Tasks

University of South Wales Prifysgol De Cymru

Faculty of Computing, Engineering and Science

Assessment Cover Sheet and Feedback Form 2018-19

Module Code:	Module Title:	Module Team:
CS2S560	Data Structures and Algorithms with Object Oriented Programming	Emlyn Everitt, Janusz Kulon

Assessment Title and Tasks:	Assessment No.
ZZ-Practical Assessment 1	1

Date Set:	Submission Date:	Return Date:
24-Sep-2018 09:00	18-Jan-2019 23:59	15-Feb-2019 23:59

IT IS YOUR RESPONSIBILITY TO KEEP RECORDS OF ALL WORK SUBMITTED

Marking and Assessment

This assignment will be marked out of 100%

This assignment contributes to 50% of the total module marks.

Learning Outcomes to be assessed (as specified in the validated module descriptor https://icis.southwales.ac.uk/):

1) Demonstrate knowledge, comprehension and discernment in the efficient application of common data structures and algorithms, and collections.
2) Demonstrate knowledge, comprehension and discernment in the efficient application of object-oriented programming.

Provisional mark only: subject to change and / or confirmation by the Assessment Board

Tasks

Grading Criteria:

	Fail (0/29)	Narrow Fail (30/39)	3rd Class / Pass (40/49)	Lower 2nd Class / Pass (50/59)	Upper 2nd Class / Merit (60/69)	1st Class / Distinction (70/100)
Class encapsulated integer stack implementation (20%) Individual	☐ Very poor Class encapsulated integer stack implementation ☐ Compilation errors ☐ Partially implemented functionality contains many mistakes ☐ Very poor adherence to the given API specification ☐ Incorrect handling of the exception	☐ Poor Class encapsulated integer stack implementation ☐ Compilation errors ☐ Partially implemented functionality with significant mistakes ☐ Significant issues with the adherence to the given API specification ☐ Significant problems with the handling of the exception	☐ Satisfactory Class encapsulated integer stack implementation ☐ No compilation errors ☐ Program design is logical, functional but some some inefficiencies in the code ☐ Some issues with the adherence to the given API specification ☐ Issues with the handling of the exception	☐ Good Class encapsulated integer stack implementation ☐ No compilation errors ☐ Program design is logical, fully functional ☐ Good adherence to the given API specification ☐ Exception handled correctly	☐ Very good Class encapsulated integer stack implementation ☐ No compilation errors ☐ Program design is logical, fully functional. Control structures are used very well ☐ Very good adherence to the given API specification ☐ Exception handled well	☐ Excellent Class encapsulated integer stack implementation ☐ No compilation errors ☐ Program design is logical, fully functional. Excellent use of control structures ☐ Excellent adherence to the given API specification ☐ Exception handled very well
Class encapsulated integer queue implementation (20%) Individual	☐ Very poor Class encapsulated integer queue implementation ☐ Compilation errors ☐ Partially implemented functionality contains many mistakes ☐ Very poor adherence to the given API specification ☐ Incorrect handling of the exception	☐ Poor Class encapsulated integer queue implementation ☐ Compilation errors ☐ Partially implemented functionality with significant mistakes ☐ Significant issues with the adherence to the given API specification ☐ Significant problems with the handling of the exception	☐ Satisfactory Class encapsulated integer queue implementation ☐ No compilation errors ☐ Program design is logical, functional but some inefficiencies in the code ☐ Some issues with the adherence to the given API specification ☐ Issues with the handling of the exception	☐ Good Class encapsulated integer queue implementation ☐ No compilation errors ☐ Program design is logical, fully functional ☐ Good adherence to the given API specification ☐ Exceptions handled correctly	☐ Very good Class encapsulated integer queue implementation ☐ No compilation errors ☐ Program design is logical, fully functional. Control structures are used very well ☐ Very good adherence to the given API specification ☐ Exception handled well	☐ Excellent Class encapsulated integer queue implementation ☐ No compilation errors ☐ Program design is logical, fully functional. Excellent use of control structures ☐ Excellent adherence to the given API specification ☐ Exception handled very well
Class encapsulated prioritised scheduler (queue subclass) implementation (40%) Individual	☐ Very poor Class encapsulated prioritised scheduler (queue subclass) implementation ☐ Compilation errors ☐ Partially implemented functionality but contains many mistakes ☐ Very poor adherence to the given API specification ☐ Incorrect handling of the exception ☐ Incorrect handling of Items with different priorities ☐ Computationally inefficient ☐ Item blocking prevention not implemented	☐ Poor Class encapsulated prioritised scheduler (queue subclass) implementation ☐ Compilation errors ☐ Partially implemented functionality with significant mistakes ☐ Significant issues with the adherence to the given API specification ☐ Significant problems with the handling of the exception ☐ Significant problems with handling of Items with different priorities ☐ Could be computationally more efficient ☐ Significant problems with item blocking prevention	☐ Satisfactory Class encapsulated prioritised scheduler (queue subclass) implementation ☐ No compilation errors ☐ Program design is logical, functional but some inefficiencies in the code ☐ Some issues with the adherence to the given API specification ☐ Issues with the handling of the exception ☐ Some issues with handling Items with different priorities ☐ Could be computationally more efficient ☐ Some issues with item blocking prevention	☐ Good Class encapsulated prioritised scheduler (queue subclass) implementation ☐ No compilation errors ☐ Program design is logical, fully functional ☐ Good adherence to the given API specification ☐ Exception handled correctly ☐ Some minor issues with handling Items with different priorities ☐ Could be computationally more efficient ☐ Some minor issues with the item blocking prevention	☐ Very good Class encapsulated prioritised scheduler (queue subclass) implementation ☐ No compilation errors ☐ Program design is logical, fully functional. Control structures are used very well ☐ Very good adherence to the given API specification ☐ Exception handled well ☐ Items with different priorities handled with fairness ☐ Reasonably computationally efficient ☐ Item blocking prevention implemented well	☐ Excellent Class encapsulated prioritised scheduler (queue subclass) implementation ☐ No compilation errors ☐ Program design is logical, fully functional. Excellent use of control structures ☐ Excellent adherence to the given API specification ☐ Exception handled very well ☐ Items with different priorities handled with fairness ☐ Computationally very efficient ☐ Item blocking prevention implemented very well
A complete program listing containing comprehensive comments (20%) Individual	☐ No or very limited comments frequently unhelpful or misleadingA complete program listing containing comprehensive comments	☐ Comments are incomplete and internal documentation is inadequate	☐ Not all the comments are complete or internal documentation is inadequate. Some comments are unhelpful or occasionally misleading	☐ Almost all the comments are complete but internal documentation is in some small fashion inadequate. Comments usually clarify meaning. Unhelpful comments may exist	☐ Comments are nearly complete. Internal documentation is nearly complete and generally well suited to the program Comments generally clarify meaning where needed	☐ Comments are completed to a very good standard. Internal documentation is complete and well suited to the program. Comments clarify meaning where needed

