	PUNE INSTITUTE OF COMPUTER TECHNOLOGY PUNE - 411043	
	Department of Electronics & Telecommunication	
	ASSESMENT YEAR: 2020-2021	CLASS: SE 5
	SUBJECT: DATA STRUCTURES	
EXPT No: 3	LAB Ref: SE/2020-21/	Starting date:
	Roll No: 22119	Submission date: 12/11/2020
Title:	Stack and Queue Implementations (Array)	
Prerequisites:	DEV C++ IDE	
	Knowledge about arrays	
	Knowledge about handling stack and queue operations	
Objectives:	1)To learn the representation of Stack and Queue data structure using Array.	
	2)Verify various methods of handling data elements by following rules of data structure operations.	
	3)Write user defined functions for push, pop, enque, deque for Stack and Queue handling.	

Theory:

Stack:

In the pushdown stacks only two operations are allowed: push the item into the stack, and pop the item out of the stack. A stack is a limited access data structure - elements can be added and removed from the stack only at the top. push adds an item to the top of the stack, pop removes the item from the top. A helpful analogy is to think of a stack of books; you can remove only the top book, also you can add a new book on the top. Stack is a container of objects that are inserted and removed according to the lst in first out principle (LIFO)

Queue:

An excellent example of a queue is a line of students in the food court of the UC. New additions to a line made to the back of the queue, while removal (or serving) happens in the front. In the queue only two operations are allowed enqueue and dequeue. Enqueue means to insert an item into the back of the queue, dequeue means removing the front item. The picture demonstrates the FIFO access. The difference between stacks and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added. Queue is a container of objects (linear collection) that are inserted and removed according to first in first out principle (FIFO)

Push:

Push enters an item on the stack

Pop:

Pop retrieves an item, moving the rest of the items in the stack up one level

Insert:

To insert an element in the queue

Delete:

To delete elements from queue

Algorithm

STACK OPERATIONS:

CREATE AN EMPTY STACK:

Step 1: start
Step 2: include all header files which are used in the program and define a constant size with specific value
Step 3: declare all the functions used in stack implementation
Step 4: Create a one-dimensional array with fixed size (int stack [SIZE])
Step 5: Define a integer variable 'top' and initialize with '-1'. (int top = -1)
Step 6: In main method, display menu with list of operations and make suitable function calls to perform operation selected by the user on the stack.
Step 7: Stop

PUSH:

Step 1: start
Step 2: Check whether stack is FULL. (top == SIZE-1)
Step 3: If it is FULL, then display "Stack is FULL" and terminate the function.
Step 4: If it is NOT FULL, then increment top value by one (top++**)** and set stack[top] to value (stack[top] = value).
Step 5: stop

POP:

Step 1: start
Step 2: If it is EMPTY, then display "Stack is EMPTY" and terminate the function.
Step 3: If it is NOT EMPTY, then delete stack[top] and decrement top value by one (top--).
Step 4: stop

DISPLAY:

Step 1: start
Step 2: Check whether stack is EMPTY. (top == -1)
Step 3: If it is EMPTY, then display "Stack is EMPTY" and terminate the function.
Step 4: If it is NOT EMPTY, then define a variable 'i' and initialize with top. Display stack[i] value and decrement i value by one (i--).
Step 5: Repeat above step until **i** value becomes '0'.
Step 6: stop

QUEUE OPERATIONS:

CREATE AN EMPTY QUEUE:

Step 1: start
Step 2: Include all the header files which are used in the program and define a constant 'SIZE' with specific value.
Step 3: Declare all the user defined functions which are used in queue implementation.
Step 4: Create a one dimensional array with above defined SIZE (int queue[SIZE])

