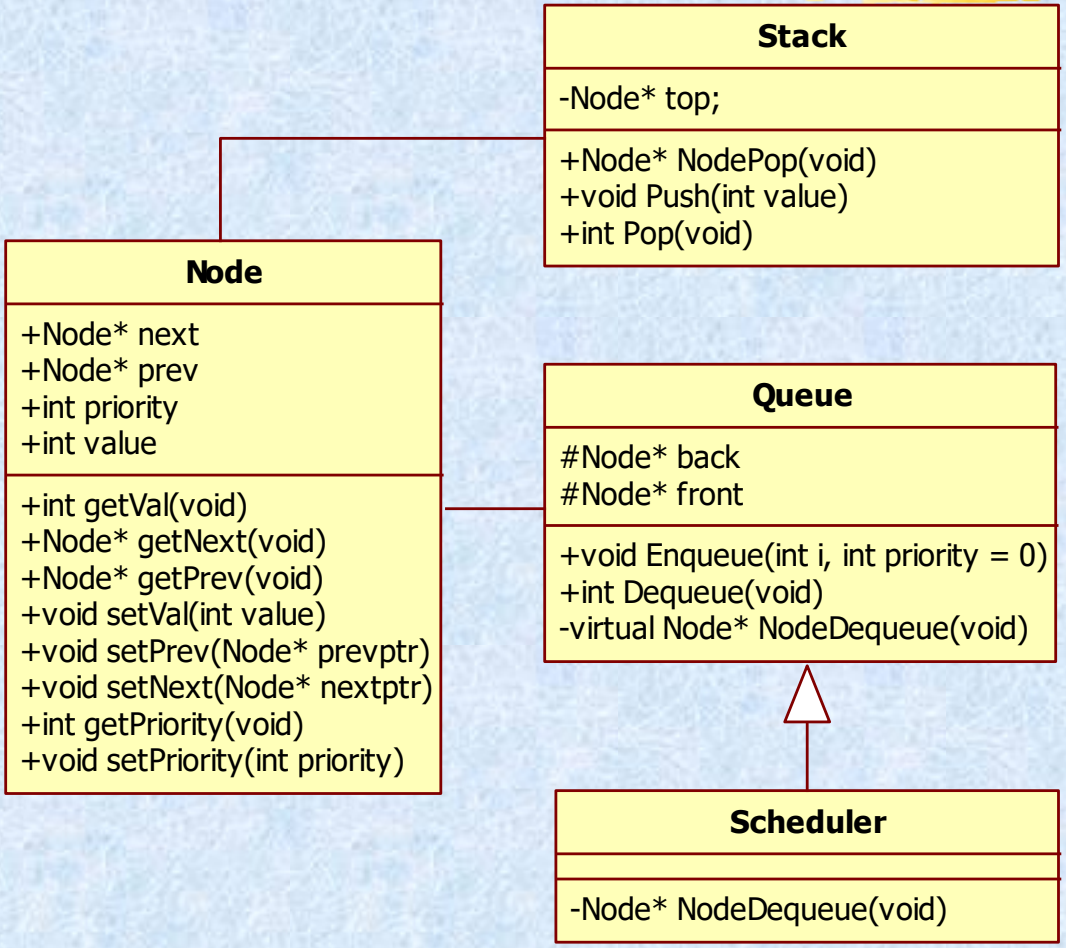
Coursework

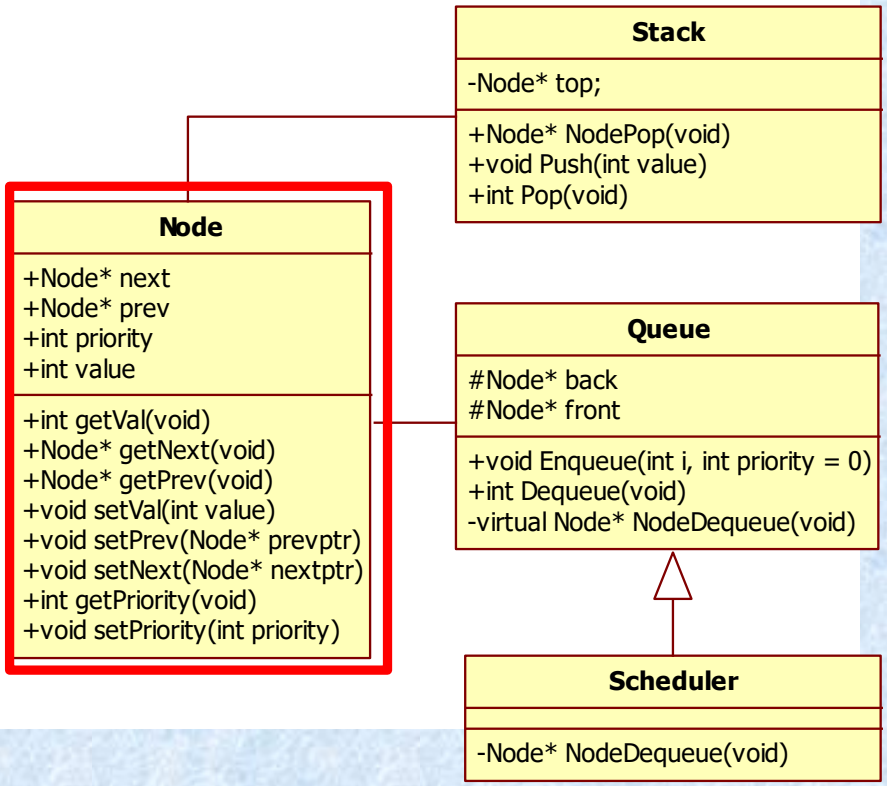


You need to develop:

- **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**

Coursework - Node

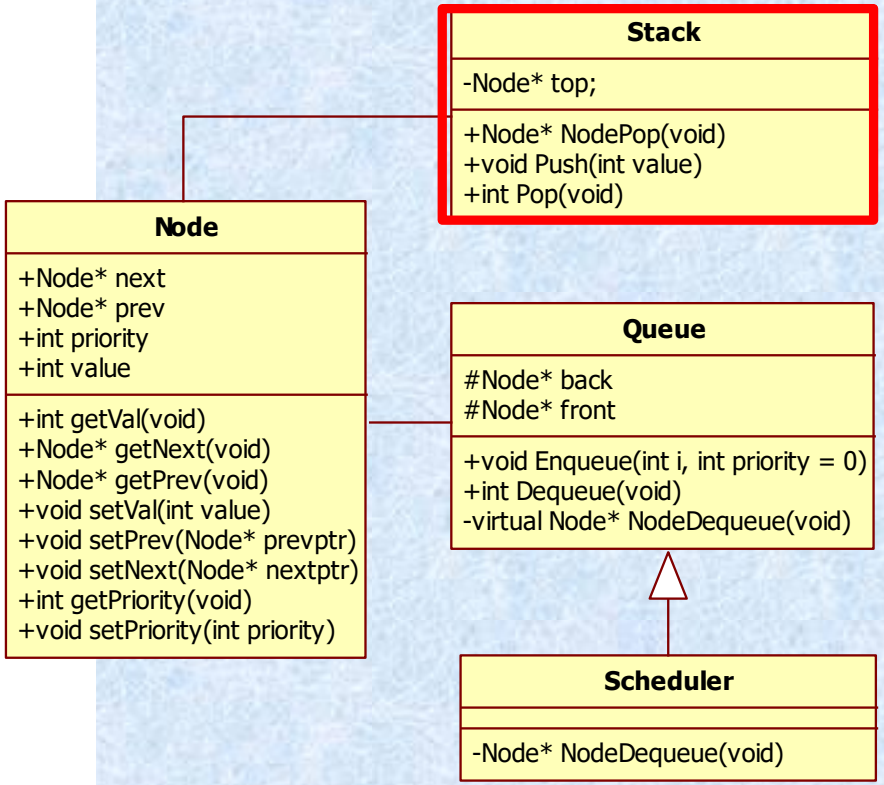
```
class Node
{
public:
        Node(int value, Node* nextptr = NULL, Node* prevptr = NULL, int currentpriority = 0);

        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;
};
```



Don’t add any new methods or attributes

Coursework - Stack

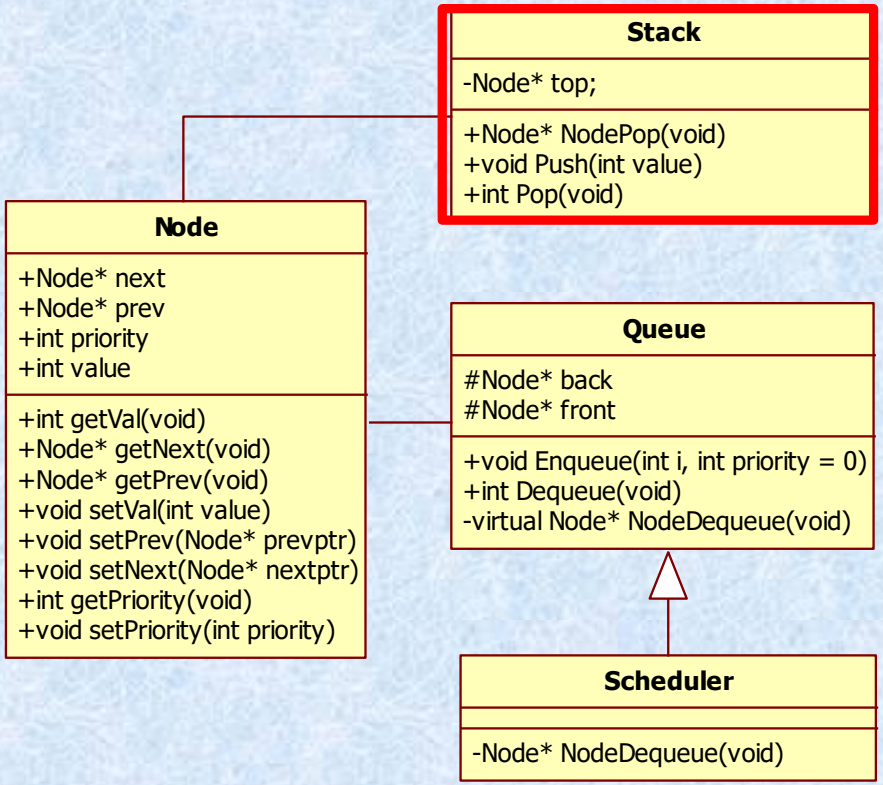
```
class Stack
{
public:
        Stack(void);

        ~Stack(void);

        void Push(int value);

        Node* NodePop(void);

        int Pop(void);

private:

        Node* top;
};
```



Don’t add any new methods or attributes

Coursework - Stack

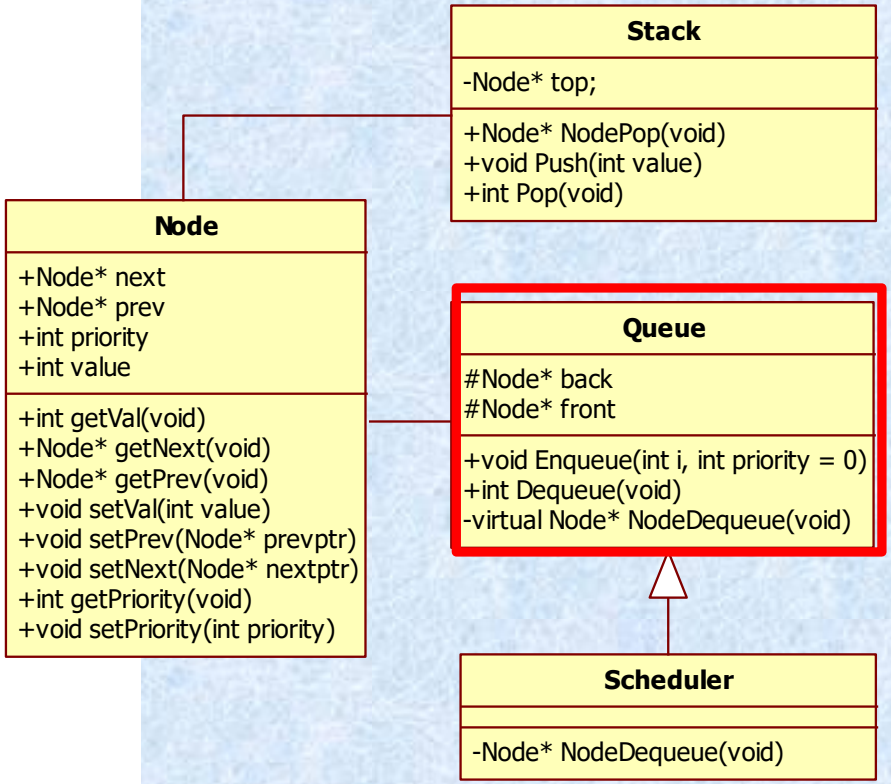
```
Stack myStack;
myStack.Push(1);
myStack.Push(2);
……
myStack.Pop();
myStack.Pop();
……
```




```
1 2 3 4 5 6 7 8 9 10 11 12 13 14
14 13 12 11 10 9 8 7 6 5 4 3 2 1
```

Coursework - Queue

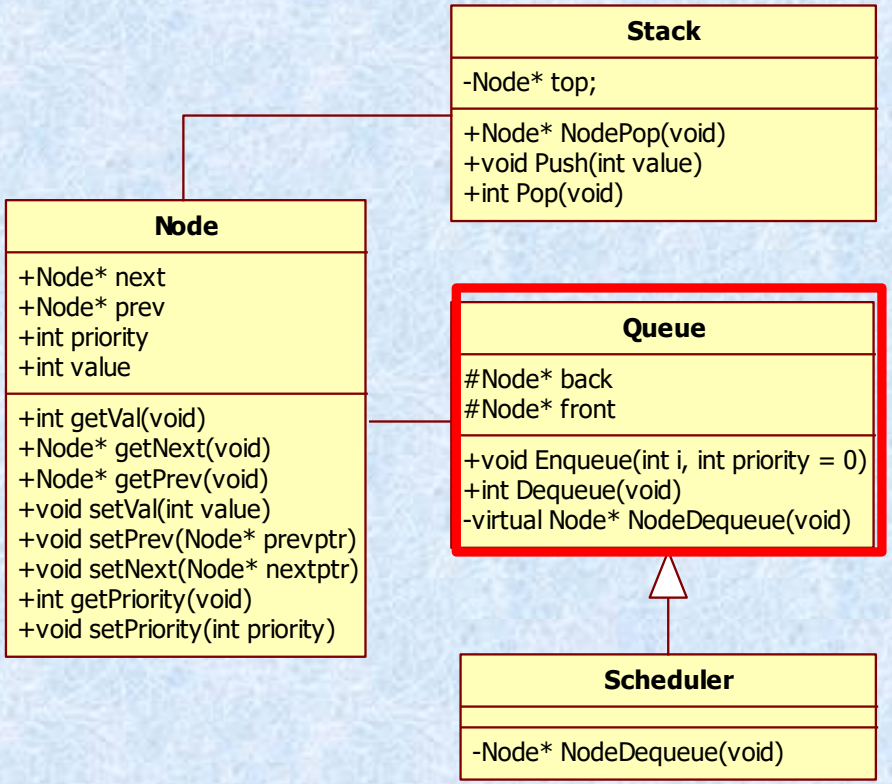
```
class Queue
{
public:
        Queue(void);

        ~Queue(void);

        void Enqueue(int i, int priority =

        int Dequeue(void);

protected:

        Node* back;
        Node* front;

private:

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



Don’t add any new methods or attributes

Coursework - Queue

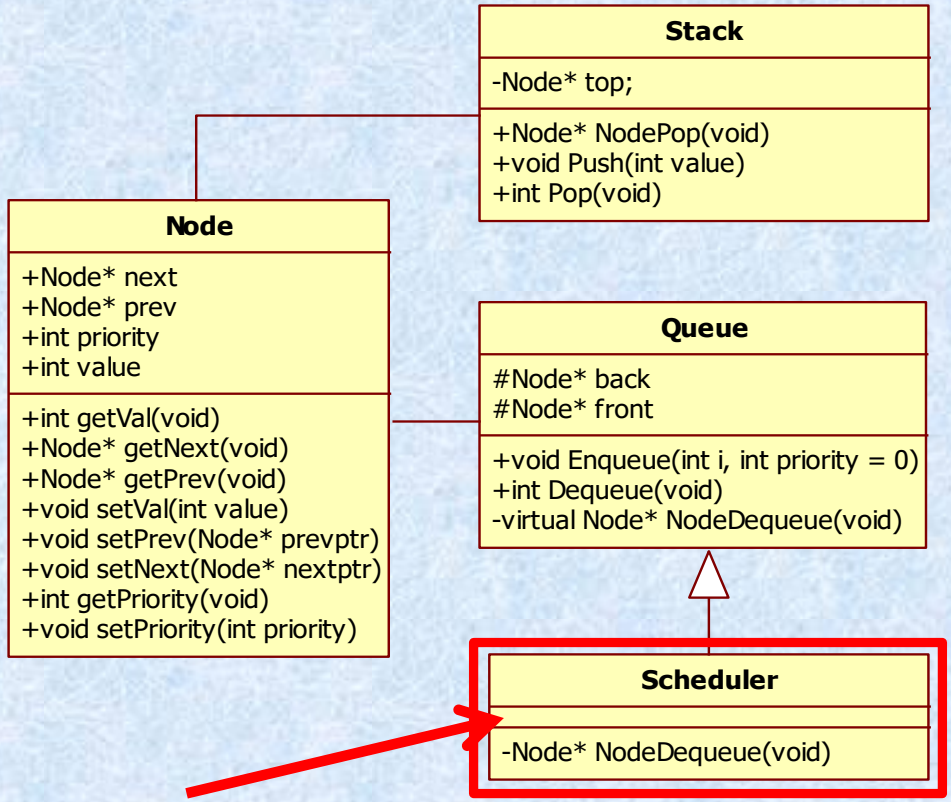
```
Queue myQueue
myQueue.Enqueue(1);
myQueue.Enqueue(2);
.....
myQueue.Dequeue();
myQueue.Dequeue();
.....
```




```
1  2  3  4  5  6  7  8  9  10  11  12  13  14
1  2  3  4  5  6  7  8  9  10  11  12  13  14
```

Coursework - Scheduler

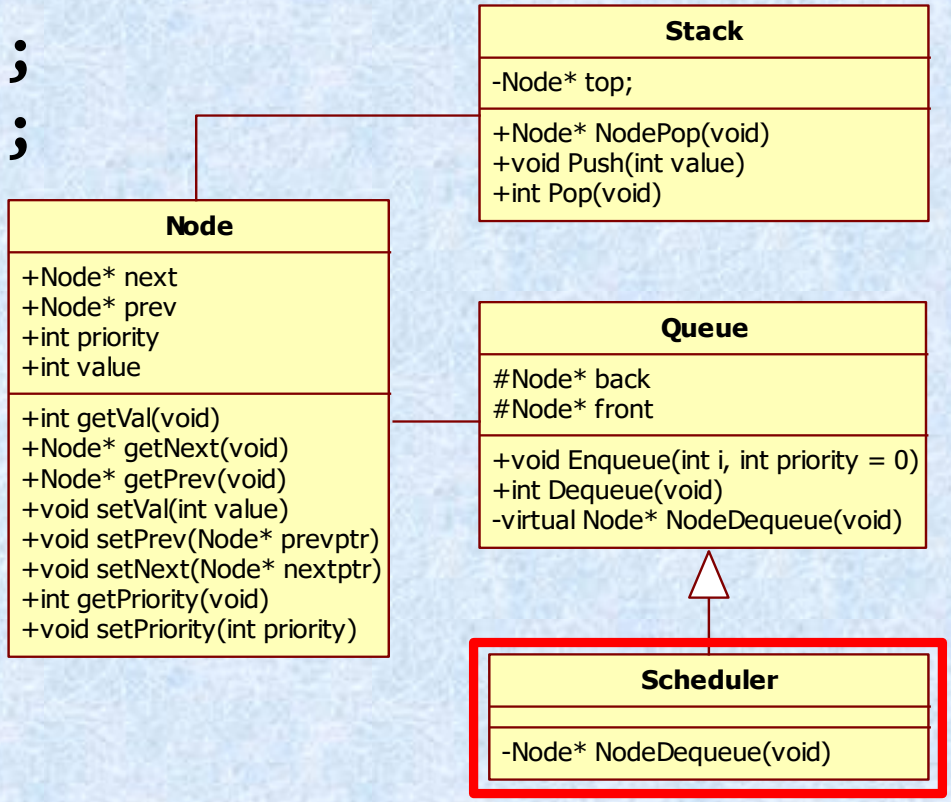
```
class Scheduler : public Queue
{
private:

