Project 4, The return of store manager and order filler
Due: Please check the due date on Blackboard

Objectives

The primary purpose of this project is to introduce interval heaps and Google Test. This project assumes that you have successfully completed project 2 and extends it. The project intends to give you experience with extending code (at the complexity level of a CMSC 341 project) and automated code testing.

Introduction

After opening a new store just a few weeks ago, the store manager recieved an overwhelming positive response. However, this also meant a lot more accounting. The store manager also noticed that recent orders of items cater to two extremes: all the ordered items being cheapest or all of them being most expensive in their category. The store manager has decided to hire a new accountant to help with this.

For each food category, the accountant keeps track of the range of prices of food items available. To keep things more orderly, and because the accountant is a computer science minor, the accountant decides to store these prices in an interval heap. That way, the accountant has direct information price range stucture for that category of food. The accountant also creates an additional attribute to each order, specifying whether the food items should be cheapest or most expensive in their category. This attribute is then used while selecting food items for the order. The accountant is also in charge of reporting the total sales for the day.

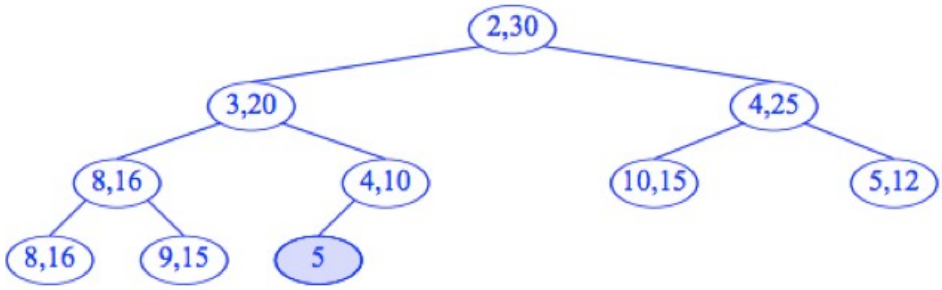
On Interval Heaps

An *interval heap* is a *double-ended priority queue* that permits us to delete from either end, either the min end or the max end. For this project, priority would correspond to the price of an item. An interval heap permits us to insert and delete elements in O(*log n*) time, where *n* is the number of elements in the double-ended priority queue. Here is the definition of interval heap and an example featuring a 19-element interval heap.

Definition: An interval heap is a complete binary tree that meets the following conditions.

- When the total number of element *n* is even, each node of this complete binary tree has two elements *a* and *b*, *a*<=*b*. We say that the node represents the interval [*a*, *b*].
- When *n* is odd, each node other than the last one has two elements *a* and *b*, *a*<=*b*. The last node has the single element *a*. The interval represented by the last node is [*a*, *a*].
- Let [a_c, b_c] be the interval represented by any non-root node in the interval heap. Let [a_p, b_p] be the interval represented by the parent of this node, then $a_p <= a_c <= b_c <= b_p$.

Let’s look at an example of a 19-element interval heap. Here we only need 10 nodes to represent the interval heap.



As you may see, the left-ends of the interval define a min-heap and the right-ends defines a max-heap.

Reference:
Sartaj, Sahni. "Data Structures, Algorithms and Applications in C++." *Computer Science, Singapore: McGraw-Hill* (1998).

On Google Test

Google Test (gtest) is a unit-testing framework for C++ programs. Unit tests are used to test individual units of source code, in this case, functions.

Implementation: Additional files for interval heap

To begin with, the project adds two new classes named Node and IntervalHeap in the form of four files (Node.cpp, Node.h, IntervalHeap.cpp and IntervalHeap.h).

The Node class implements a self refrential class (it contains a data member of type class pointer). Links are maintained between child and parent nodes. The Node class does not provide functionality on its own: it is used by the IntervalHeap class.

Each node in the interval heap is corresponds to two food items. The price range reprenented by that node is [price of left food item, price of right food item]. The prices of left food items of nodes in the interval heap correspond to a min heap. Similarly, the prices of right food items of nodes in the interval heap correspond to a max heap.

There are three methods that you will need to implement for the IntervalHeap class: insert, deleteMin and deleteMax.

