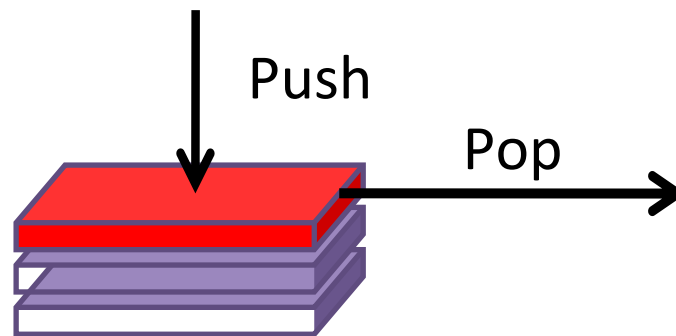


The Stack

A Stack memory structure is a Last In First Out (LIFO) queue. When data is added to memory, it is placed on top of the 'stack' and it will be the first piece of data to be recalled. This behaviour is governed by two operations; push and pop.



The Stack is simple to implement, fast and suitable for certain activities such as function calls. However, the Stack is limited by LIFO operation and its integrity falls apart when accessed by operations other than push and pop. The Stack also has a vulnerability - when a program is running, it is provided the location of the top of the Stack and thus it is possible to manipulate the data within the Stack by accessing its memory location directly.

Implementing a Stack in C++ - Theory

In C++, it is easy to form a Stack using a chain of Nodes that each contain a value of that Node and a pointer to the next Node in the chain. Two functions, push and pop, handle adding and removing a Node directly to the top of the chain.

This yields two classes. A Node class for an object that contains the Node value and a pointer for linking, and a Stack class that is used to interface with the Node chain by providing push and pop functions, and a pointer to the current top Node. The Stack class will need to facilitate an organised destruction of the Node chain to prevent memory leakage as well.

Implementing a Stack in C++ - Classes

- Node
 - Purpose: represents a Node within the memory structure chain
 - **Public members**
 - T **tValue**: templated value of the current Node
 - Node* **ptrNext**: pointer to the memory location of the next Node in the chain
- Stack

- Purpose: represents the Stack memory structure
- **Public methods**
 - **~Stack(void)**: destructor of Stack class; pops and deletes all Nodes upon class destruction
 - void **push(T)**: creates new Node with the provided value and adds it to the top of the Node chain
 - Node<T>* **nodePop(void)**: removes the current top Node from the chain and returns the Node for deletion
 - T **pop(void)**: deletes the current top Node and returns its value
- **Protected members**
 - Node<T>* **ptrTop**: pointer to the memory location of the current top Node in the chain

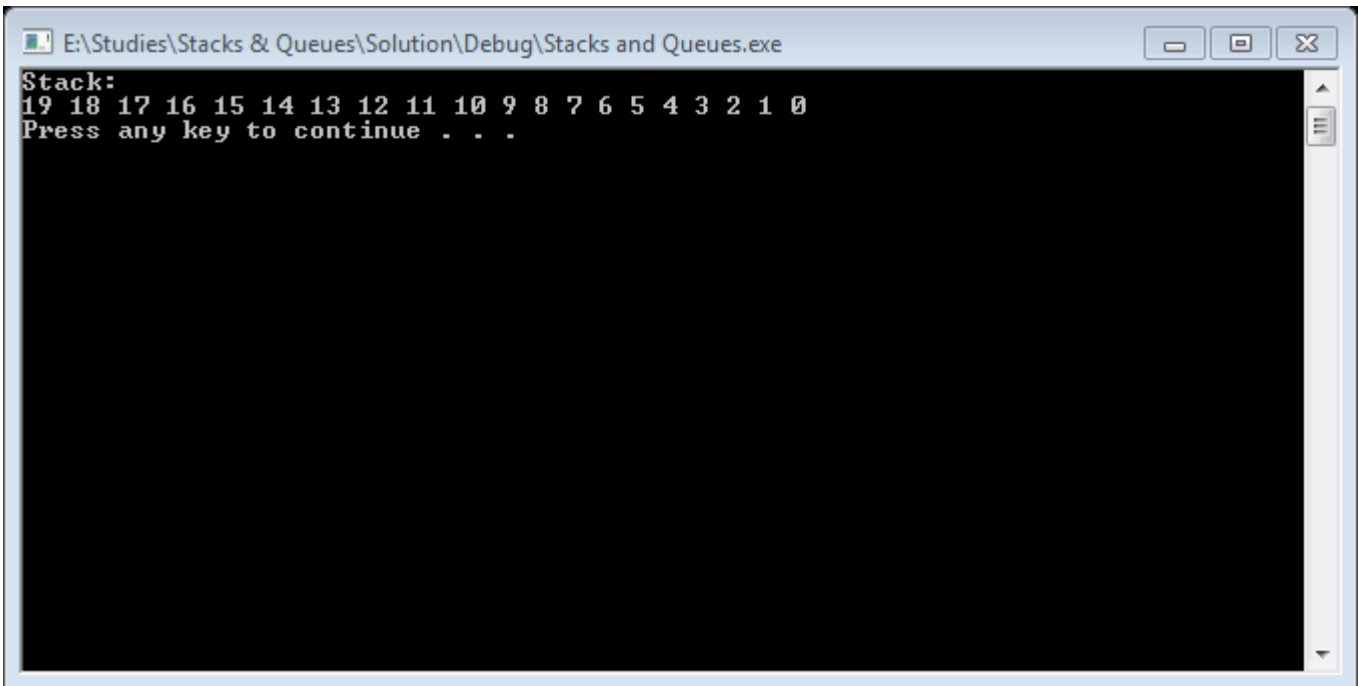
Implementing a Stack in C++ - Function hierarchy

