Coding Assignment 4

1. Create Code4_xxxxxxxxxx.cpp

Copy your working version of Code3_xxxxxxxxxx.cpp to Code4_xxxxxxxxxx.cpp.

2. Create CokeMachine.cpp

Move the member function code out of the CokeMachine's class structure. Use the scope resolution operator to tie the prototypes left in the class definition to the actual member function code in CokeMachine.cpp. Only the function prototypes and data members should still reside in CokeMachine.h. This includes the constructor.

3. Add include guard to CokeMachine.h

Add an include guard to CokeMachine.h. Use the name COKE_MACHINE_H in your include guard.

4. makefile

Download CokeLib.cpp and CokeLib.h from Canvas. **Do not alter these files.** If you alter them to make your code work, then your code will fail when graded because the graders will not use your versions – they will be using the versions from Canvas to grade your assignment. Be sure to add includes where needed to use the new functions in CokeLib.

Create a new makefile that can compile these 3 files and creates an executable named Code4_xxxxxxxxxx.e

Code4_xxxxxxxxxx.cpp

CokeMachine.cpp

CokeLib.cpp

4. Add two new member functions – setMachineName() and setCokePrice()

Add a new member function called setMachineName(). Add it as menu item 5, "Update Machine Name". Prompt the user "Enter a new machine name". Take in newMachineName and call setMachineName() with one string parameter of newMachineName and no return value. After calling member function from Code4_xxxxxxxxxx.cpp, print "Machine name has been updated".

Add a new member function called setCokePrice(). Add it as menu item 6, "Update Coke Price". Prompt the user "Enter a new Coke price". Take in newCokePrice and call setCokePrice() with one int parameter of newCokePrice and no return value. After calling member function from Code4_xxxxxxxxxx.cpp, print "Coke price has been updated".

5. Add default parameter to CokeMachine's constructor

Update CokeMachine's constructor to take defaults for all parameters.

Default Values – machineName of "New Machine", Coke price of 50, change level of 500 and inventory level of 100.

6. Add a message to CokeMachine's destructor

When the destructor is called, print a message. "CokeMachine" plus the machine's name + "has been destroyed"

Example

CokeMachine Machine Bugs Bunny has been destroyed

7. Command line parameters

Use function, `get_command_line_params`, from `CokeLib`. You should not alter this function. Your code will be graded using the copy available with the assignment – not whatever copy you turn in.

Return type

`void`

Parameters

`argv` and `argc` – `Code4.cpp` passes these

string to hold and **pass back** input file name

string to hold and **pass back** output file name

This function will obtain two named command line parameters

`INPUTFILENAME`

`OUTPUTFILENAME`

This function will extract the string from the command line after `INPUTFILENAME=` and `OUTPUTFILENAME=`. For example, if your code is run as

```
./Code4_xxxxxxxxxx.e INPUTFILENAME=Cat OUTPUTFILENAME=Dog
```

then the function will fill in `InputFileName` with `Cat` and `OutputFileName` with `Dog`. Your program would then have access to those two filenames when function called finished.

Your program will be run using

```
./Code4_xxxxxxxxxx.e INPUTFILENAME=xxxxx OUTPUTFILENAME=yyyyy
```

where `xxxxx` and `yyyyy` are files of any name. Do not extensions or anything else to the filename.

If either command line argument is missing, then an exception of `invalid_argument` will be thrown by the function with a string of

```
"Missing command line parameters  -  - Usage : INPUTFILENAME= OUTPUTFILENAME="
```

`Code4.cpp` will be required to catch and handle this exception.

8. Overload the << operator

Overload `<<` to print out the machine information that menu option 4 "`Display Machine Info`" currently calls member functions to do. Menu option 4 in `Code4_xxxxxxxxxx.cpp` should now be one `cout` statement and a `break`.

9. Code4_xxxxxxxxxx.cpp

`main()` should take in `argv` and `argc`.

Use a try-catch block with `get_command_line_params()` to take action if a parameter is missing. If the function throws an exception, then catch it in `main()`, print the string from the exception object and `exit` from the program to immediately terminate.

Create a `vector` of type `CokeMachine` called `SetOfCokeMachines`.

Open the file passed in with `INPUTFILENAME` using file handle `CokeInputFile`.

Open the file passed in with `OUTPUTFILENAME` using file handle `CokeOutputFile`. Open the file with `ios::out`.

If `CokeInputFile` is open

While not reaching `EOF` of `CokeInputFile`

Use `getline()` to read a line from `CokeInputFile` into `CokeMachineLine` (`string` variable)

Call `ParseCokeLine()` from `CokeLib.cpp` with two parameters

`CokeMachineLine`

`ConstructorValues` - a `vector` of type `string` of size 4

If `ParseCokeLine()` returns TRUE

Using each of the elements returned in `ConstructorValues`, construct a temporary `CokeMachine` object.

Push that temporary object into your `SetOfCokeMachine vector`. The default copy constructor will fire to copy it (you won't be able to see it) and the destructor will fire (you will see that once you customize the destructor) to destroy it when you leave the scope of the `while` loop. You will also see the destructor fire multiple times as the vector grows as you add `CokeMachine` objects.

Else (`CokeInputFile` did not open)

print "`Unable to open file`" and `exit()`

do-while machine picked is not the exit option of 0

Ask user "`Pick a Coke Machine`"

Use a `for` loop to iterate from `0` to `size()` of `SetOfCokeMachines`

For each `CokeMachine` object, print the loop counter and the object's machine name. Make option 0 the exit. You will need to deal with offsetting the vector index with the menu index in order to use 0 as the exit option. Print out a final option to add a new machine.

```
Pick a Coke Machine

0. Exit

1. Machine Bugs Bunny

2. Machine Cecil Turtle

3. Machine Daffy Duck

4. Machine Elmer Fudd

5. Machine Fog Horn

6. Add a new machine

Enter choice
```

```
Machine Bugs Bunny|50|500|50
Machine Cecil Turtle|45|545|45
Machine Daffy Duck|40|540|1
Machine Elmer Fudd|100|1000|10
Machine Fog Horn|35|350|99
```

```
} (choice=0)
{

}
else
{
  switch?
}
```

If choice is 0,

then write all Coke Machines to OUTPUTFILENAME using the same pipe delimited format and order of fields as the input file and use `return` to end program. Each pipe delimited line of file output should be created by calling function

`CreateCokeOutputLine`. Pass each object from `SetOfMachines` and `CreateCokeOutputLine()` will directly access the necessary private data members to form the pipe delimited output line. `CreateCokeOutputLine()` is not a member function. The output file created should be able to serve as an input file for the next run of your program.

If choice is to add a new machine, call the default constructor and add the new machine to the `SetOfCokeMachines vector` and print "`New machine added`".

`do-while` user wants to manipulate a single Coke Machine (same `do-while` already in code)

Display existing menu of options for a single Coke Machines (same menu from previous assignment)

Take choice from menu and allow operations on chosen machine (same code from previous assignment except that it must use the chosen Coke Machine from the "`Pick a Coke Machine`"menu. Remember that if you added a new machine,then that is the machine being manipulated currently.

When the program ends, a custom destructor will be called which will print

```
CokeMachine Machine Bugs Bunny has been destroyed

CokeMachine Machine Cecil Turtle has been destroyed

CokeMachine Machine Daffy Duck has been destroyed

CokeMachine Machine Elmer Fudd has been destroyed

CokeMachine Machine Fog Horn has been destroyed
```