+ IntervalHeap::insert (Food*): void

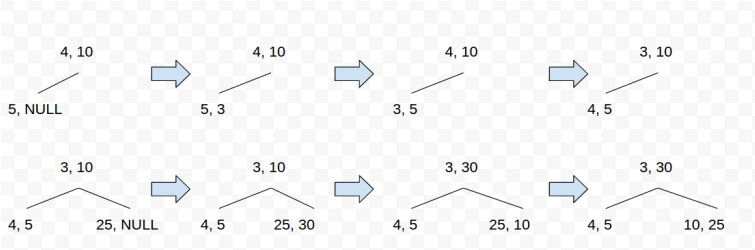
Note that food items corresponding to a node are initialized as NULL. A new food item could then be inserted into the interval heap in the following manner.

- Insert the new food item at the end of the interval heap (the next NULL location corresponding to a node) as would be done in the case of a min or max heap (in complete tree order). This may be as the left food item of a new node, or the right food item of an existing node.
- Ensure that the node individually satisfies the min heap and max heap properties. That is, the price of the left food item is less than the price of the right food item (in case of two food

items). In case of 1 food item in a node, the food item must be compared to both its parents using corresponding heap properties. This is because a node with one food item represents a range of *cost-cost*, where cost is the cost of that food item.

- Ensure that all the left food items of nodes in the interval heap consistently form a min heap.
- Ensure that all the right food items of nodes in the interval heap consistently form a max heap.

Example: (insert)



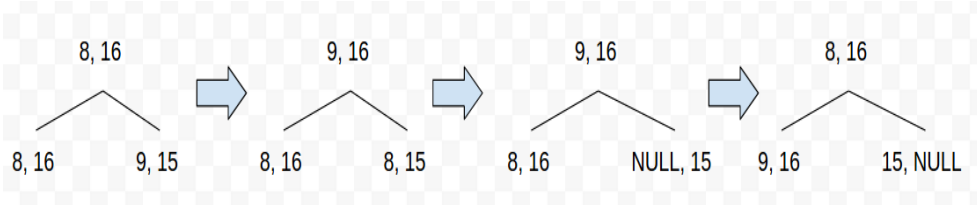
+ IntervalHeap::deleteMin (void): Food*
+ IntervalHeap::deleteMax (void): Food*

Delete operations are the same as that of a min or max heap. After deletion, an additional check must be performed to ensure that the involved nodes individually satisfy the min heap and max heap properties. While the delete operations remove the food item from the interval heap, they do not destroy its contents (delete the food item). This food item is then returned by the function. If a node's left and right elements both become NULL, delete that node.

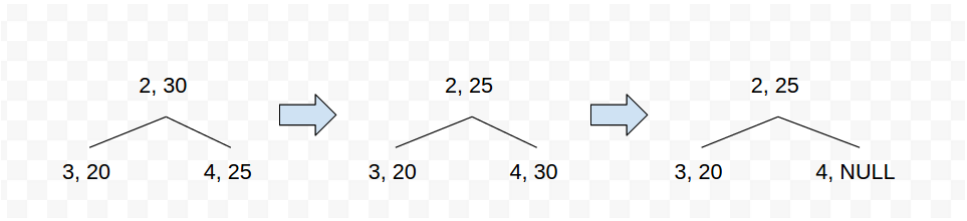
The food item returned by this function is then to be provided to addFoodToOrder as the input argument.

Note: In case of a tie while shifting an element down from the root, the left child node is given priority over the right child node.

Example: (deleteMin)



Example: (deleteMax)



Implementation: Additions to the Store class

+ Store::printHeaps(void): void

This function displays contents of all interval heaps based on level-wise node traversal. The values for each heap (corresponding to each food type) are displayed in sequence. Each heap is displayed in a single line. Refer to the output section of this document for more detail.

Implementation: Additions to the Order class

+ Order::ORDER_TYPE: enum
- Order::m_eType: ORDER_TYPE

Whether we use deleteMin for all interval heaps while completing this order or whether we use deleteMax for all interval heaps while completing this order is decided by the value of m_eType. This is reflected by an additional data item as “min” or “max” (corresponding to deleteMin and deleteMax over all heaps respectively) after the “yes” or “no” (partial order acceptance information) in each order.

Implementaion: Additions to the Manager class

- Manager::m_totalSales: double
+ Manager::getTotalSales (void): double

The total sales made by the store are recorded in m_totalSales and retrieved using getTotalSales( ). You do not need to take care of precision. Just add the prices as they are. Doing otherwise will cause automated tests to fail since we would be comparing against the wrong number. The initial value of m_totalSales is 0.