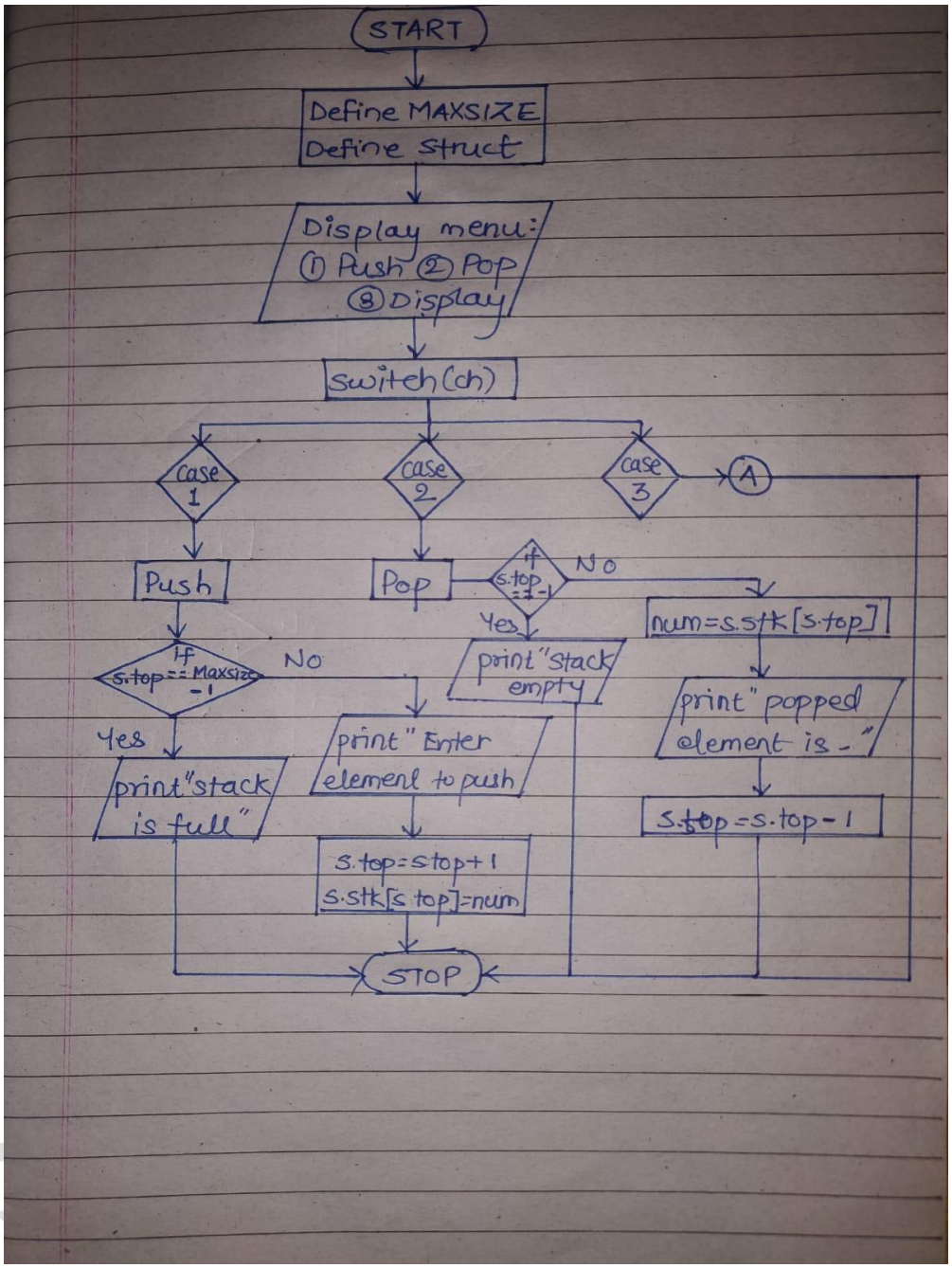
Step 5: Define two integer variables 'front' and 'rear' and initialize both with '-1'. (int front = -1, rear = -1)
Step 6: Then implement main method by displaying menu of operations list and make suitable function calls to perform operation selected by the user on queue.
Step 7: stop