        Node* NodeDequeue(void);

};
```



You can add ONLY private methods and attributes

Coursework - Scheduler

```
Scheduler myScheduler;
myScheduler.Enqueue(1, 1);
myScheduler.Enqueue(2, 2);
......
myScheduler.Dequeue();
myScheduler.Dequeue();
......
```




```
1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 10
8 9 10 10 6 8 9 7 6 7 5 5 4 4 1 3 2 3 1 2
```

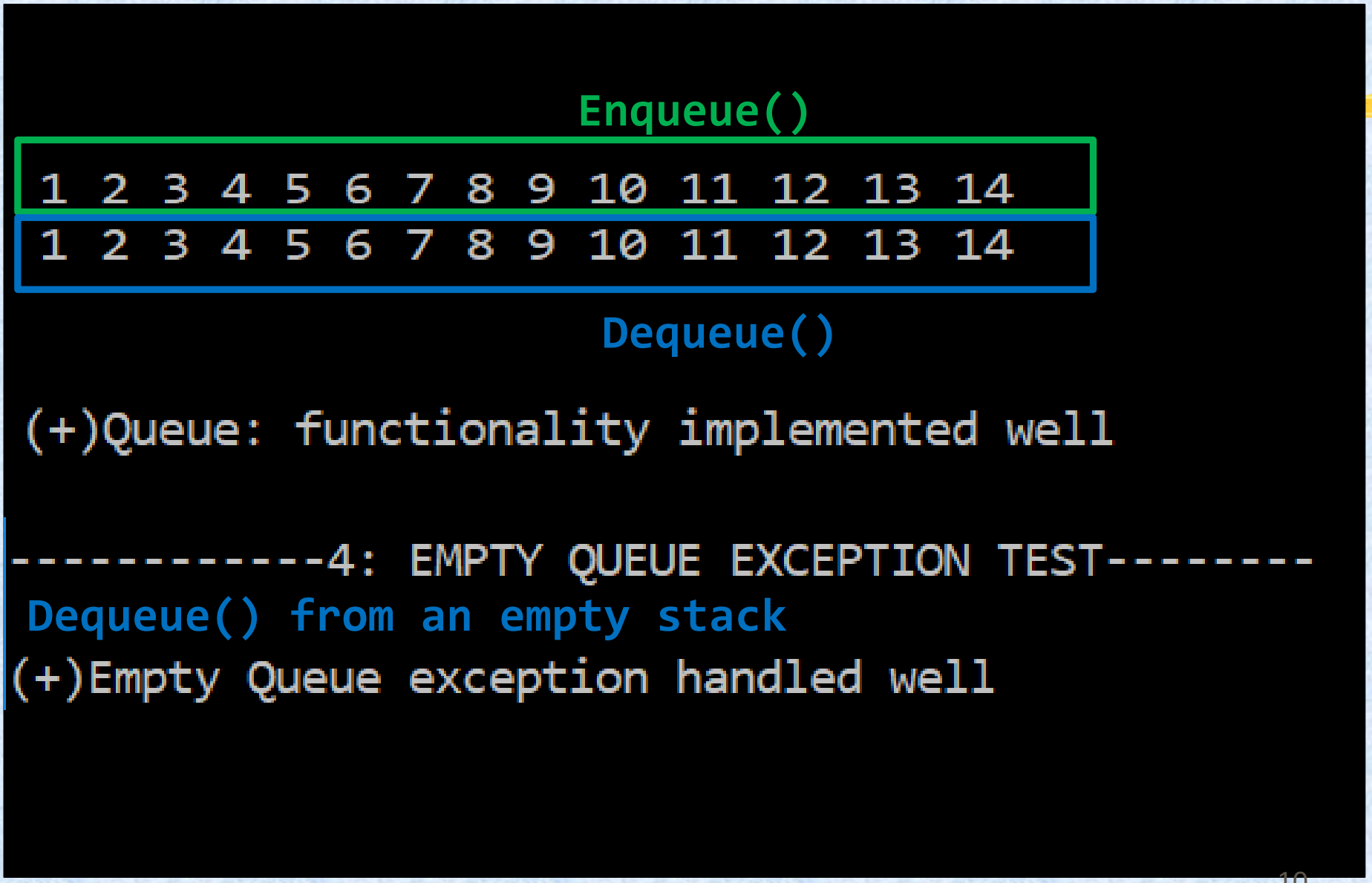
Stack Tests

