
Department of Computer Science
Faculty of ICT
University of Malta

Dr Neville Grech
neville.grech@um.edu.mt

CPS2004 Object-Oriented Programming

Study-Unit Assignment

October 31, 2021

Preamble: This document describes the assignment for *CPS2004: Object Oriented Programming*. This assignment is worth **100%** of the total, final mark for this unit. You are expected to allocate approximately 80 hours to complete the assignment. The dealine is available on VLE.

You may **not** copy code from internet sources or your colleagues. The Department of Computer Science takes a very serious view on plagiarism. For more details refer to plagiarism section of the Faculty of ICT website[1].

Important Notice

This coursework assessment **may** include an interview with an examination board, so that we can make sure you have clearly understood how to solve the problems in this coursework.
You are to regularly commit all of your code to a GitHub or GitLab private repo and give me (nevillegrech) read access. In the documentation, please state the **last commit hash** and **repository URL** on the front page.

1 Deliverables

The code and documentation is to be submitted on the VLE website. Failure to submit this in the required format will result in your assignment not being graded. Please give me (nevillegrech) read accessrepo as soon as you start working on this. Marks will be deducted if you fail to commit and push regularly to this repo.

Only one file is required for electronic submission, a zip archive containing your assignment code and the report. This needs to be uploaded to VLE. The

[1] https://www.um.edu.mt/ict/Plagiarism

code of each task should be located in a top level directory in the archive, named `task1`, `task2`, and `task3`. A single report named `report.pdf` (for all tasks) should be placed in the top level directory.

The report should **not** be longer than 14 pages (including figures and references) and, for each task, should contain the following:

- UML Diagram showing the classes' makeup and interactions
- A textual description of the approach (highlighting any extras you implemented)
- Test cases considered (and test results)
- Critical evaluation and limitations of your solution
- Answers to any questions asked in the task's description

2 Technical Specification

Your code will be compiled and run on Linux. C++ code will be compiled using the `g++` compiler (version 10) from the GNU Compiler Collection. Please make sure to compile C++ using the command line:

```
g++ -Wall -std=c++20 -march=native *.cpp
```

Your Java submissions will be compiled and run using the OpenJDK (version 17). Please make use of standard libraries only. While you are free to use any IDE, make sure that your code can be compiled (and run) from the command line.

Failure to do so will have an impact on your grade. As a requirement, add a bash script (`compile.sh`) in each task directory to compile your task. Also make sure to add a bash script (`run.sh`) in each task directory to execute your task. In these bash scripts, include any command line arguments and test data files – if applicable. Each task should have a `Launcher` file (either `.java` or `.cpp`) which contains a `main(...)` method used to run your solutions. **Note that you should not have any absolute paths hardcoded in your programs/bash scripts.**

3 Tasks

This assignment consists of three programming tasks, in their respective language.

3.1 DAG (in C++)

Build a templatized directed acyclic graph data structure. The graph has to support objects of arbitrary types and encapsulate them inside nodes. It is up to you to design the API, however it has to support the following features:

- The graph data structure needs to **own** the idividual objects.
- The constructor needs to take a list of edges (with obvious checks such as cycles). The ownership of the objects needs to be moved from this list to the graph.
- A method to remove individual nodes.
- A method to return the list of edges, without destroying the in-memory graph structure.

At face value this looks like a very easy question, but marks will be awarded for how elegant, efficient, concise, and correct your code is. You can start working on this before we cover templates, and add templates later.

(Total for task: 33 marks)

3.2 Crypto Exchange (in Java)

Design a simple crypto exchange platform, emphasizing the concepts in **boldface**. Implement some simple functionality where appropriate and write unit tests that exercise this logic.

A commercial exchange would take months to develop, so in this exercise I will be looking at your ability to apply the design principles discussed in class to an artificial problem. You do not need to implement a GUI or CLI. You do not need persistance, nor networking support. The requirements below are not fully specified, but hopefully not conflicting. It will be up to you to interpret these in a way that respects the intention of the client (the examiner).

An exchange platform is a facility where traders can buy and sell different kinds of **CRYPTO** for **FIAT** currencies. Users of the exchange have to **REGISTER** and be **APPROVED** by the **ADMINISTRATORS** in order to use the system. Therefore there are two types of users: **TRADERS** and **ADMINISTRATORS**. To be able to trade a **CRYPTO** on this stock exchange. There are various kinds of **CRYPTO**, each having its own information: total supply, decimals, symbol, etc.

The exchange platform maintains an **ORDER BOOK**.

Orders can be **BUY ORDERS**, **SELL ORDERS**. Each order can also can be a **MARKET** order or **LIMIT** order. Orders are placed by a trader, and include a **QUANTITY**. In addition they have a **BID** or **ASK** price in case they're buying or selling respectively. Market orders do not have a bid or ask price. When orders are sent to the platform, the platform places these on the order book, which is visible to every trader. The system will try to **FILL** orders by matching buy and sell order prices together using a **MATCHING ENGINE**. This can be done by maintaining a sorted (by price) data structure of orders and matching these in a first-come first-served manner. Some orders will be **FILLED**, some won't and some will be partially **FILLED**. Orders that

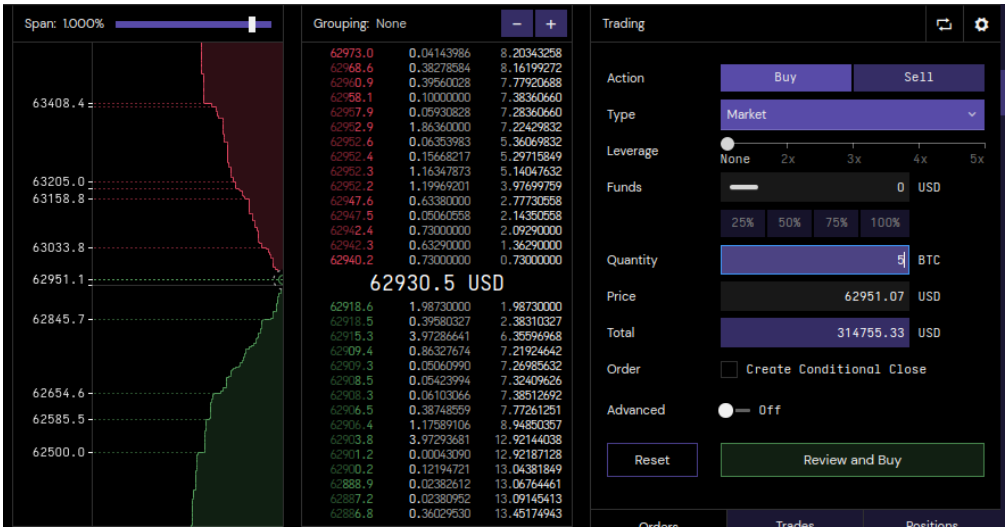

Figure 1: An order book in a real exchange (sorted by price).

ID	CLOSED	TYPE	PAIR	PRICE	VOLUME EXEC'D	COST	STATUS
OBM74U	09/22/21 24:42:15	Sell Limit	USDC/EUR	0.8531 EUR	50,000.0000 USDC	42,655.00 EUR	Filled
O5L4LE	08/20/21 17:54:21	Sell Limit	USDC/EUR	0.8559 EUR	47,904.19161676 USDC	41,001.3080 EUR	Filled
OJM2WC	08/20/21 17:50:47	Sell Limit	USDC/EUR	0.8570 EUR	–	–	Canceled

Figure 2: Examples of orders in an exchange.

are partially **FILLED** may start have their **VOLUME EXECUTED** quantity increase as more matching orders are executed, until they become fully filled. Unfilled orders can also be **CANCELLED**.

Please design your system in such a way as to facilitate future changes, because additional requirements may come up at short notice. Please justify how your system can be easily changed to implement the following requirement:

- Audit trails: The FATF has just imposed new regulations on the government, who has in turn threatened to close down the exchange. In order to avoid closure, your boss asks that every action by a user needs to produce a log (that cannot be tampered with) so that any fraudulent activity can be investigated by competent authorities and scenarios recreated.
- On the other hand, you realize that modifying all the classes and methods will take too much time to meet the deadline. Describe a technical solution of your preference.

(Total for task: 33 marks)

3.3 Big Integers (C++)

C++ supports a number of integer types, but not all. Many cryptographic algorithms need to use 256-bit or 512-bit numbers, or even larger.

Write a templated integer library that supports integers, from 1-bit to 2048-bits, which the user can exploit in the following way:

```
int foo() {
    myuint<1024> i(5); // creates a 1024-bit unsigned int '5'
    myuint<2048> j = i << 1000 + 23; // shifts it by 1000 bits
                                     // and adds 23
    return j.template convert_to<int>() // returns 23
}
```

This library must only support unsigned integers with 2^n bits. A compilation error should be emitted if the user tries to use an arbitrary length integer. Solutions that make use of bitsets or strings will not be accepted.
Detailed instructions:

1. Design the library, implement and overload all binary integer operators (except `*, /, %`), a templated conversion function, together with move and copy constructors from integer types. Getting the design right, so as not to re-implement a lot of functionality will be one of your challenges. A good implementation can be done with very little code, and should be enough to get a decent mark. (18 marks)
2. Implement operators `*, /, %` (6 marks).
3. Make your implementation efficent: space, time, and code. In order to optimize space and time, the templated library should be able to automatically switch to built in integers if it can, without additional overhead. For those feeling really adventurous, go ahead and use bit manipulation techniques or modern CPU features (e.g., AVX...) to reduce overhead, but this is not a requirement (you may find `godbolt.org` to be useful if you go down this path). To get efficiency in code size, it is important to understand C++ semantics and OO concepts so as to better exploit templates and other modularity techniques. Document any design or implementation features related to efficiency. (9 marks)

(Total for task: 33 marks)

4 Grading Criteria

The following criteria, described in Table 1, will be taken into consideration when grading your assignment.

Table 1: CPS2004 Assignment grading criteria. Equal consideration will be given to each.

Overall Considerations	
OO Concepts	Demonstrable and thorough understanding and application of OO concepts and design. You should make use of most of the concepts presented during lectures.
Thoughtful Design	As presented during lectures and clearly documented. Please include: design (in UML), technical approach, testing, critical evaluation and limitations.
Functionality	Completeness and adherence to tasks' specification. Correctness of solutions provided.
Quality and Robustness	Proper error handling. Proper (and demonstrable) testing of solutions. No memory leaks or wasted resources.
Programming Skill	Concise, precise, easy to understand code utilizing appropriate Java and C++ features.
Environment	Use of Linux/Unix setup, version control and reliable build environment.