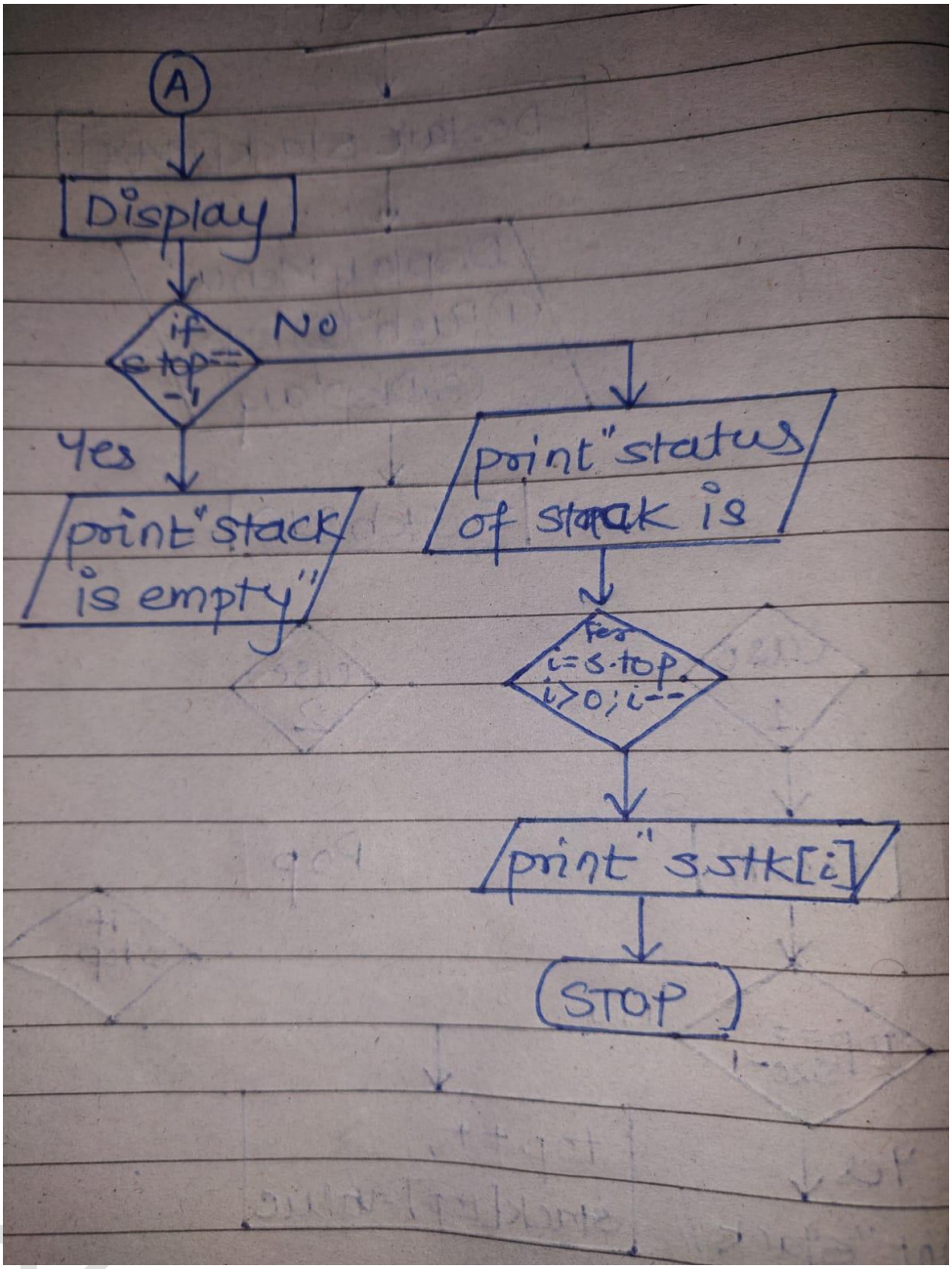
ENQUEUE/INSERT:
Step 1: start
Step 2: Check whether queue is FULL. (rear == SIZE-1)
Step 3: If it is FULL, then display "Queue is FULL" and terminate the function.
Step 4: If it is NOT FULL, then increment rear value by one (rear++) and set queue[rear] = value.
Step 5: stop

DEQUEUE/DELETE:
Step 1: start
Step 2: Check whether queue is EMPTY. (front == rear)
Step 3: If it is EMPTY, then display "Queue is EMPTY" and terminate the function.
Step 4: If it is NOT EMPTY, then increment the front value by one (front ++). Then display queue[front] as deleted element. Then check whether both front and rear are equal (front == rear), if it TRUE, then set both front and rear to '-1' (front = rear = -1).
Step 5: stop

