COMP2012

Object-Oriented Programming and Data Structures

INTRODUCTION

In the first assignment, you will model a shopping mall in C++ with OOP. You are going to have a good practice on C++ OOP basics and refresh on your C++ knowledge you have acquired in the fundamental C++ course.

HONOR CODE

We value academic integrity very highly. Please read the Honor Code section on our course webpage to make sure you understand what is considered as plagiarism and what the penalties are. The following are some of the highlights:

- Do NOT try your "luck" - we use some sophisticated plagiarism detection software to find cheaters. It is much better than most students think. It has been proven times and times again tricks didn't work. We also review codes for potential cases manually.
- The penalty (for **BOTH** the copier and the copiee) is not just getting a zero in your assignment - it is much more than that. It is simply not worth to cheat at all. You would hurt your friend and yourself by doing that. It is obvious that a real friend won't ask you to get involved in a plagiarism act in any way due to the consequences. Read the Honor Code again before you even try to think about cheating.
- Serious offenders will fail the course immediately and there may be additional disciplinary actions from the department and university.

INSTRUCTIONS

The overview of the shopping mall model is as follows:

- A **Mall** has a collection of **Shop**s. (stored in a linked list which uses the **Node** class)
- A **Shop** has a collection of **Product**s. (stored in a dynamic array)

Your solution to this assignment must be based on the given skeleton code provided in the *Download* page.

Your task is to complete the following:

- Mall implementation (except for the *Mall::duplicateShopLinkedList* function) in mall.cpp
- *Mall::duplicateShopLinkedList*'s recursive implementation in noloop.cpp
- Shop implementation in shop.cpp
- Product implementation in product.cpp
- Node implementation in node.cpp

These are the only 5 files you are supposed to modify, zip, and then submit to Canvas.

Comments in the corresponding header file lists out the detailed requirements. The description below supplements the only missing information. You need to read both the source/header files and the webpage description carefully to get the whole picture.

While you probably will jump back and forth as you read the materials as they reference each others, we recommend the following general reading order:

1. mall.h
2. node.h
3. shop.h
4. product.h
5. Remaining of this webpage
6. main.cpp

Read the *FAQ* page for some common clarifications. You should check that a day before the deadline to make sure you don't miss any clarification, even if you have already submitted your work then.

Submission details are in the *Submission and Deadline* page.

PRODUCT ORDERING IN A SHOP

The Product objects are stored in a dynamic array in a Shop object. They should be stored in ascending order by their prices in the array. If two products have the same price, they will be ordered lexicographically by their names. Comparing names can be done easily by basic string comparison - since a product name is of the string class, the names can be directly compared with operator ">", "<", or "==". For example:

```
string a = "Apple";
string b = "Banana";
cout << boolalpha << (b>a) << endl; //prints true, as b is lexicographically larger
```

In the following examples, we denote a Product object as a tuple (NAME_OF_THE_PRODUCT, PRICE).

- ("Apple", 20f) is larger than ("Banana", 10f) due to its higher price, and hence "Apple" should appear after "Banana" in the array.
- ("Apple", 30f) is smaller than ("Banana", 30f) due to its lexicographically smaller name, and hence "Banana" should appear after "Apple" in the array.

Note that no two product objects will have the same name in a shop's product array.

As prices are floating point numbers, you should be careful when comparing them as they are not precise by nature. In general, for equality comparison of floats, we would allow for a small margin of error. So instead of:

```
if( priceA == priceB ) ... //comparing floats priceA and priceB directly with == operator
```

You should do:

```
if( fabs(priceA - priceB) < 0.01f ) ... //for this application, 0.01f is a good margin
```

where fabs is a cmath library (already included in product.h) function that returns the absolute value of the given floating point number.

ADDING/REMOVING SOME AMOUNT OF PRODUCTS IN A SHOP

In the following examples, we denote a Product object as a tuple (NAME_OF_THE_PRODUCT, TYPE, PRICE, QUANTITY).