Tasks

Scenario

You work for Operating Systems Ltd, a company specialising in the retailing of operating systems and a keen competitor of Microsoft. One day your boss comes to you to say that he wants you to develop:

- An integer stack
- An integer queue
- A scheduling algorithm (subclass of queue) – using an integer for priority and storing integer values to represent process IDs

for their main flagship OS, Windowless 9.

Towards this end, the company has provided you with a **public API specification** (see Appendix A) that you **must adhere** to in the delivery of this functionality. You are also required to code everything yourself, restricting your use of libraries to <iostream> only. Make sure that you handle exceptions (e.g. when removing an item from an empty stack or queue). Also, make sure that you **do not include** the 'main' function in your submission.

The **scheduling algorithm** needs to provide the following functionality, as discussed in lectures:

- Be a subclass of the Queue class
- Use an int value as a process ID
- Use an int value as a priority (i.e. 1 = low, 10 = high)
- Handle exceptions in case the priority value exceeds the above range
- Be able to schedule as many processes as there is memory available; this memory will need to be allocated on demand from the heap

Additional marks will then be awarded for:

- Incorporating a system of prioritisation that allows each process to have a priority value assigned to it and to then be processed in order of that priority outside of the default FIFO ordering
- Developing a system that prevents blocking – e.g. when the continual addition of high priority events prevent lower priority events from being processed

Extra marks will be awarded to those who:

- Utilise efficient ways to deliver some of the above functionality (e.g. O(log N))

The highest marks will only be awarded to those students who address some or all of the more complex issues stated above. Please examine the grading criteria for further details of the assessment.

Tasks

Code Comments

It is of particular importance that your code is well commented. The comments you make will form your documentation for this assignment and the quality of those comments will contribute significantly to your overall mark. Comments should attempt to describe the functionality of each section of code and how it fits into the overall behaviour of the program rather than just superficially describing what each line of code does in isolation.

e.g.

Right

```
x++;    // x is the iterator used to keep track of the position in the data structure
        //which is incremented each time the code loops
```

Wrong

```
x++     //x is incremented
```

It should also be noted that the marks awarded for comments will be weighted in favour of the feature complexity they are attempting to describe; with comments successfully describing the more advanced features resulting in the most marks being earned.

Submission

When submitting your work, please submit your code in **a single .cpp file** to Unilearn.

Appendix A

```cpp
class Node
{
public:
	Node(int value, Node* nextptr = NULL, Node* prevptr = NULL, int currentpriority = 0);

	int getVal(void);

	Node* getNext(void);

	Node* getPrev(void);

	void setVal(int value);

	void setPrev(Node* prevptr);

	void setNext(Node* nextptr);

	int getPriority(void);

	void setPriority(int priority);

private:
	Node* next;
	Node* prev;
	int priority;
	int value;
};

class Stack
{
public:
	Stack(void);

	~Stack(void);

	void Push(int value);

	Node* NodePop(void);

	int Pop(void);

private:

	Node* top;
};

class Queue
{
public:
	Queue(void);

	~Queue(void);

	void Enqueue(int i, int priority = 0);

	int Dequeue(void);

protected:

	Node* back;
	Node* front;

private:

	virtual Node* NodeDequeue(void);
};
```

```
class Scheduler : public Queue
{
private:

        Node* NodeDequeue(void);

};
```