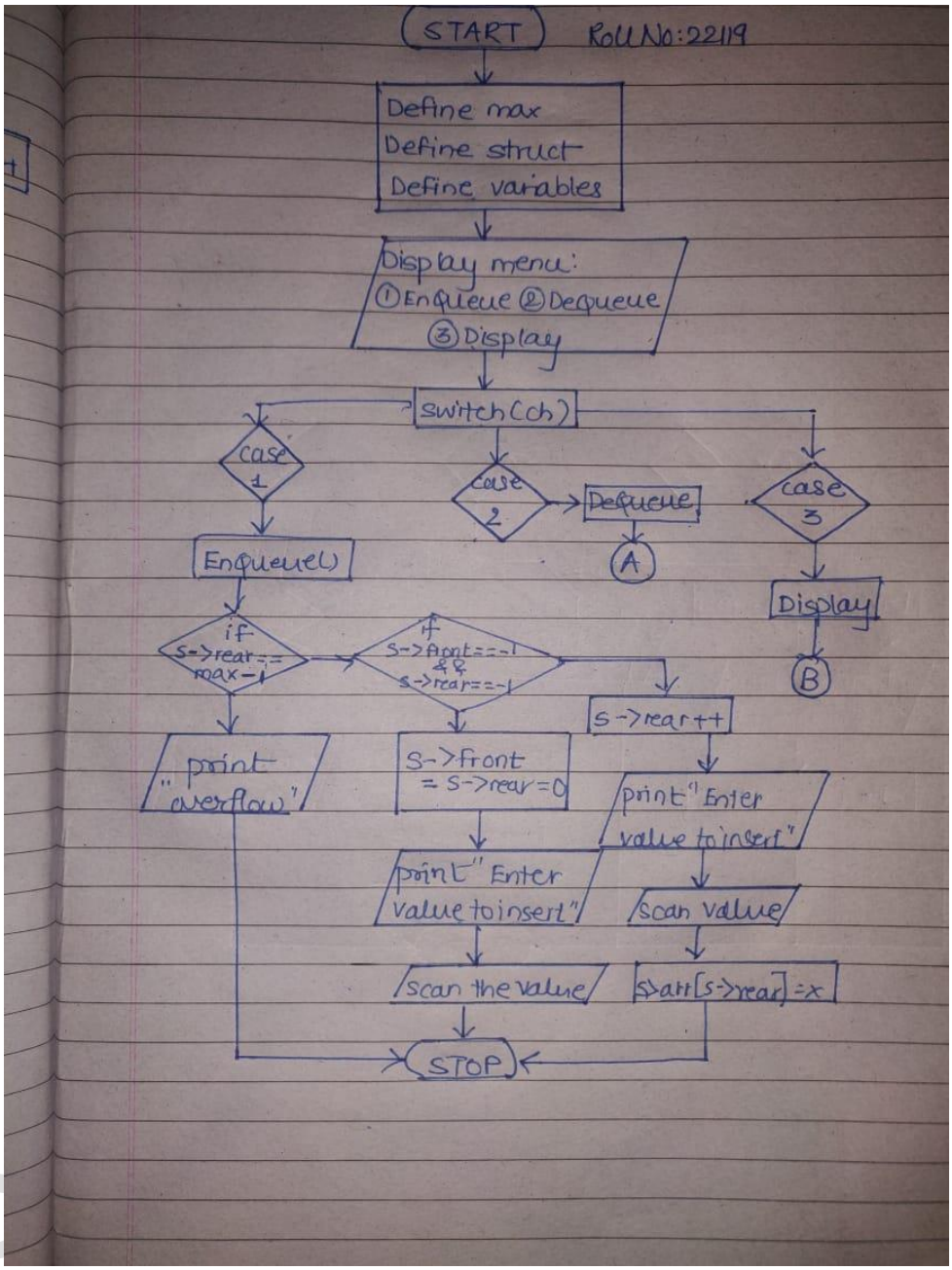
DISPLAY:
Step 1: start
Step 2: Check whether queue is EMPTY. (front == rear)
Step 3: If it is EMPTY, then display "Queue is EMPTY" and terminate the function.
Step 4: If it is NOT EMPTY, then define an integer variable 'i' and set 'i = front+1'.
Step 5: Display 'queue[i]' value and increment 'i' value by one (i++). Repeat the same until 'i' value reaches to rear (i <= rear)
Step 6: stop

