UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science
Faculty of Engineering, Built Environment & IT
University of Pretoria

COS110 - Program Design: Introduction

Practical 4 Specifications: Operator overloading with dynamic memory

Release date: 04-09-2023 at 06:00
Due date: 08-09-2023 at 23:59
Total Marks: 420

Contents

1 General Instructions

- ***Read the entire assignment thoroughly before you start coding.***
- This assignment should be completed individually; no group effort is allowed.
- **To prevent plagiarism, every submission will be inspected with the help of dedicated software.**
- Be ready to upload your assignment well before the deadline, as no extension will be granted.
- You may not import any of C++'s built-in data structures. Doing so will result in a mark of zero. You may only make use of 1-dimensional native arrays where applicable. If you require additional data structures, you will have to implement them yourself.
- If your code does not compile, you will be awarded a mark of zero. Only the output of your program will be considered for marks, but your code may be inspected for the presence or absence of certain prescribed features.
- If your code experiences a runtime error, you will be awarded a mark of zero. Runtime errors are considered unsafe programming.
- Read the entire specification before you start coding.
- **Ensure your code compiles with C++98**

2 Plagiarism

The Department of Computer Science considers plagiarism a serious offence. Disciplinary action will be taken against students who commit plagiarism. Plagiarism includes copying someone else's work without consent, copying a friend's work (even with consent) and copying material (such as text or program code) from the Internet. Copying will not be tolerated in this course. For a formal definition of plagiarism, the student is referred to http://www.library.up.ac.za/plagiarism/index.htm (from the main page of the University of Pretoria site, follow the Library quick link, and then choose the Plagiarism option under the Services menu). **If you have any form of question regarding this, please ask one of the lecturers to avoid any misunderstanding.** Also note that the OOP principle of code reuse does not mean that you should copy and adapt code to suit your solution.

3 Outcomes

On completion of this practical, you will have gained experience with the following:

- Operator overloading.
- Using dynamic memory in classes and using operators to interact with it.
- Resizable dynamic arrays.

4 Introduction

Operator overloading is a C++ feature that allows programmers to define functions for some of the built-in C++ operators. This can be implemented to allow more elegant use of the functions for a specific class if implemented correctly, but can also cause confusion since there are very few limitations on how the operators can be implemented. For this practical, some operators will be overloaded to implement some functions that make sense with the corresponding operators, but some operators will also be overloaded in a way that is not the normal C++ standard.
C++ allows dynamic arrays but not resizable dynamic arrays. That said, in C++ we usually store dynamic arrays inside a pointer, thus if we were to create new arrays and then just change the pointer, it is possible for us to emulate dynamically resizable dynamic arrays in C++. For this practical, you will be implementing a resizable dynamic array. So when the length of the array has to change, you will need to create a second array, move the elements over, then delete the original one and move the pointer over. **Be careful** with this, since if it is done incorrectly, it will cause memory leaks.

5 Tasks

5.1 crate

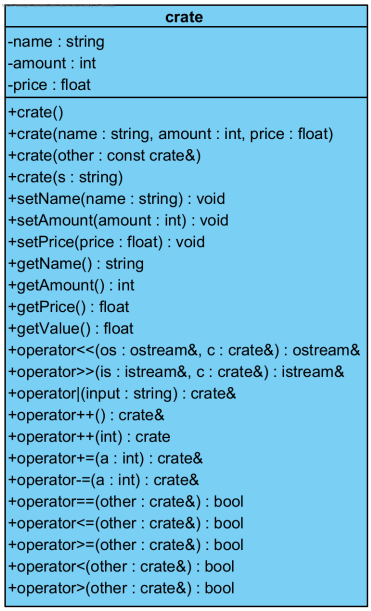

Figure 1: crate UML

Note for the operators, the return type is not mentioned in the spec, as you should be able to figure out what to return using the lecture slides and the textbook.