```
------------1: STACK BASIC TEST------------
1 2 3 4 5 6 7 8 9 10 11 12 13 14
14 13 12 11 10 9 8 7 6 5 4 3 2 1
(+)Stack: functionality implemented well


------------2: EMPTY STACK EXCEPTION TEST--
Pop() from an empty stack
(+)Empty Stack exception handled well
```

Push()

Pop()

Queue Tests


Enqueue()
1 2 3 4 5 6 7 8 9 10 11 12 13 14
1 2 3 4 5 6 7 8 9 10 11 12 13 14
Dequeue()
(+)Queue: functionality implemented well
----------4: EMPTY QUEUE EXCEPTION TEST--------
Dequeue() from an empty stack
(+)Empty Queue exception handled well

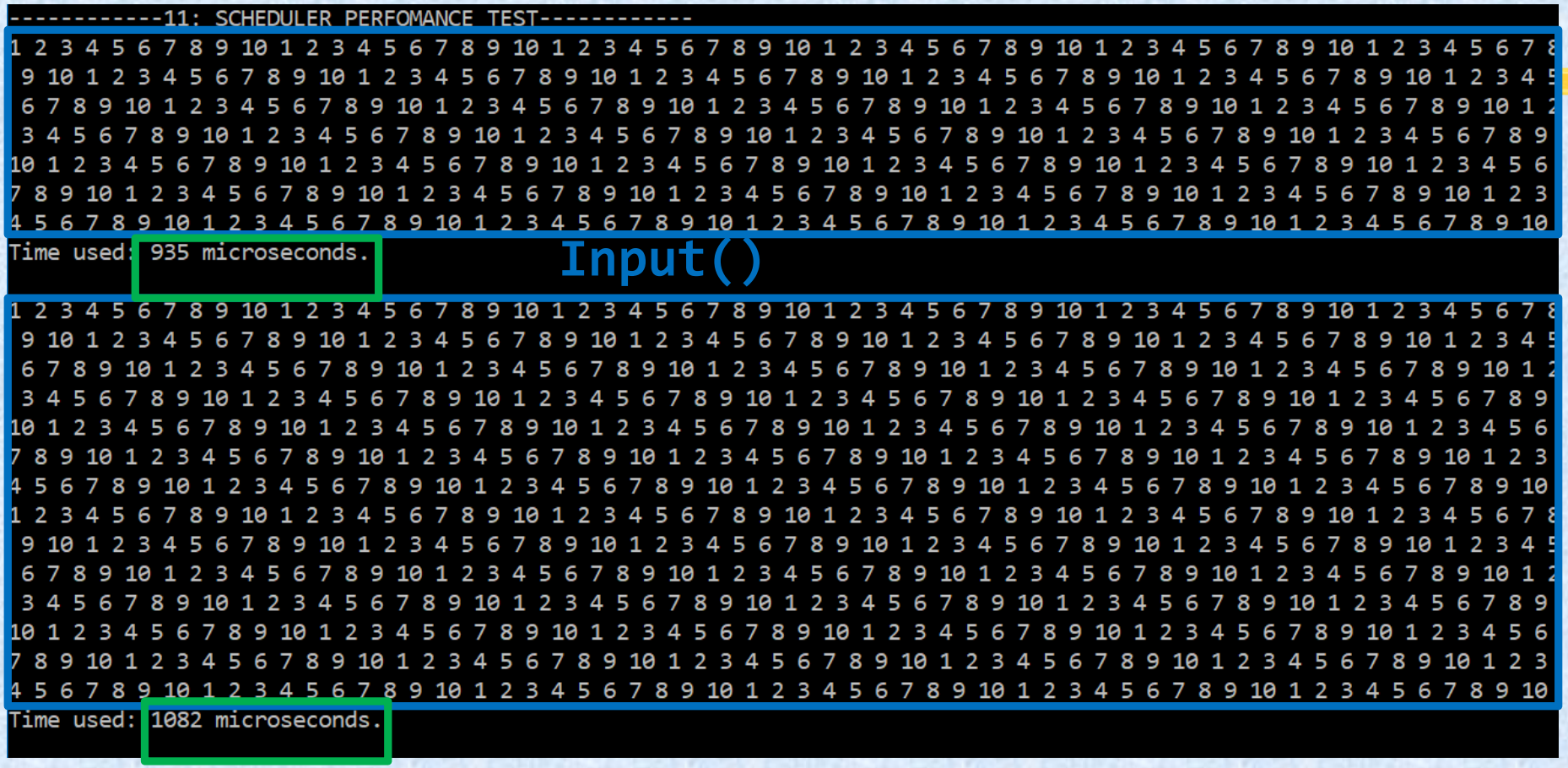
Scheduler Tests