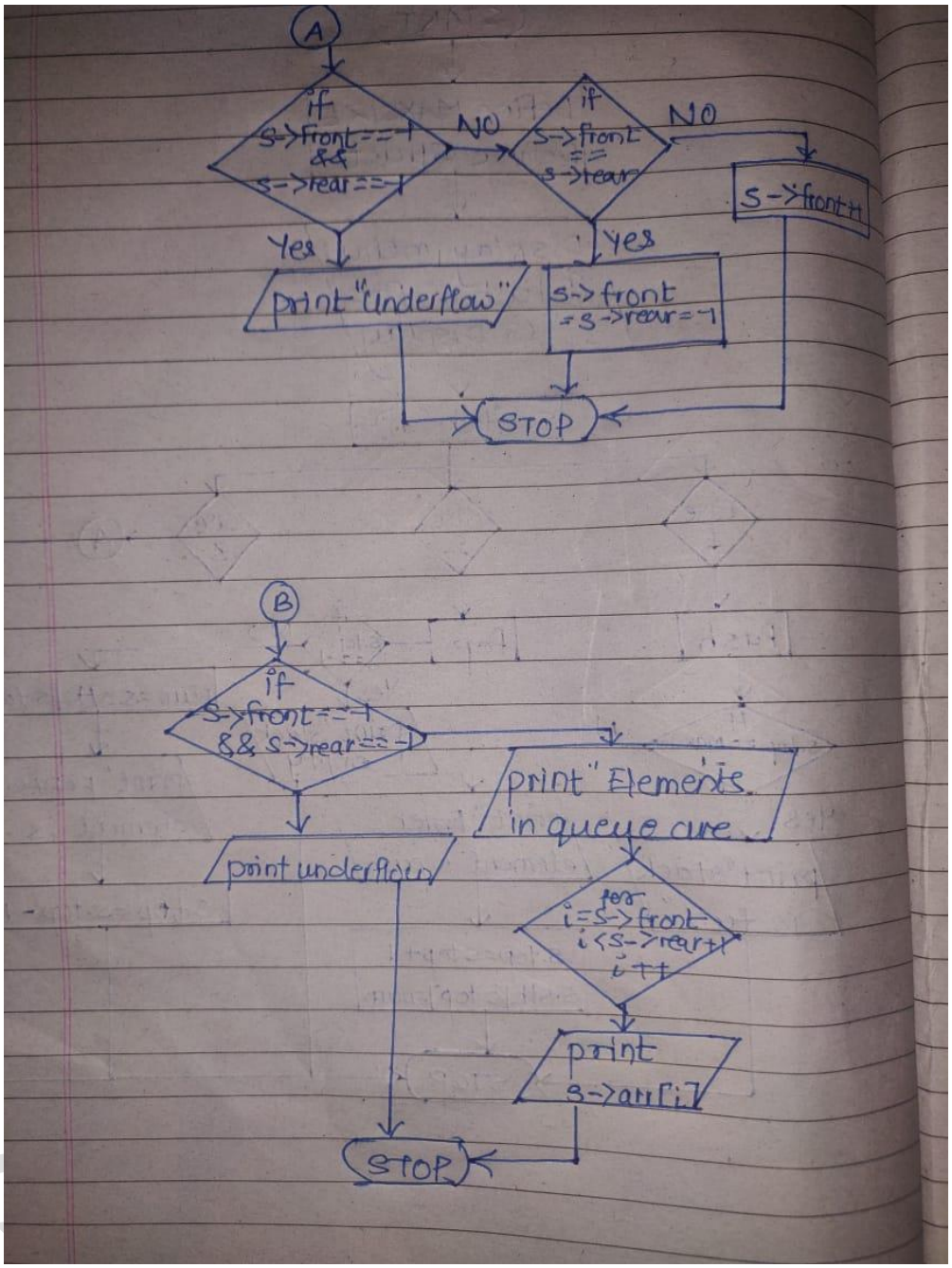
Flow-chart	**STACK OPERATIONS:**




A
Display
if s.top == -1
No
Yes
print "stack is empty"
print "status of stack is
for i=s.top; i>0; i--
print" s.stk[i]
STOP

QUEUE OPERATIONS:




A
if
S->front==-1
&&
S->rear==-1
NO
if
S->front
==
S->rear
NO
S->front++
Yes
Yes
print "underflow"
S->front
=S->rear=-1
STOP
B
if
S->front==-1
&& S->rear==-1
print "Elements
in queue are"
print underflow
for
i=S->front
i<S->rear+1
i++
print
S->arr[i]
STOP

ERROR	No errors occurred
REMEDY	No remedy applied
CONCLUSION:	
	1. Learnt to implement program for stack and queue 2. Understood about FIFO & LIFO 3. Verified various methods like push, pop, enqueue, dequeue used in stack and Queue operations
REFERENCES:	
	1) Ellis Horowitz, SartajSahani, "Fundamentals of Data Structures", Galgotia books. 2) E Balgurusamy, "Programming in ANSI C", Tata McGraw-Hill, 3rd Edition. 3) YashvantKanetkar-Understanding Pointers in C BPB publications 3rd Edition.

Continuous Assessment			**Assessed By**
RPP (5)	**ARR (5)**	**Total (10)**	**Signature:**
			Date: