Q1 [60 marks] Iterator Design Pattern

You are to use the Iterator Design Pattern to implement iterators for a provided composite class structure of Menu, submenus, and MenuItem that are designed according to the Composite Design Pattern.

Note: Your code will differ from the Iterator Design Pattern code provided in class\(^1\) in two fundamental ways:

1. your iterators will be **ForwardIterators**, whereas the code in class used an **InputIterator**, and
2. your iterators will support methods that the STL algorithms expect. Specifically:
   - Each class in the Composite class structure must have a method **begin()** for creating an iterator to the first element in a composite object.
   - Each class in the Composite class structure must have a method **end()** for creating an end-of-iteration iterator (that can be used to check that the value of an active iterator has finished iterating through a composite object).
   - The iterator for each class in the Composite class structure must be defined as a nested class.
   - Your iterators must overload the following operators so that the iterators can be used in STL algorithms, necessary for Q2:
     - **operator==** (to be able to compare iterator values)
     - **operator!=** (to be able to compare iterator values)
     - **operator*** (to be able to dereference an iterator, to retrieve a MenuComponent*)
     - **operator->** (to be able to dereference an iterator, to retrieve a MenuComponent*)
     - **operator++** (prefix - to be able to advance the iterator)
     - **operator++** (postfix - to be able to advance the iterator)

---

\(^1\) See lecture notes [https://www.student.cs.uwaterloo.ca/~cs247/current/Lectures/15Iterator-1up.pdf](https://www.student.cs.uwaterloo.ca/~cs247/current/Lectures/15Iterator-1up.pdf), slides 12-21, especially slides 18-21.
Objective
The objective is to practice implementing the Iterator design pattern, which will require you to also understand the Composite design pattern. You will also get some practice understanding the structure of the STL iterators.

Provided files
1. The Composite classes header and implementation files (MenuComponent, Menu, MenuItem) are provided, but lack the iterator implementation. You will have to fill in that part.
2. Object (.o) files for the Composite classes and the ComponentIterator² class, fully implemented, are provided so that you link them together to create an executable. As you complete your code, you can remove the associated .o file and replace it with your own.
3. The ComponentIterator header has been provided. You will have to fill in the private declarations, and implement the code in an associated .cc file. You may add other, private or protected members if you need them; however, you may not change the interface since the rest of the code relies upon it.
4. An initial Makefile that creates the executable menu. You can use it, or modify it, as you wish. Do not, however, change the name of the executable since our Marmoset tests will rely upon it.
5. We also provide a main program for your solution, IteratorTestHarness.cc. The main program is a test harness that you can use to test your implementation by creating various menus and using your iterators to print them. The test harness is not robust (it is throwaway code). If you enter invalid commands, it might cause the program to terminate. Your implementation will be tested using our test harness, so do not make any changes to this code.
6. We have also provided you with a sample executable, sampleIterator, that you can use to verify that your output matches ours.

Execution
There are 6 valid commands that the test harness will recognize. The provided code implements all of the commands except the print command. The print command needs your iterator. If you comment out the print command, the program will compile and can be used to build composite menus.

Your program will be tested only on valid input commands.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m &lt;#&gt; &quot;&lt;name&gt;&quot;</td>
<td>Creates a new menu at index &lt;#&gt; named &lt;name&gt;³.</td>
</tr>
<tr>
<td>i &lt;#&gt; &quot;&lt;name&gt;&quot; &lt;price&gt; &lt;calories&gt; &lt;bool&gt;</td>
<td>Creates a new menu item named name, stored at index #, with the specified price and calorie count, calories. The item is vegetarian if the boolean value is true. name is case-sensitive and you may assume it is unique.</td>
</tr>
<tr>
<td>a &lt;#1&gt; &lt;#2&gt;</td>
<td>Add the menu (or submenu) #2 as a child of menu #1. Menu #2 is set to a nullptr.</td>
</tr>
<tr>
<td>r &lt;#&gt; &quot;&lt;name&gt;&quot;</td>
<td>Remove the menu item or submenu name from menu #1. The search is not recursive, since that would require your iterator to be implemented. name is case-sensitive and you may assume it is unique.</td>
</tr>
<tr>
<td>p &lt;#&gt;</td>
<td>Prints all elements and descendants of the menu stored at index #. For every level in the composite tree that the iterator descends, a tab character is added to the prefix for the item</td>
</tr>
</tbody>
</table>

² ComponentIterator is the abstract base class for the Iterator design pattern's hierarchy.
³ The name is surrounded by double-quotiation marks to allow whitespace in the name, though the test harness removes them from the string.
being printed. Note that the test harness already prints each item on a separate line.

