## Faculty of Engineering, Environment and Computing

# 122COM Introduction to Algorithms

### Assignment Brief 2017/18

| Module Title<br>Introduction to Algorithms | Ind/Group | Cohort<br>Sept | Module Code<br>122COM |
|---|---|---|---|
| Coursework Title (e.g. CWK1)<br>Retake Cw | | | Hand out date:<br>2/4/18 |
| Lecturer<br>David Croft | | | Due date:<br>27/4/18 |
| Estimated Time (hrs): 11 (average)<br><br>Word Limit*: N/A | Coursework type:<br>Code submission | | % of Module Mark<br>60 |
| Submission arrangement online via CUMoodle:<br>File types and method of recording: Single .zip file containing source code.<br>Mark and Feedback date: Within 3 weeks of deadline<br>Mark and Feedback method: Written comments | | | |

Module Learning Outcomes Assessed:

1. Write software to solve a range of problems
2. ~~Implement and use simple searching and sorting algorithms~~
3. Use libraries to extend the functionality of the base language
4. Use basic design and testing strategies

Task and Mark distribution:

**1 Introduction**

The core of this coursework is a single Abstract Data Type (ADT) that you need to implement and test.

The coursework should be undertaken individually without collaboration between students.

All code not written by yourself (other than that on the 122COM Moodle/Github page/s) must be referenced. You are only marked based on the code that you have written. All submissions will be checked against each other and the internet for possible plagiarism and/or collaboration.

You are still expected to complete Phase Test 1 & 2.

**1.1 Support**

If you are unclear on the specifications you should contact the module leader or other member

of teaching staff to ask for clarification. The programming support centre (https://gitlab.com/coventry-university/programming-support-lab/wikis/home) runs during semesters 1, 2 & 3 and is able to assist with resit coursework if you are having programming difficulties.

### 1.2 Submitting

You should submit a single zip file to Moodle with a filename in the following format:
STUDENTNUMBER.zip.
For example if your student ID was 1234567 then you should submit a zip file named 1234567.zip.

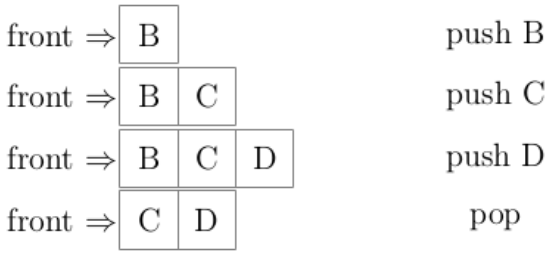
You will be penalised if you do not follow these instructions.

## 2 Task specification

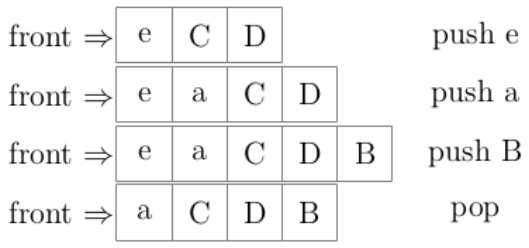
You should be familiar with the language requirements from the 122COM lab sessions but to reiterate:
Information Technology for Business and Multimedia Computing students are allowed to complete the task in either C++14 or Python3.
All other students must complete the task in C++14.

### 2.1 Abstract Data Type (ADT)

You are expected to design and implement an ADT representing an unfair queue.
A queue is an ADT which follows the same rules as real queues, new items are added to the back of the queue (pushing) and old items are removed from the front of the queue (popping).




However, in your unfair queue when you push an lowercase letter onto the queue it should push its way past all of the uppercase letters (because it's smaller and can sneak past). Lowercase letters should only sneak their way past uppercase letters, they can't push their way past other lowercase letters.



Be aware that factors such as sensible variable names, comments, docstrings, use of Object Oriented Programming (OOP), functional decomposition, error handling etc will all be considered when marking your ADT implementation.

#### 2.1.1 Requirements

- Your ADT should be called UnfairQueue (note the capitalisation).
- As a minimum it needs to have front, pop, push and size methods.
  - You can add any additional methods that you want/need.
- It needs to be able to store at least 100 items.
- The items in the unfair queue must be single characters.
  - I.e. if you are coding in Python they will be single character strings.
  - I.e. if you are coding in C++ they will be char variables.
  - Anything that isn't a lowercase letter (i.e. a to z) is considered to be uppercase.
- Use of classes, global variables, raw pointers, pure functions and other indications of good/bad programming practices will taken into account when marking.

#### 2.1.2 Rules regarding use of existing code

- You cannot use any existing queue or priority queue library or implementation.
  - E.g. if you are coding in Python you may not use the Queue, heapq, collections.deque or any other queue modules.
  - E.g. if you are coding in C++ you may not use the queue, priority_queue or any
  - other queue library in the STL, boost, EASTL or any other set of libraries.
- You can use code that has been provided for you in the labs and lectures for inspiration, you should not copy it directly.
  - Remember you are only marked on the code that you write yourself.