Implementation: Changes in Driver

In order to integrate with Google Test, Driver.cpp must not contain main( ). It is therefore required to rename main( ) to runDriver( ). Additionally, runDriver( ) now returns the value total sales using Manager::getTotalSales( ) and its return type is changed to double. A corresponding Driver.h will be required to be used by Google Test.

Implementation: Integrating with Google Test

Copy the factorial.cpp, factorial.h, project4_test.cpp and Makefile_test files into your working directory (available for download as a .zip file). The test code is currently set up to test a factorial function (implementation contained). To build and run the tests, use make with the specified makefile (using -f) as follows.

make -f Makefile_test

Don't worry about any warnings that may show up. If the build is successful, an executable file named project4_test will be created in your working directory. Run project4_test (./project4_test) and you should see output similar to the following.

```
Terminal
linux1[9]% make -f Makefile_test
g++ -isystem /afs/umbc.edu/users/c/m/cmarron/pub/gtest-1.7.0/include -I/afs/umbc.edu/users/c/m/cmarron/pub/gtest-1.7.0 -g -Wall -Wextra -pthread -c \
            /afs/umbc.edu/users/c/m/cmarron/pub/gtest-1.7.0/src/gtest-all.cc
In file included from /afs/umbc.edu/users/c/m/cmarron/pub/gtest-1.7.0/src/gtest-all.cc:42:
/afs/umbc.edu/users/c/m/cmarron/pub/gtest-1.7.0/src/gtest.cc:370: warning: missing initializer for member 'testing::internal::MutexBase::owner_'
g++ -isystem /afs/umbc.edu/users/c/m/cmarron/pub/gtest-1.7.0/include -I/afs/umbc.edu/users/c/m/cmarron/pub/gtest-1.7.0 -g -Wall -Wextra -pthread -c \
           /afs/umbc.edu/users/c/m/cmarron/pub/gtest-1.7.0/src/gtest_main.cc
ar rv gtest_main.a gtest-all.o gtest_main.o
ar: creating gtest_main.a
a - gtest-all.o
a - gtest_main.o
g++ -isystem /afs/umbc.edu/users/c/m/cmarron/pub/gtest-1.7.0/include -g -Wall -Wextra -pthread -c ./project4_test.cpp
g++ -isystem /afs/umbc.edu/users/c/m/cmarron/pub/gtest-1.7.0/include -g -Wall -Wextra -pthread -c ./factorial.cpp
g++ -isystem /afs/umbc.edu/users/c/m/cmarron/pub/gtest-1.7.0/include -g -Wall -Wextra -pthread -lpthread gtest_main.a project4_test.o factorial.o -o project4_test
linux1[10]% ls
factorial.cpp  factorial.h  factorial.o  gtest-all.o  gtest_main.a  gtest_main.o  Makefile_test  project4_test  project4_test.cpp  project4_test.o
linux1[11]% ./project4_test
Running main() from gtest_main.cc
[==========] Running 2 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 2 tests from FactorialTest
[ RUN      ] FactorialTest.HandlesValidInput
[       OK ] FactorialTest.HandlesValidInput (0 ms)
[ RUN      ] FactorialTest.HandlesInvalidInput
[       OK ] FactorialTest.HandlesInvalidInput (0 ms)
[----------] 2 tests from FactorialTest (0 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test case ran. (0 ms total)
[  PASSED  ] 2 tests.
linux1[12]%
linux1[12]%
```

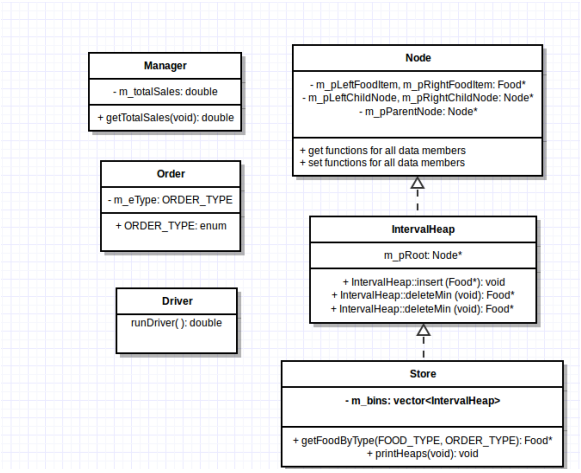
Details of the internals of these files are as explained in the references and are not repeated here for the sake of brevity.