```
Enqueue()
------------5: SCHEDULER BASIC TEST---------
1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 10

Dequeue()
8 9 10 10 6 8 9 7 6 7 5 5 4 4 1 3 2 3 1 2

------------6: EMPTY SCHEDULER EXCEPTION TEST---------

(+)Scheduler: Empty scheduler exception handled well
```

Scheduler Tests

```
------------7: SCHEDULER ITEM WITH ZERO PRIORITY TEST------------
myScheduler.Enqueue = 0 Scheduler.Dequeue() = -1
(+)Scheduler:Items with priority <= 0 handled by throwing exception in NodeDequeue()

------------8: SCHEDULER OUT OF RANGE PRIORITY TEST------------
myScheduler.Enqueue = 20 Scheduler.Dequeue() = -1

(+)Scheduler: Items with priority > 10 priority handled by throwing exception in NodeDequeue()


------------9: SCHEDULER FAIRNESS TEST-----------
Enqueue: 1 2 3 4 5 6 7 8 9 10
Dequeue: 10 9 7 8 6 4 1 5 3 2



------------10: SCHEDULER BLOCKING TEST------------
Enqueue: 1 2 3 4 5 6 7 8 9 10
Flood with Enqueue(10, 10)
(+)Scheduler: Item blocking prevention succesfully implemented
```

Scheduler Tests



Output processing time example

Coursework

- You **MUST adhere to public API specification** in the delivery of this functionality.
- You are also required to code everything yourself, **restricting your use of libraries to <iostream>**.
- Also, make sure that you **DO NOT INCLUDE** the **'main' function** in your submission.

Coursework

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**)
- Be able to schedule as many processes as there is memory available; this **memory will need to be allocated on demand** from the **heap**

Coursework

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

Coursework

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.

Coursework

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

Coursework

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
```

Coursework

Submission

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

Stack

	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

20%

Queue

	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 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

20%

Scheduler

	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 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

40%

Comments

	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)
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

20%