- Assume that at the beginning the array has the following content in the exact order:
  ("Banana", FOOD, 10f, 500), ("Apple", FOOD, 20f, 400)

- If we invoke addProduct("Orange", FOOD, 10, 300), then the array will have:
  ("Banana", FOOD, 10f, 500), ("Orange", FOOD, 10f, 300), ("Apple", FOOD, 20f, 400)

- If we invoke addProduct("Apple", FOOD, 20f, 600), then the array will have:
  ("Banana", FOOD, 10f, 500), ("Orange", FOOD, 10f, 300), ("Apple", FOOD, 20f, 1000)
  Notice that we do not add a new item to the array but rather to update the quantity of the existing "Apple" product in the array. Also, as we are updating the quantity of an existing item, you can assume that the addProduct call will always be given the correct price (i.e. 20f for apple) and type (i.e FOOD for apple) in the parameters. That means, for simplicity, you can assume there will never be any conflict with the existing data.

- If we invoke removeProduct("Apple", 50), then the array will have:
  ("Banana", FOOD, 10f, 500), ("Orange", FOOD, 10f, 300), ("Apple", FOOD, 20f, 950)

- If we invoke removeProduct("Apple", 950), then the array will have:
  ("Banana", FOOD, 10f, 500), ("Orange", FOOD, 10f, 300)
  Notice that the product "Apple" has been removed from the array as its quantity reaches 0. Note also that the function removeProduct, unlike addProduct, may do nothing and return false when certain conditions hold, please refer to the shop.h for details.

ADDITIONAL NOTES

- You should not submit or modify any file except mall.cpp, shop.cpp, product.cpp, node.cpp, and noloop.cpp. When we do the grading, we will provide our header files and main.cpp. Therefore, it is very important for you to make sure that your program can compile with those files in their original unmodified states.
- Only exception to the above rule: you may edit main.cpp temporarily for creating and testing with your own test cases, but that file should not be submitted. Your submitted code must still be able to be compiled with the original main.cpp.
- You are NOT allowed to include any additional library.
- *Mall::duplicateShopLinkedList* must be implemented as a recursion in the noloop.cpp file. No loop of any kind is allowed. Before you submit your work, do a search (ctrl+f in Windows Eclipse) on noloop.cpp and make sure the keywords "for", "while", and "goto" do not exist in the noloop.cpp file. Those keywords should not appear even in the comments. If we find any of those words in your noloop.cpp file or that you are using any kind of loop, 0 mark will be given for that function.
- For C++ string class usage, you may refer to our COMP2011 string lecture note here.

SAMPLE OUTPUT

Your finished program should produce the exact same output as below. Please note that sample outputs, naturally, do not show all possible cases. It is part of the assessment for you to design your own test cases to test your program. Be reminded to remove any debugging message that you might have added before submitting your code.

```
Test case 1:
Mall: HKUST Mall @ Clear Water Bay
Shops in HKUST Mall: nullptr
Mall: HKUST Super Mall @ Clearer Water Bay
Shops in HKUST Super Mall: (1,"Supermarket") -> nullptr
Shop: #1 Supermarket
printShop: No such shop.

Test case 2:
Shops in HKUST Mall: (1,"Supermarket A") -> (2,"Supermarket B") -> (3,"Supermarket C") -> nullptr
Remove result: true
Shops in HKUST Mall: (2,"Supermarket B") -> (3,"Supermarket C") -> nullptr
Shops in HKUST Original Mall: (1,"Supermarket A") -> (2,"Supermarket B") -> (3,"Supermarket C") -> nullptr

Test case 3:
Shops in HKUST Mall: (1,"Supermarket A") -> (2,"Supermarket B") -> (3,"Supermarket C") -> nullptr
Remove result: true
Shops in HKUST Mall: (1,"Supermarket A") -> (3,"Supermarket C") -> nullptr

Test case 4:
Shops in HKUST Mall: (1,"Supermarket A") -> (2,"Supermarket B") -> (3,"Supermarket C") -> nullptr
Remove result: true
Shops in HKUST Mall: (1,"Supermarket A") -> (2,"Supermarket B") -> nullptr

Test case 5:
Shops in HKUST Mall: (1,"Supermarket A") -> (2,"Supermarket B") -> (3,"Supermarket C") -> nullptr
Remove result: false
Shops in HKUST Mall: (1,"Supermarket A") -> (2,"Supermarket B") -> (3,"Supermarket C") -> nullptr

Test case 6:
Add result: true
Add result: false
Shops in HKUST Mall: (1,"Supermarket A") -> nullptr
```

```
Test case 7:
Products in Supermarket: none.
Products in Supermarket: ("XXBOX",3,2000,10)
Products in Supermarket: ("XXBOX",3,2000,20)
Products in Supermarket: ("XXBOX",3,2000,30)

Test case 8:
Products in Supermarket: none.
Products in Supermarket: ("XXBOX",3,2000,10)
Products in Supermarket: ("XXBOX",3,2000,10) ("YBOX",3,3000,40)
Products in Supermarket: ("ZZZBOX",3,1000,20) ("XXBOX",3,2000,10) ("YBOX",3,3000,40)
Products in Supermarket: ("Apple",0,5,200) ("ZZZBOX",3,1000,20) ("XXBOX",3,2000,10) ("YBOX",3,3000,40)
Products in Supermarket: ("Apple",0,5,200) ("ZZZBOX",3,1000,20) ("XXBOX",3,2000,10) ("PreyStation",3,2500,37) ("YBOX",3,3000,40

Test case 9:
Products in Supermarket: ("Apple",0,5,200) ("ZZZBOX",3,1000,20) ("XXBOX",3,2000,10) ("PreyStation",3,2500,37) ("YBOX",3,3000,40
Remove result: false
Products in Supermarket: ("Apple",0,5,200) ("ZZZBOX",3,1000,20) ("XXBOX",3,2000,10) ("PreyStation",3,2500,37) ("YBOX",3,3000,40
Remove result: true
Products in Supermarket: ("Apple",0,5,100) ("ZZZBOX",3,1000,20) ("XXBOX",3,2000,10) ("PreyStation",3,2500,37) ("YBOX",3,3000,40
Remove result: false
Products in Supermarket: ("Apple",0,5,100) ("ZZZBOX",3,1000,20) ("XXBOX",3,2000,10) ("PreyStation",3,2500,37) ("YBOX",3,3000,40
Remove result: true
Products in Supermarket: ("ZZZBOX",3,1000,20) ("XXBOX",3,2000,10) ("PreyStation",3,2500,37) ("YBOX",3,3000,40)
Remove result: true
Products in Supermarket: ("ZZZBOX",3,1000,20) ("PreyStation",3,2500,37) ("YBOX",3,3000,40)
Remove result: true
Products in Supermarket: ("ZZZBOX",3,1000,20) ("PreyStation",3,2500,37)

Test case 10:
Products in Supermarket: ("Apple",0,5,200) ("ZZZBOX",3,1000,20) ("XXBOX",3,2000,10) ("PreyStation",3,2500,37) ("YBOX",3,3000,40
Update result: true
Products in Supermarket: ("Apple",0,5,200) ("ZZZBOX",3,1000,20) ("PreyStation",3,2500,37) ("YBOX",3,3000,40) ("XXBOX",3,9000,10
Products in Supermarket Clone: ("Apple",0,5,200) ("ZZZBOX",3,1000,20) ("XXBOX",3,2000,10) ("PreyStation",3,2500,37) ("YBOX",3,3
```

(all program executions should terminate without any runtime error and memory leak.)

DOWNLOAD

- Skeleton code: skeleton.zip

Create a standard Eclipse C++ project (MinGW gcc compiler on Windows) and add all files from the zip package to it.

It is required that your program can be compiled and run successfully in the pre-installed Eclipse environment on our lab machines. If you use other IDE/compiler/OS to work out your solution, you **must** test your program in the aforementioned official environment before submission.

Using our Windows Eclipse zipped package downloaded from the "Using Eclipse at home (Windows) " section here on a standard Windows machine is also good enough to verify that your program can be compiled by us. HKUST virtual barn may be used if you have no access to any other Windows machines. Notice that the Desktop there has limited space, you may use C:\temp there to unzip the Eclipse package, but don't leave your source code there in C:\temp.

DEADLINE

23:59:00 on Mar 10th, 2018.