Your mission, should you choose to accept it, involves editing project4_test.cpp to test your code. This can be done by providing Driver::runDriver( ) input arguments in a manner similar to what is done for Factorial (example in given code). The output of Driver::runDriver( ) can then be compared with a constant value. Note that any trailing zeros in the constant value or return value will not cause erraneous comparison.

References:
https://code.google.com/p/googletest/
http://www.csee.umbc.edu/courses/undergraduate/202/fall15_marron/labs/lab03/page2.shtml

Implementation: UML

The changes made in the several classes are summarized in the following UML. Edits to existing data memebers and memeber functions are in bold font. Please refer to the link below for an overall picture. Note that this is a template. You may add more data members and memeber functions if you think needed.



Reference:
http://userpages.umbc.edu/~slupoli/notes/DataStructures/projects/F15Proj2QueuesStoreManager/StoreManager.pdf

Implementation: Base Code (Project 2)

Good news! An implementation of project 2 is available to start with, for students who would like to use it. It is located at: “Blackboard->Resources->Source->Project 2 Source Code”

Input

The input requirements are similar to those of project 2. Note that for simplicity, acceptance of partial order is always “yes”. An additional “min” or “max” paramter follows the partial order parameter, indicating the use of deleteMin or deleteMax respectively over all interval heaps.

Reference:
http://userpages.umbc.edu/~slupoli/notes/DataStructures/projects/F15Proj2QueuesStoreManager/StoreManager.pdf

---

Output

The output requirements for results.txt are similar to those of project 2. However, the store is no longer printed in the format specified by project 2. Instead, the states of the interval heaps (in level order) for each food type must be written to results.txt after each order is written to results.txt.

Format for each food type:
<food type>: <cost of left food item> <cost of right food item> ... <print based on level order traversal of nodes>

Note that as in project 2, costs are printed to 2 decimal places. In case of an empty food item in a node, print NULL. In case there are no nodes in the interval heap, print NULL.

Store::printHeaps( )

Example: (content added to results.txt by this function, after 1 deleteMax in meat and 1 deleteMin in fruit in two subsequent orders)

Meat: 2.20 30.65 3.16 20.15 4.65 25.25
Fruit: 8.94 16.99 8.99 16.50 9.97 15.36
Vegetable: 4.10 10.02 5.01 NULL
Starch: 4.58 25.65 10.45 15.79
Sweet: NULL

Meat: 2.20 25.25 3.16 20.15 4.65 NULL
Fruit: 8.94 16.99 8.99 16.50 9.97 15.36
Vegetable: 4.10 10.02 5.01 NULL
Starch: 4.58 25.65 10.45 15.79
Sweet: NULL

Meat: 2.20 25.25 3.16 20.15 4.65 NULL
Fruit: 8.99 16.99 9.97 16.50 15.36 NULL
Vegetable: 4.10 10.02 5.01 NULL
Starch: 4.58 25.65 10.45 15.79
Sweet: NULL

...

Since we will be executing the driver via project4_test and not directly (as was done for project 4), an additional visual output will be provided by Google Test.

Reference:
http://userpages.umbc.edu/~slupoli/notes/DataStructures/projects/F15Proj2QueuesStoreManager/StoreManager.pdf

---

Running and Compiling Requirements

Make sure your code runs and compiles on GL. If you have coded it in another environment e.g. Visual Studio, make sure that it has been ported to GL.

What to Submit

Follow the course project submission procedures. You should copy over all of your C++ source code with .cpp/.h files and makefiles under the src directory.

Make sure that your code is in the ~/cs341proj/proj4/ directory and not in a subdirectory of ~/cs341proj/proj4/. In particular, the following Unix commands should work.

```
cd ~/cs341proj/proj4/src
make -f Makefile_test
./project4_test
```

Don't forget the Project Submission requirements shown online! One hint, **after you submit**, if you type the following

```
ls ~/cs341proj/proj4/
```

and you see a bunch of .cpp and .h files, this is WRONG. You should see:

```
src
```

instead. The C++ programs must be in directories under the src directory. Your submissions will be compiled by a script. The script will go to your proj1 directory and run your makefile. This is required. You will be severely penalized if you do not follow the submission instructions.