- main()
 - push()
 - pop()
 - nodePop()
 - ~Stack()
 - nodePop()

Implementing a Stack in C++ - Pseudocode example on how to use this Stack

- Create Stack<int> object as MyStack
- For loop (intCount = 0, iterate 20 times, increment intCount)
 - Call MyStack.push(intCount)
- Try
 - While loop (true)
 - Output MyStack.pop()
- Catch
 - Output end line

Implementing a Stack in C++ - Example output

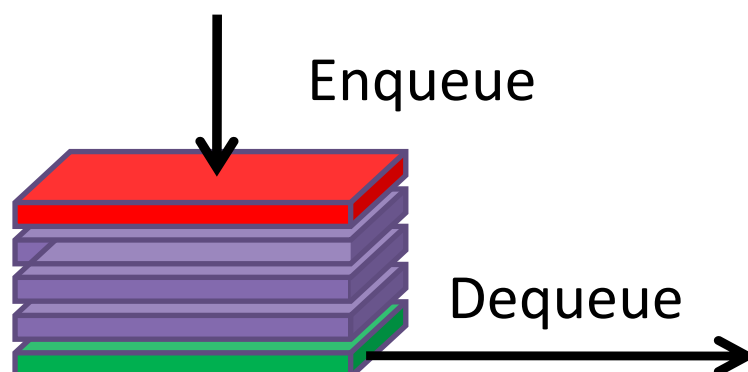


```
E:\Studies\Stacks & Queues\Solution\Debug\Stacks and Queues.exe
Stack:
19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Press any key to continue . . .
```

As evidenced, this structure is operating as LIFO. The numbers were pushed into the Stack starting from 0 and ending in 19, yet 19 was the first number to be popped out and 0 was the last number in the Stack.

The Queue

A Queue memory structure is a First In First Out (FIFO) queue. When data is added to memory, it is placed on top of the queue as it will be the last piece of data to be recalled. This behaviour is governed by two operations; enqueue and dequeue.



Like a Stack, the Queue is also simple to implement, but requires a little more resource overhead to operation. It is suited for activities such as scheduling.

Implementing a Queue in C++ - Theory

Compared to a Stack, a Queue has to make changes both sides of the Node chain since it adds Nodes to the top and removes Nodes from the bottom. So when a new Node is added to the Queue, both this new Node and the previous Node need to know they are now connected. Back in the Stack, all the code needed to know is what the next Node underneath was because Nodes were added and removed from the same place (the top).

The Queue approach requires a double linked list. With repurposing the Node class before, all that needs to be done is add another pointer for linking the chain up. This pointer is needed to prevent a complete rewrite of the code since the alternative is to reverse the order of the pointers. In a sense, this approach is a trade off since the extra pointer is worth 64 bits of memory but saves development time.

The time saving also requires that the Stack's push method be reused in the Queue's enqueue method. Therefore, the new Queue class inherits from the Stack class. As aforementioned, the Node class needs an extra pointer. This approach does allow the usage of the Stack and the Queue side by side, which is good for demonstrative purposes.