### 2.2 Testing

In order to prove that your unfair set implementation works as it should you will need to supply evidence of testing.

If you are testing by hand you must supply a list of test cases, the expected result for each test and the result that you got when you ran that test. Be aware that factors such as sensible test names, comments, test coverage, edge cases etc. will all be considered when marking your tests.

If you are using automated unit tests then you must supply the unit test code which will be run during marking to see the results. Be aware that factors such as sensible test names, comments, docstrings, test coverage, edge cases, use of an automated test library etc will all be considered when marking your tests.

### 2.3 Using the ADT

Once you have written your unfair queue implementation and have tested that it works you need to write a program that uses it.

You will be provided with an SQLite3 database file containing details of famous films. Your program will be expected to get a year as an input. The program should then get a list of all films from that year.
For each film add the letters in the title to an UnfairQueue and then print the UnfairQueue and the director's name.

For example, if the year was 1927 then the films would be "Easy Virtue", "Drop Kick, The" and "My Best Girl" and the program should print:

```
EVasy irtue = Hitchcock, Alfred
DKTrop ick, he = Webb, Millard
MBGy est irl = Taylor, Samd
```

**3 Marking**

30% A bug free unfair queue implementation which meets all the requirements specified in section 2.1
20% A comprehensive testing strategy.
10% Automated unit tests.
20% A bug free program that access the SQLite3 database and meets all the requirements specified in section 2.3.
20% Code elegance and professionalism

Notes:

1. You are expected to use the CUHarvard referencing format. For support and advice on how this students can contact Centre for Academic Writing (CAW).
2. Please notify your registry course support team and module leader for disability support.
3. Any student requiring an extension or deferral should follow the university process as outlined here.
4. The University cannot take responsibility for any coursework lost or corrupted on disks, laptops or personal computer. Students should therefore regularly back-up any work and are advised to save it on the University system.
5. If there are technical or performance issues that prevent students submitting coursework through the online coursework submission system on the day of a coursework deadline, an appropriate extension to the coursework submission deadline will be agreed. This extension will normally be 24 hours or the next working day if the deadline falls on a Friday or over the weekend period. This will be communicated via email and as a CUMoodle announcement.

**Marking Rubric**

| Guide grading rubric | 0% | 20% | 40% | 60% | 80% | 100% |
|---|---|---|---|---|---|---|
| UnfairQueue | UnfairQueue implementation does not meet requirements at all or cannot be made to run without significant changes to the code. | UnfairQueue implementation does not meet that majority of requirements or cannot be made to run without some changes to the code. | UnfairQueue implementation meets the majority of the requirements and is mostly bug free. | UnfairQueue implementation meets the majority of the requirements and is free of known bugs. | UnfairQueue implementation meets all of the requirements and is mostly bug free. | UnfairQueue implementation is complete and bug free and of sufficient standard that it could be used in future programs. |
| Testing of UnfairQueue | No tests have been written or run. | Evidence of a limited number of tests written and run. | A small number of test cases that check some of the code. | A set of tests that check the major features of the code. | A set of tests that covers most of the code or includes edges cases but not both. | A full set of tests that cover all the code including testing edge cases. |
| Automated testing of UnfairQueue | No tests have been written or run. | Evidence of a limited number of unit tests written and run. | All a small number of test cases that check some of the code. | A set of tests that check the major features of the code. | A set of automated tests that covers all methods or includes edges cases but not both. | A full set of automated tests that cover all methods including testing edge cases. |
| SQLite program | Program cannot be made to run. | Minor edits are needed to the code in order to get program to run. | Code runs and meets some requirements and has bugs. | Code runs and meets most requirements and has bugs. | Code runs and meets all requirements but has bugs. | Program is fully functional with no identified bugs. |
| Code elegance and professionalism | No comments, variable and function names are not descriptive. | Variable and/or functions names are not descriptive or there are no comments. | Some comments, variable and function names are meaningful and useful. | Most comments, variable and function names are meaningful and useful. | Comments, variable and function names are meaningful and useful. | Code is fully commented with meaningful comments, variable and function names. |
| | Variable type and/or data structure choices are entirely unsuitable and don't work. | Variable type and/or data structure choices are entirely unsuitable. No thought given to efficiency. | Variable type and/or data structure choices are unsuitable. No thought given to efficiency. | Variable and data structures used are suitable. Code is not efficient. | Variable and data structures used are suitable. Code could be more efficient. | Suitable variable types and data structures have been used in all places, code is efficient. |
| | Coding style is bad, indentation is nonsensical or absent. | Coding style is inconsistent, some indentation is present. | Coding style is present, indentation is present. | Coding style is mostly consistent, indentation is clear. | Coding style is consistent, indentation is clear. | Coding style and indentation is clear and consistent and aids the readability of the code. |
| | Submission rules were not followed. | | | | | Submission rules were followed correctly. |