- Members
  - name: string
    - The name describes the items within the create, i.e. if the crate stores apples, then the crate's name is "apples".
    - If the crate is empty, then the name of this crate will be "empty".
  - amount: int
    - This is the number of items in this crate.
    - If the crate is empty, this number will be 0.
  - price: float
    - This is the price **per item** for the items in this crate.
    - If the crate is empty, this number will be 0.
- Functions
  - crate()
    - This is the default constructor. It should create an empty crate.
  - crate(name: string, amount: int, price: float)
    - This is the parameterized constructor. Use the passed-in parameters to initialize the object.
  - crate(other: const crate&)
    - This is the copy constructor. Use the passed-in parameter to initialize the object.

- crate(s: string)
  - For this constructor, first create an empty crate.
  - Then use the | operator (this will be explained later in the spec) to initialize the object.
- setName(name: string): void
  - Set the corresponding variable to the passed-in parameter.
- setAmount(amount: int): void
  - Set the corresponding variable to the passed-in parameter.
- setPrice(price: float): void
  - Set the corresponding variable to the passed-in parameter.
- getName(): string
  - Return the corresponding variable.
- getAmount(): int
  - Return the corresponding variable.
- getPrice(): float
  - Return the corresponding variable.
- getValue(): float
  - Return this crate's value.
  - The value of a crate is defined as the amount multiplied by the price.
- operator«(os: ostream&, c: crate&): ostream&
  - This function should be declared as a friend function to crate class.
  - This will be used to print out the crate object.
  - The formatting is as follows:
    - | *name*[50] | *amount*[10] | R_[2] | *price*[8] | R_[2] | *value*[8] |
    - A space is indicated with _
    - Values inside square brackets indicate the length of that column. Spaces should be padded to the right to achieve this.
    - Words in *italics* indicate variables or something that must be calculated.
    - All float values should be displayed with two decimal points.
    - This should be returned on its own line.
    - Use the provided Main and output.txt to see an example.
  - *Note for testing :* FitchFork needs to be set up in a special way to allow cin input. Thus for the testing task, use textfiles and fstream to test this function. Remember to upload the textfiles you use.
- operator»(is: istream&, c: crate&): istream&
  - This function should be declared as a friend function to the crate class.
  - This operator will be used to set the member variables of the object.
  - Extract a single line from *is*, then pass that single line as a parameter to the | operator.
  - Note that it can be used for both terminal input and file input.

- operator|(input: string): crate&
  - This operator is used to set the member variables of the crate object.
  - The string passed in will always contain 2 |'s.
  - The format is as follows:

```
name|amount|price
```

  - Note that there are no spaces in this string (unless the name contains a space, in which case it is considered part of the name).
  - Note that some or all of the parameters can be empty. An empty parameter will be indicated using an empty string. If an empty parameter is passed in, that means to leave the corresponding member variable unchanged.
  - Examples:
    - "||" will leave the crate unchanged.
    - "|10|" will change the amount to 10 and leave the name and price unchanged.
    - "apples|5|1.00" will change the *name* to "apples", the *amount* to 5 and the *price* to 1.00.
- operator++(): crate&
  - This is the pre-increment operator.
  - The *amount* of items in the crate should increase by 1.
- operator++(int): crate
  - This is the post-increment operator.
  - The *amount* of items in the crate should increase by 1.
- operator+=(a: int): crate&
  - The *amount* of items in the crate should increase by *a*.
- operator-=(a: int): crate&
  - The amount of items in the crate should decrease by *a*.
  - If the amount of items is less than or equal to 0, set the member variables to show that this crate is now empty. *Look at the Members section to see what an empty crate is.*
- operator==(other: crate&): bool
  - This is the equality operator for crates.
  - Two crates are considered equal if their *names* and *prices* are the same.
  - **The *amount* of items is irrelevant for equality.**
  - **Very important:** Later in the spec when crates are mentioned to being equal or not equal, use this operator.
- operator<=(other: crate&): bool
  - The relational operators **with** the = equal sign indicate that the comparison is based on the *names*.
  - Return the result of <= based on the names of the crates.
- operator>=(other: crate&): bool
  - The relational operators **with** the = equal sign indicate that the comparison is based on the names.
  - Return the result of >= based on the names of the crates.