Each Menu/MenuItem has its name printed, followed by the information from its components, where applicable. A Menu, when printed, starts with a newline, the appropriate number of tabs, its name, a colon (":"), and then its contents.

See the sample output and the executable for details on the format for the MenuItem. Note that all prices are formatted to two decimal places using the stream formatting operations, not via rounding.

<table>
<thead>
<tr>
<th>d</th>
<th>Terminates the menu creation phase in the program.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl-d</td>
<td>Terminates the program.</td>
</tr>
</tbody>
</table>

**Hints**

1. If you want to use your iterator in question 2 rather than the provided iterator, you will need to implement a non-polymorphic iterator that works when "iterating" over a MenuItem as well as when iterating over the elements of Menu. You will still need to implement a stack-based iterator, but it will need to work for all types of MenuComponent classes. Then your nested iterator classes simply inherit from this non-polymorphic iterator. (See the Factory Method design pattern code for an example of nested classes that inherit from another class.)

2. Also, your non-polymorphic iterator will need to inherit from a base iterator class in the standard library in order to support the type definitions required by the STL. The std base iterator classes are in <iterator>. For example, the definition of my iterator looks like the following:

   ```cpp
class ComponentIterator:
   std::iterator< std::forward_iterator_tag, MenuComponent*> {...};
   My Menu::iterator and MenuItem::iterator inherit from this class.
   ```

3. See Eckel volume 1, chapter 12 for advice on overloading operators, in particular operator->.

**Sample Execution**

Below is a sample partial execution. User input is shown in **bold** font. Note that the newline and tab characters output are shown as \t and \n respectively in the output, that is you're not printing an actual string consisting of the backslash followed by a t or n.

---

Test harness for Menu Composite and Iterator

Command: m 1 "vegetables"
Command: i 2 "carrots" 1.5 10 true
Command: i 3 "cabbage" 0.99 20 true
Command: i 4 "parsley" 0.10 1 true
Command: a 1 2
Command: a 1 3
Command: a 1 4
Command: m 2 "meats"
Command: i 3 "eggs" 2 100 false
Command: i 4 "steak" 15.87 350 false
Command: i 5 "fish" 8.67 120 false
Command: a 2 3
Command: a 2 4

---

\(^4\) In Q1, the program ends. In Q2, this allows you to test the extra code you are inserting.
Command: a 2 5
Command: m 3 "pseudo-meats"
Command: i 4 "tofu" 3.54 200 true
Command: i 5 "tofurkey" 6.22 200 true
Command: i 6 "pseudo ham" 5.431 230 true
Command: a 3 4
Command: a 3 5
Command: a 3 6
Command: a 2 3
Command: m 0 "WatWat Foods Emporium"
Command: a 0 2
Command: a 0 1
Command: p 0

WatWat Foods Emporium:
meats:

teggs, $2.00, cal=100

tsteak, $15.87, cal=350

tfish, $8.67, cal=120

tpseudo-meats:
ttofu, $3.54, cal=200 (V)
ttofurkey, $6.22, cal=200 (V)
tpseudo ham, $5.43, cal=230 (V)

vegetables:
tcarrots, $1.50, cal=10 (V)
tcabbage, $0.99, cal=20 (V)
tparsley, $0.10, cal=1 (V)
Command: r 0 "pseudo ham"
Command: r 0 "vegetables"
Command: p 0

WatWat Foods Emporium:
meats:
teggs, $2.00, cal=100

tsteak, $15.87, cal=350

tfish, $8.67, cal=120

tpseudo-meats:
ttofu, $3.54, cal=200 (V)
ttofurkey, $6.22, cal=200 (V)
tpseudo ham, $5.43, cal=230 (V)
Command: d