marks to memory management. The following command is used:

```
valgrind --leak-check=full ./main
```

Please ensure, at all times, that your code *correctly* de-allocates *all* the memory that was allocated.

5 Testing

As testing is a vital skill that all software developers need to know and be able to perform daily, 10% of the assignment marks will be allocated to your testing skills. To do this, you will need to submit a testing main (inside the `main.cpp` file) that will be used to test an Instructor Provided solution. You may add any helper functions to the main.cpp file to aid your testing. In order to determine the coverage of your testing the gcov [1] tool, specifically the following version *gcov (Debian 8.3.0-6) 8.3.0*, will be used. The following set of commands will be used to run gcov:

```
g++ --coverage *.cpp -o main
./main
gcov -f -m -r -j ${files}
```

[1] For more information on gcov please see `https://gcc.gnu.org/onlinedocs/gcc/Gcov.html`

This will generate output which we will use to determine your testing coverage. The following coverage ratio will be used:

$$\frac{\text{number of lines executed}}{\text{number of source code lines}}$$

We will scale this ration based on class size.

The mark you will receive for the testing coverage task is determined using Table 1:

Coverage ratio range	% of testing mark
0%-5%	0%
5%-20%	20%
20%-40%	40%
40%-60%	60%
60%-80%	80%
80%-100%	100%

Table 1: Mark assignment for testing

Note the top boundary for the Coverage ratio range is not inclusive except for 100%. Also, note that only the functions stipulated in this specification will be considered to determine your testing mark. Remember that your main will be testing the Instructor Provided code and as such, it can only be assumed that the functions outlined in this specification are defined and implemented.

As you will be receiving marks for your testing main, we will also be doing plagiarism checks on your testing main.

6 Implementation Details

- You must implement the functions in the header files exactly as stipulated in this specification. Failure to do so will result in compilation errors on FitchFork.
- You may only use **c++98**.
- You may only utilize the specified libraries. Failure to do so will result in compilation errors on FitchFork.
- Do not include using `namespace std` in any of the files.
- You may only use the following libraries:
  - string
  - fstream
  - sstream
  - iostream

- You are supplied with a **trivial** main demonstrating the basic functionality of this assessment.

7 Upload Checklist

The following c++ files should be in a zip archive named uXXXXXXXX.zip where XXXXXXXX is your student number:

- item.h
- item.cpp
- inventory.h
- inventory.cpp
- `main.cpp`
- Any textfiles used by your `main.cpp`
- `testingFramework.h` and `testingFramework.cpp` if you used these files.

The files should be in the root directory of your zip file. In other words, when you open your zip file you should immediately see your files. They should not be inside another folder.

8 Submission

You need to submit your source files on the FitchFork website (`https://ff.cs.up.ac.za/`). All methods need to be implemented (or at least stubbed) before submission. Your code should be able to be compiled with the following command:

```
g++ *.cpp -o main
```

and run with the following command:

```
./main
```

Remember your `h` file will be overwritten, so ensure you do not alter the provided `h` files.
You have 10 submissions and your best mark will be your final mark. Upload your archive to the Practical 2 slot on the FitchFork website. Submit your work before the deadline. **No late submissions will be accepted!**

9 Accessibility

1 Show the class diagram of the two classes to be implemented in this practical:

- item

- inventory

The member variables and functions are discussed in Section 3. The following relationships exists between the item and inventory class:

- The inventory class has a relationship with the item class because of the items and shelves member variables.