  - operator<(other: crate&): bool
    * The relational operators **without** the = equal sign indicate that the comparison is based on the *value*.
    * Return the result of <= on the values of the crates (this does not make mathematical sense, but it is the only way to compare name and value separately).
  - operator>(other: crate&): bool
    * The relational operators **without** the = equal sign indicate that the comparison is based on the *value*.
    * Return the result of >= on the values of the crates (this does not make mathematical sense, but it is the only way to compare name and value separately).

5.2 warehouse

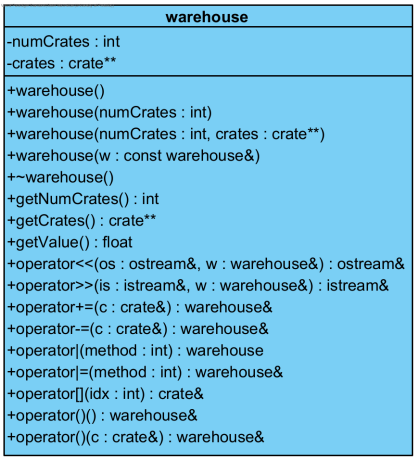

Figure 2: Caption

- Members
  - numCrates: int
    * This is the number of crates in the warehouse.
  - crates: crate**
    * This is a 1D dynamic array of dynamic crate object.
    * The size of this array is numCrates.
- Functions
  - warehouse()
    * This is the default constructor.
    * Initialize the crates variable to be of size 0.
  - warehouse(numCrates: int)
    * Initialize the crates variable to be the size of the passed-in parameter.
    * All the crates should be empty.
    * You may assume that the passed in parameter is greater than or equal to 0.
  - warehouse(numCrates: int,crates: crate**)
    * Initialize the member variables using the passed-in parameters.
    * Use **deep copies** for the crates.

- warehouse(w: const warehouse&)
  - This is the copy constructor.
  - Use **deep copies** for the crates.
- ∼warehouse()
  - This is the destructor for warehouse.
  - This should deallocate all of the dynamic memory for this class.
- getNumCrates(): int
  - This should return the number of crates.
- getCrates(): crate**
  - This should return the *crates* variable.
- getValue(): float
  - This should return the value of the warehouse.
  - The value of the warehouse is defined as the sum of the values of the crates it contains.
- operator«(os: ostream&,w: warehouse&): ostream&
  - This function should be declared as a friend function to the warehouse class.
  - This will be used to print out the warehouse.
  - Print out the following, with every bullet on its own line.
    - The character "-" repeated 80 times. *Hint: Lookup the fill constructor for strings.*
    - Print out of every crate inside this warehouse on its own line.
    - The character "-" repeated 80 times.
    - The following formatted using the same rules as the crate's operator«.

Warehouse[50]	*numCrates*[10]	_[10]	R_[2]	*value*[8]

    - The character "-" repeated 80 times.
- operator»(is: istream&,w: warehouse&): istream&
  - This function should be declared as a friend function to the warehouse class.
  - The first line should be converted to an integer, and this will indicate how many crates will be added.
  - Read in only as many crates as was indicated in the previous bullet. Note there might be more lines, but these **should not** be read in.
  - For every crate, start by creating an empty crate. Then call the insertion stream operator on this crate. Finally, call the warehouse += operator with the crate.
  - Note that it can be used for both terminal input and file input.
- operator+=(c: crate&): warehouse&
  - There are three possibilities for this function. They are listed below:
  - Only perform one of the three possibilities. Start from the top and do the first one that meets the requirements.
    - If there is a crate in the warehouse that is equal to the passed-in parameter, then increase that crate's amount by the amount of the passed-in crate.
    - If there is an empty crate in the warehouse, replace this with the passed-in parameter. Make a **deep copy.**
    - Lastly, increase the array size by 1, such that this crate can be placed at the back. Make a **deep copy.**

- operator-=(c: crate&): warehouse&
  - This needs to loop through the crates array to possibly find a crate that is equal (i.e. have the same *name* and same *price*) to the passed-in crate.
  - If such a crate is found, then the *amount* of the crate that was found is decreased by the amount of the passed-in crate's *amount*.
- operator|(method: int): warehouse
  - This is the sorting operator.
  - This operator should not change the original warehouse but only return a sorted version of this warehouse.
  - The sorting should be done based on the passed-in parameter below, highlighted in bold:
  - If the sorting method is **0**:
    - Sort the crates in alphabetical order using *name*.
    - If there are crates with the same name, then these crates should be sorted in ascending value.
    - You may assume that there will be no crates with the same name and value.
  - If the sorting method is **positive**:
    - Sort the crates in ascending order by *value*.
    - If there are crates with the same value, then these crates should be sorted by name in alphabetical order.
    - You may assume that there will be no crates with the same name and value.
  - If the sorting method is **negative**:
    - Sort the crates in descending order by *value*.
    - If there are crates with the same value, then these crates should be sorted by name in alphabetical order.
    - You may assume that there will be no crates with the same name and value.
- operator|=(method: int): warehouse&
  - This is also a sorting operator.
  - It should follow the same rules as the | operator, but this time the current object must be changed such that the array is sorted according to the passed in parameter.
- operator`[]`(index: int): crate&
  - This is the subscript operator.
  - If the passed-in parameter is inside the valid range of the array, then return the crate at that index.
  - If the passed-in parameter is outside the valid range, then return the crate with the largest value. If there are multiple crates with the same maximum value, then return the crate with the maximum value and then the minimum name (in alphabetical order).
  - You may assume that there will be at least 1 crate inside the warehouse when this is called.

- operator()(): warehouse&
  - This is the clear operator.
  - All empty crates should be removed from the array.
  - The array should be resized such that the order stays the same, but there are no empty crates in the array.
- operator()(c: crate&): warehouse&
  - This is another clear operator.
  - All crates that are equal to the passed-in parameter should be removed from the array as well as any empty crates.
  - The array should be resized such that all crates that equal the passed-in parameter are removed and any empty crates as well.

6 Memory Management

As memory management is a core part of COS110 and C++, each task on FitchFork will allocate approximately 10% of the marks to memory management. The following command is used:

```
valgrind --leak-check=full ./main
```

Please ensure, at all times, that your code *correctly* de-allocate *all* the memory that was allocated.

7 Testing

As testing is a vital skill that all software developers need to know and be able to perform daily, 10% of the assignment marks will be allocated to your testing skills. To do this, you will need to submit a testing main (inside the main.cpp file) that will be used to test an Instructor Provided solution. You may add any helper functions to the main.cpp file to aid your testing. In order to determine the coverage of your testing the gcov [1] tool, specifically the following version *gcov (Debian 8.3.0-6) 8.3.0*, will be used. The following set of commands will be used to run gcov:

```
g++ --coverage *.cpp -o main
./main
gcov -f -m -r -j crate warehouse
```

This will generate output which we will use to determine your testing coverage. The following coverage ratio will be used:

$$\frac{\text{number of lines executed}}{\text{number of source code lines}}$$

and we will scale this ration based on class size.
The mark you will receive for the testing coverage task is determined using table 1:

Coverage ratio range	% of testing mark
0%-5%	0%
5%-20%	20%
20%-40%	40%
40%-60%	60%
60%-80%	80%
80%-100%	100%

Table 1: Mark assignment for testing

Note the top boundary for the Coverage ratio range is not inclusive except for 100%. Also, note that only the function stipulated in this specification will be considered to determine your mark. Remember that your main will be testing the Instructor provided code and as such it can only be assumed that the functions outlined in this specification are defined and implemented.

[1] For more information on gcov please see `https://gcc.gnu.org/onlinedocs/gcc/Gcov.html`

8 Upload checklist

The following files should be in the root of your archive

- main.cpp
- crate.cpp
- warehouse.cpp
- All textfiles that your main.cpp uses.

9 Allowed libraries

- iomanip
- iostream
- sstream
- string

10 Submission

You need to submit your source files, only the .cpp files, on the FitchFork website (https://ff.cs.up.ac.za/). All methods need to be implemented (or at least stubbed) before submission. Place the above-mentioned files in a zip named uXXXXXXXX.zip where XXXXXXXX is your student number. There is no need to include any other files or .h files in your submission. Your code must be able to be compiled with the C++98 standard

For this practical, you will have 10 upload opportunities and your best mark will be your final mark. Upload your archive to the appropriate slot on the FitchFork website well before the deadline. **No late submissions will be accepted!**