

CPSC 131, Data Structures – Fall 2025

Homework 1: Introduction & Review

Student Learning Objectives:

- Become familiar with creating, compiling, running, and submitting programming assignments on Linux
- Demonstrate mastery of basic OOD/OOP and Modern C++ skills, including
 - managing dynamic memory using smart pointers
 - identity, state, scope, linkage, lifespan
 - working with standard vectors, formatters, and modules
 - overloading insertion, extraction, and spaceship operators
- Demonstrate the ability to translate requirements into solutions in a timely manner
- Refresh your memory and normalize our point of departure. Depending on your background and how long ago you actively practiced programming in C++, some of this may be review and some may seem new to you

Description:

In this assignment, you will play both the role of class developer and the role of class consumer. As class consumer you will create and use objects of this class to solve a simple problem, and then as class developer you will implement the provided class interface. The class itself has a few private attributes, and a multiparameter constructor. Objects of the class have the fundamental capability to format, insert, extract, and compare themselves. The problem being solved is to simply read and dynamically store several objects, and then after they've all been read print them in reverse order. You will reuse class GroceryItem in all future homework programming assignments, so effort you apply now getting it right will greatly benefit you throughout the entire semester.



Part 1 – class developer: Implement class GroceryItem's interface, which is provided to you.

- A member-function interface summary is shown in Figure 1
- In addition, the interface also consists of the non-member insertion and extraction operators, and formatter template specializations.

Part 2 – class consumer: Implement function main() to use the GroceryItem class:

- Read a grocery item¹ from standard input (std::cin) until end of file². For each grocery item read:
 - Store the grocery item in a dynamically created object (e.g., new, make_unique, ...)
 - Store the grocery item's pointer in a standard vector
 - Try prompting for data within the loop's condition clause³
- After you have reached the end of file, write the grocery items⁴ to standard output (std::cout) in reverse order using constant reverse iterators
- Be sure to release the dynamically allocated objects before exiting the program

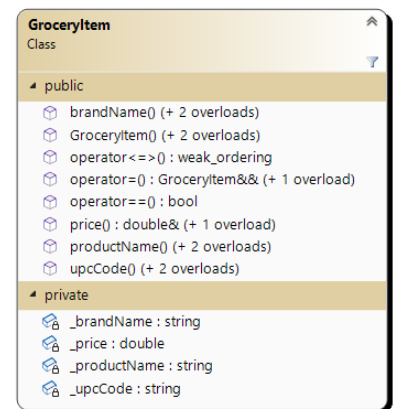


Figure 1: Class GroceryItem Summary

¹ Do not read a grocery item's attributes (three strings and a number), use GroceryItem's extraction operator to read a grocery item. You know you have an incorrect solution if you have defined variables to hold product name, brand name, UPC, or price.

² This program **requires** you to not explicitly open files. Simply write your program extracting data from std::cin. Enter Cntl-D (Linux) or Cntl-Z (windows) to indicate end-of-file. Better yet, create a text file with your input and then simply redirect input from that text file (see below). You know you have an incorrect solution if you have called the ifstream::open function.)

³ This pattern is called the [loop-and-a-half](#) pattern. But do it without a decision statement within the body of the loop. (e.g., no if and no break)

⁴ Again, don't write grocery item attributes; write grocery item objects.

Rules and Constraints:

1. You are to modify only designated TO-DO sections. **The grading process will detect and discard any changes made outside the designated TO-DO sections, including spacing and formatting.** Designated TO-DO sections are identified with the following comments:

```

//////////////////// TO-DO (X) //////////////////////
...
//////////////////// END-TO-DO (X) //////////////////////

```

Keep and do not alter these comments. Insert your code between them. In this assignment, there are 23 such sections of code you are being asked to complete. One of them is in `GroceryItem.cppm` and 22 are in `GroceryItem.cpp`. In addition, you need to create and populate `main.cpp` from scratch.

Hint: In most cases, the requested implementation requires only a single line or two of code. Of course, finding those lines is non-trivial. All can be implemented with less than 9 or 10 lines of code. If you are writing significantly more than that, you may have gone astray.

2. Avoid inserting complex expressions into streams. Instead, use the standard format and standard print functions. Instead of `std::cout << ... << ... <<` and so on, use `std::print(std::cout, "...", ..., ..., and so on.`
3. Avoid the `to_string(...)` like functions. Instead, use `std::format(..., ..., ..., ...)`.
4. Avoid `#include` directives. We are using C++ Modules exclusively, which are imported not included. Rare exceptions are used in very specific circumstances and are specifically noted.

Reminders:

- Multiple submissions may be available and a waiting period between submissions may be enforced.
- The C++ using directive `using namespace std;` is **never allowed** in any file in any deliverable product. Being new to C++, you may have used this in the past. If you haven't done so already, it's now time to shed this crutch and fully decorate your identifiers.
- A clean compile is an entrance criterion. Deliveries that do meet the entrance criteria cannot be graded.
- Object Oriented programming suggests that objects know how to read and write themselves. Classes you write shall overload the insertion and extraction operators.
- Object Oriented programming suggests that objects know how to compare themselves. Classes you write shall overload the spaceship (`<=>`) and equality (`==`) relational operators.
- Always initialize your class's attributes, either with member initialization, within the constructor's initialization list, or both. Avoid assigning initial values within the body of constructors.
- Use `Build.sh` on Ubuntu to compile and link your program. There is nothing magic about `Build.sh`, all it does is save you (and me) from repeatedly typing the very long compile command and all the source files to compile.
- Using `std::system("pause")` is not permitted. If you don't know what this is, good!
- Filenames are case sensitive, both in source code and in your OS file system. Windows doesn't care about filename case, but Linux does.
- You may redirect standard input from a text file, and you must redirect standard output to a text file named `output.txt`. Failure to include `output.txt` in your delivery indicates you were not able to execute your program and will be scored accordingly. A screenshot of your terminal window is not acceptable. See [How to build and run your programs](#). Also see [How to use command redirection under Linux](#) if you are unfamiliar with command line redirection.

Deliverable Artifacts:

Provided files	Files to deliver	Comments
GroceryItem.cppm GroceryItem.cpp	1. GroceryItem.cppm 2. GroceryItem.cpp	Start with the files provided. Make changes in the designated TO-DO sections (only). The grading process will detect and discard all other changes.
	3. main.cpp	Create this file as described above.
	4. output.txt	Capture your program's output to this text file using command line redirection. See command redirection . Failure to deliver this file indicates you could not get your program to execute. Screenshots or terminal window log files are not permitted.
	Readme.txt	Optional. Use it to communicate your thoughts to the grader.
quoted_string_patch.inc		You shan't modify this file. GCC requires a patch to use the std::quoted manipulator with istreams. This patch will be added to your delivery and included during the build process.
RegressionTests/ CheckResults.cppm GroceryItemTests.cppm private_frgement_patch.inc		You shan't modify these files. These files contain code to regression test your GroceryItem class. When you're far enough along and ready to have your class regression tested, then place these files in your project's root directory and Build.sh will find them. Simply having these files in your project's root directory will add them to your program and run the tests – you do not need to #include or import anything, call any functions, or run any "test" programs. These tests will be added to your delivery and executed during the grading process. The grading process expects all tests to pass.
sample_input.txt		A sample set of data to get you started.
sample_output.txt		A sample of a working program's output. Your output may vary.