CANVAS SUBMISSION

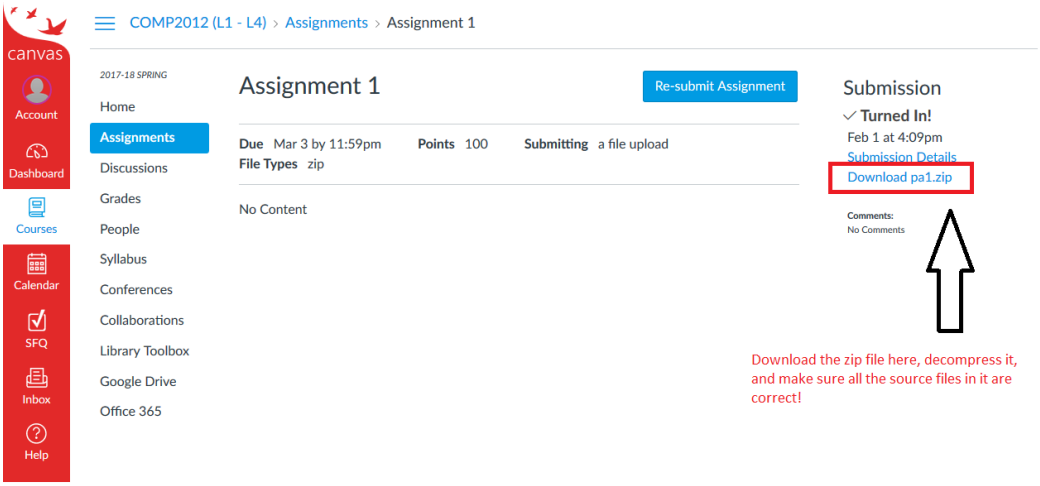
We will show you how to create a zip file and submit it to canvas in lab 1.

Create a single zip file that contains all the **5 cpp files** mall.cpp, shop.cpp, product.cpp, node.cpp, and noloop.cpp. The zip file should be named as **pa1.zip**. And then you should submit only the **pa1.zip** through the **Canvas Assignment 1 Submission Page. The filenames have to be exactly the same**. Must be a zip file, not rar, not 7z, not tar, not gz, etc. Inside the zip file, the 5 cpp files also need to be of the correct names.

Make sure your source file can be successfully compiled. It is required that your program can be compiled and run successfully in the pre-installed Eclipse environment on our lab machines. If we cannot even compile your source file, your work will not be graded. Therefore, you should at least put in dummy implementations to the parts that you cannot finish so that there will be no compilation error.

Make sure you actually upload the correct version of your source files - we only grade what you upload. Some students in the past submitted an empty file or a wrong file or an exe file which is worth zero mark. So **you must download and double-check the file you have submitted**. We

will show you how in lab 1, and you may refer to the illustration below.



You may submit your file multiple times, but only the latest version will be graded.

Submit early to avoid any last-minute problem. Only canvas submissions will be accepted.

Note 1: If you have no idea how to create a zip file, you may see How to create a zip file in Windows 10 or How to create a zip file in Mac OS X . Try to zip just the 5 cpp files, not a folder containing the 5 files.

Note 2: Canvas may append a number to the filename of the file you have submitted. e.g. pa1-1.zip. It is OK as long as you have named your file as pa1.zip when you submit it.

LATE SUBMISSION POLICY

There will be a penalty of -1 point (out of a maximum 100 points) for every minute you are late. For instance, since the deadline of assignment 3 is 23:59:00 on Mar 10th, if you submit your solution at 1:00:00 on Mar 11th , there will be a penalty of -61 points for your assignment. However, the lowest grade you may get from an assignment is zero: any negative score after the deduction due to late penalty (and any other penalties) will be reset to zero.

FREQUENTLY ASKED QUESTIONS

Q: Will two different products have the same name?

A: No. Products can be identified solely by their names.

Q: Can the same product have two different prices? When we do addProduct, will a new/different price be provided for an existing product?

A: No and no. Won't be tested. As written in the addProduct comment description: "for simplicity, you can assume the given parameters are all valid ... type and price are always consistent with those of an existing item if any". Therefore, in your products dynamic array, a product should only appear once.

Q: Are the shops sorted?

A: As described in addShop: "add a shop with the given name and shop number to the end of the shop linked list", so you always add a new shop to the end of the linked list. The shops are not ordered in any other way. Don't mind the shop numbers in terms of ordering consideration.