Implementing a Queue in C++ - Classes

- Node
 - Same as before, except one new public member to be added:
 - Node* **ptrPrevious**: pointer to the memory location of the previous Node in the chain
- Queue : inherits protected Stack<T> class
 - Purpose: represents the Queue memory structure
 - **Private members**
 - Node<T>* **ptrFront**: pointer to the memory location of the current front Node in the chain
 - **Public methods**
 - **Queue(void)**: constructor of Queue class; sets pointers as null pointers
 - **~Queue(void)**: destructor of Queue class; same as Stack class destructor
 - virtual void **enqueue(T)**: uses Stack class's push to create a new Node and add it to the chain whilst setting the new pointer to the previous Node as the top of the chain
 - virtual Node<T>* **nodeDequeue(void)**: removes references to Node from the chain and returns the severed Node
 - virtual T **dequeue(void)**: deletes the severed node and returns its value

Implementing a Queue in C++ - Function hierarchy

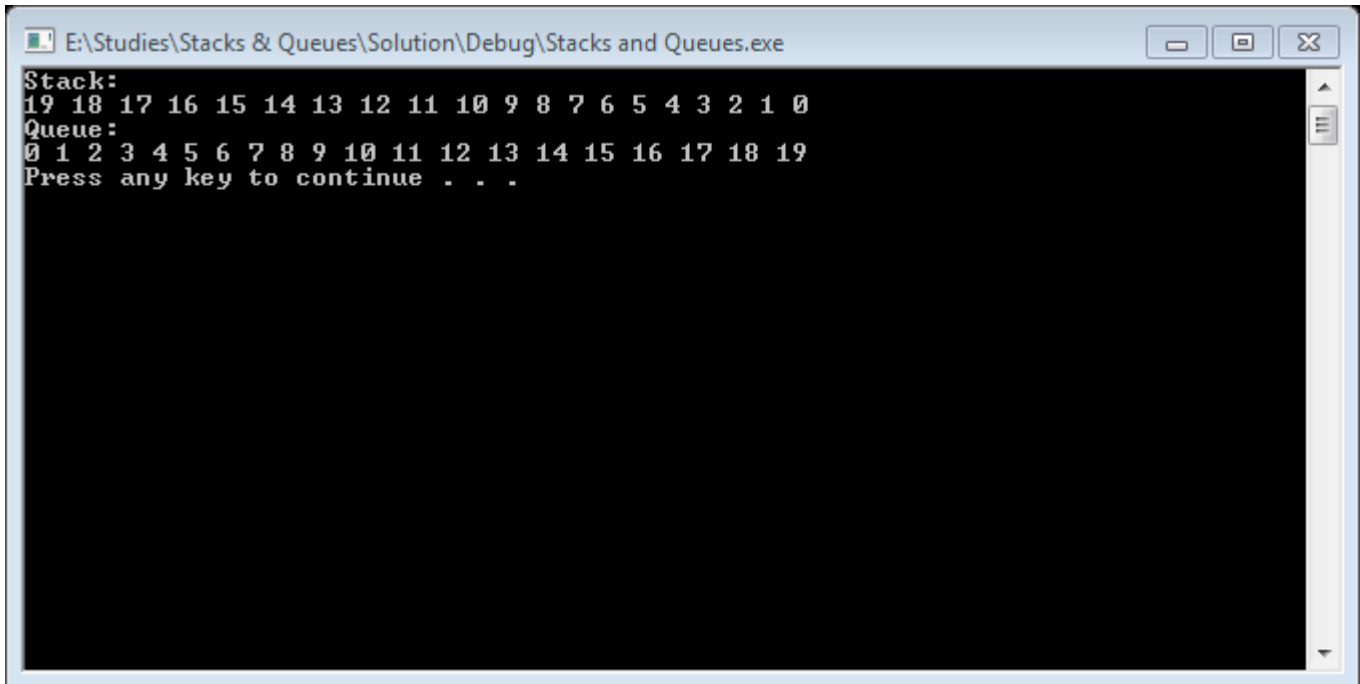
- main()
 - enqueue()
 - push()
 - dequeue()
 - nodeDequeue()
 - ~Queue()
 - nodeDequeue()

Implementing a Queue in C++ - Pseudocode example on how to use this Queue

- Create Queue<int> object as MyQueue
- For loop (intCount = 0, iterate 20 times, increment intCount)
 - Call MyQueue.enqueue(intCount)
- Try
 - While loop (true)
 - Output MyQueue.dequeue()
- Catch
 - Output end line

Implementing a Queue in C++ - Example output

STACKS & QUEUES C++ STUDY



```
E:\Studies\Stacks & Queues\Solution\Debug\Stacks and Queues.exe
Stack:
19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Queue:
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
Press any key to continue . . .
```

This output is generated by using the pseudocode shown above. The code for using the Stack was left in to demonstrate the difference in Node removal techniques between the two memory structures. Even though both the Stack and Queue use basically the same technique for Node creation, the Queue is clearly demonstrating that it is removing the Nodes from the bottom instead of the top, producing a numerically-ascending output.