**Department of Computer Science**

**Forman Christian College University**

**COMP360: Introduction to AI**

**Fall 2021**



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Task 1 (5)** | **Task 2 (5)** | **Task 3 (5)** | **Task 4 (5)** | **Total (20)** |
|  |  |  |  |  |

**Lab 1: FIFO and LIFO Implementation**

**Lab Instruction:**

* Get your attendance marked before leaving the classroom.
* This is an individual Lab assignment. Each student must submit their own work.
* Plagiarism will not be tolerated in any case.

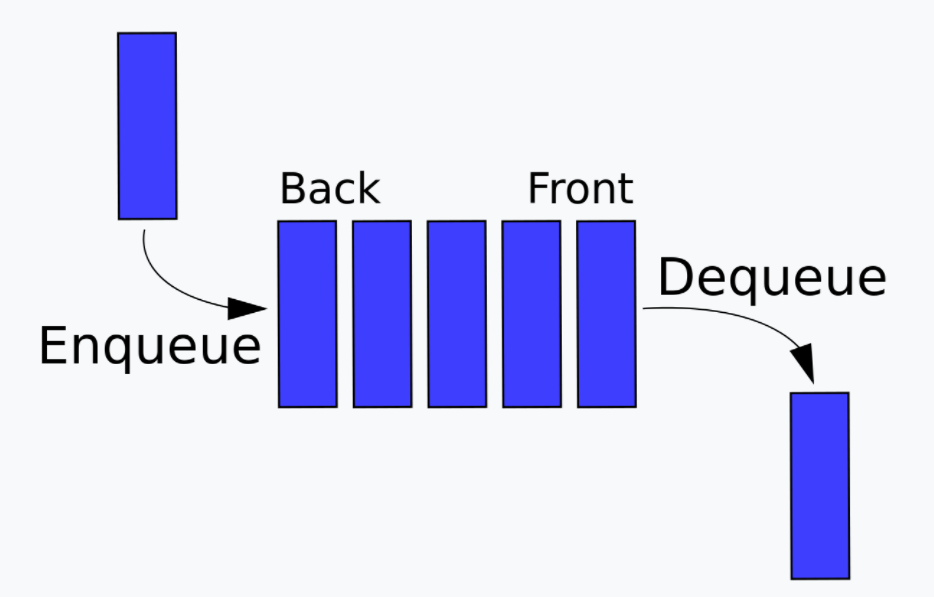
**FIFO Queue:**

**Introduction:**

A FIFO Queue is a linear First-In-First-Out data structure. In this data structure, first element is added to the queue and the same is the first to be removed.

A queue supports the following standard operations:

1. enqueue: Inserts an element at the rear (right side) of the queue.
2. dequeue: Removes the element from the front (left side) of the queue and returns it.
3. peek: Returns the element at the front of the queue without removing it.
4. isEmpty: Checks whether the queue is empty.
5. size: Returns the total number of elements present in the queue.



**Task 1: Implementation of the FIFO Queue:**

Observe the following ***FIFO\_Queue*** Class it contains the variables necessary for the implementation of the FIFO Queue. You need to complete the empty functions in a way that when you run those functions they should give the correct output.

|  |  |
| --- | --- |
| # Custom queue implementation in Python  class FIFO\_Queue:      # Initialize queue      def \_\_init\_\_(self, size=1000):  # list to store queue elements          self.q = [None] \* size  # maximum capacity of the queue  self.capacity = size  # front points to the front element in the queue  self.front = 0  # rear points to the last element in the queue    self.rear = -1  # current size of the queue  self.count = 0  # Function to dequeue the front element      def dequeue(self):          # check for queue underflow          if self.isEmpty():              print('Queue Underflow!! Terminating process.')              exit(-1)          x = self.q[self.front]          print('Removing element…', x)          self.front = (self.front + 1) % self.capacity          self.count = self.count - 1          return x      # Function to check if the queue is empty or not      def isEmpty(self): | # Function to add an element to the queue      def enqueue(self, value):          # check for queue overflow          if self.isFull():              print('Overflow!! Terminating process.')              exit(-1)          print('Inserting element…', value)          self.rear = (self.rear + 1) % self.capacity          self.q[self.rear] = value          self.count = self.count + 1    # Function to return the front element of the queue      def peek(self):          # Function to return the size of the queue      def size(self):        # Function to check if the queue is full or not      def isFull(self): |

**Task 2:** Run the following function main() and record the observations and note down the exact results under the output.

|  |  |
| --- | --- |
| if \_\_name\_\_ == '\_\_main\_\_':        # create a queue of capacity 5      q = Queue(5)        q.enqueue(1)      q.enqueue(2)      q.enqueue(3)        print('The queue size is', q.size())      print('The front element is', q.peek())      q.dequeue()      print('The front element is', q.peek())        q.dequeue()      q.dequeue()        if q.isEmpty():          print('The queue is empty')      else:          print('The queue is not empty') | **OUTPUT:** |

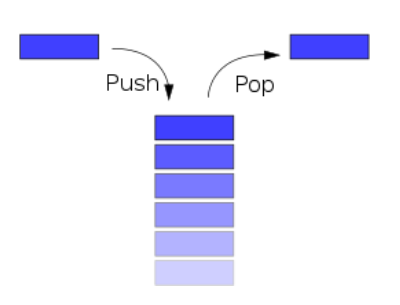
**LIFO Stack:**

**Introduction:**

A stack is a linear data structure that follows the LIFO (Last–In, First–Out) order, i.e., items can be inserted or removed only at one end of it.

The stack supports the following standard operations:

* push: Pushes an item at the top of the stack.
* pop: Remove and return the item from the top of the stack.
* peek: Returns the item at the top of the stack without removing it.
* size: Returns the total number of items in the stack.
* isEmpty: Checks whether the stack is empty.
* isFull: Checks whether the stack is full.



**Task 3: Implementation of the LIFO Stack**

Observe the following ***LIFO\_Stack*** class. You need to complete the empty functions in a way that when you run those functions they should give the correct output.

|  |  |
| --- | --- |
| # Custom stack implementation in Python  class LIFO\_Stack:      # Constructor to initialize the stack      def \_\_init\_\_(self, size):          self.arr = [None] \* size          self.capacity = size          self.top = -1        # Function to add an element `val` to the stack      def push(self, val):          if self.isFull():              print('Stack Overflow!! Calling exit()…')              exit(-1)            print(f'Inserting {val} into the stack…')          self.top = self.top + 1          self.arr[self.top] = val        # Function to pop a top element from the stack      def pop(self):          # check for stack underflow          if self.isEmpty():              print('Stack Underflow!! Calling exit()…')              exit(-1)            print(f'Removing {self.peek()} from the stack')            # decrease stack size by 1 and (optionally) return the popped element          top = self.arr[self.top]          self.top = self.top - 1 | # Function to return the top element of the stack      def peek(self):        # Function to return the size of the stack      def size(self):          # Function to check if the stack is empty or not      def isEmpty(self):          # Function to check if the stack is full or not      def isFull(self): |

**Task 4:** Run the following function main(), observe the output, and note it down under the output.

|  |  |
| --- | --- |
| if \_\_name\_\_ == '\_\_main\_\_':        stack = Stack(3)        stack.push(1)       # Inserting 1 in the stack      stack.push(2)       # Inserting 2 in the stack        stack.pop()         # removing the top element (2)      stack.pop()         # removing the top element (1)        stack.push(3)       # Inserting 3 in the stack        print('Top element is', stack.peek())      print('The stack size is', stack.size())        stack.pop()         # removing the top element (3)        # check if the stack is empty      if stack.isEmpty():          print('The stack is empty')      else:          print('The stack is not empty') | **Output:** |

***Note: This is an individual Lab and Groups are not allowed. No need to search on internet treat it like a lab exam.******There could be errors in the above code you need to fix them in order to perform successful execution.***