	PUNE INSTITUTE OF COMPUTER TECHNOLOGY PUNE - 411043	
	Department of Electronics & Telecommunication	
	ASSESMENT YEAR: 2020-2021	CLASS: SE 5
	SUBJECT: DATA STRUCTURES	
EXPT No: 7	LAB Ref: SE/2020-21/	Starting date: 19/11/2020
	Roll No: 22119	Submission date:26/11/2020
Title:	Stack and Queue Implementation (LL)	
Prerequisites:	DEVC++ IDE	
	Linked list knowledge	
Objectives:	To learn the representation of Stack and Queue data structure using linked list.	
	Verify various methods of handling data elements by following rules of data	
	structure operations.	
	Write user defined functions for push, pop, enque, deque for Stack and Queue handling.	
Theory:		

Linked list:
a linked list consists of nodes where each node contains a data field and a reference(link) to the next node in the list.
There are 3 different implementations of Linked List available, they are:

1. Singly Linked List-Singly linked lists contain nodes which have a data part as well as an address part i.e. next, which points to the next node in the sequence of nodes. The operations we can perform on singly linked lists are insertion, deletion and traversal.
2. Doubly Linked List-In a doubly linked list, each node contains a data part and two addresses, one for the previous node and one for the next node.
3. Circular Linked List-In circular linked list the last node of the list holds the address of the first node hence forming a circular chain.

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)

	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 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) Enqueue: To insert an element in the queue Dequeue: To delete elements from queue
Algorithm	**STACK OPERATIONS:** 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 PUSH: Step 1: start Step 2: struct Node *newNode; Set newNode = (struct Node*)malloc(sizeof(struct Node)); Step 3: set newNode->data = value Step 4: if top == NULL , newNode->next = NULL, else go to step 4 Step 5: set newNode->next = top; Set top = newNode; and print node is inserted Step 6: stop POP: Step1: start Step 2: if top == NULL, print the stack is empty and go to step 6, else step 3 Step 3: set struct Node *temp = top Step 4: the element popped

Step 5: set top = temp->next;
Set free(temp);
Step 6: stop

DISPLAY:
Step 1: start
Step 2: if top == NULL, print the stack is empty and go to step 6, else go to step 3
Step 3:print the stack
Step 4: set struct Node *temp = top
Step 5: while temp->next != NULL, print the elements to display stack and temp = temp -> next
Step 6: stop

QUEUE OPERATIONS:

CREATE AN EMPTY queue:
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 queue implementation
Step 4: create "create" function
Step 5: front=rear=null

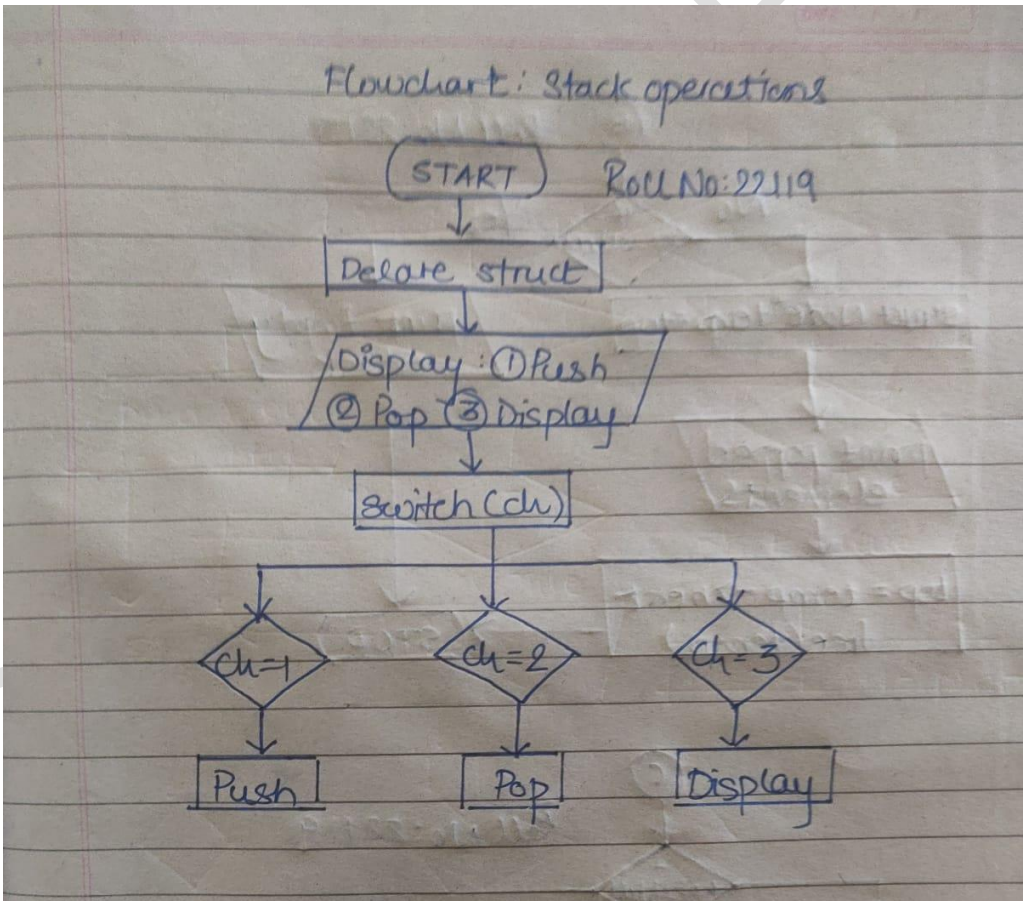
ENQUEUE:
Step 1: start
Step 2: check if rear==null, if yes go to step 3 else go to step 4
Step 3: set rear = (struct node *) malloc(1*sizeof(struct node));
set rear->ptr = NULL;
set rear->info = data;
set front = rear;
step 4: set temp=(struct node *)malloc(1*sizeof(struct node));
set rear->ptr = temp;
set temp->info = data;
set temp->ptr = NULL;
set rear = temp;
step 5: increment count
step 6: stop

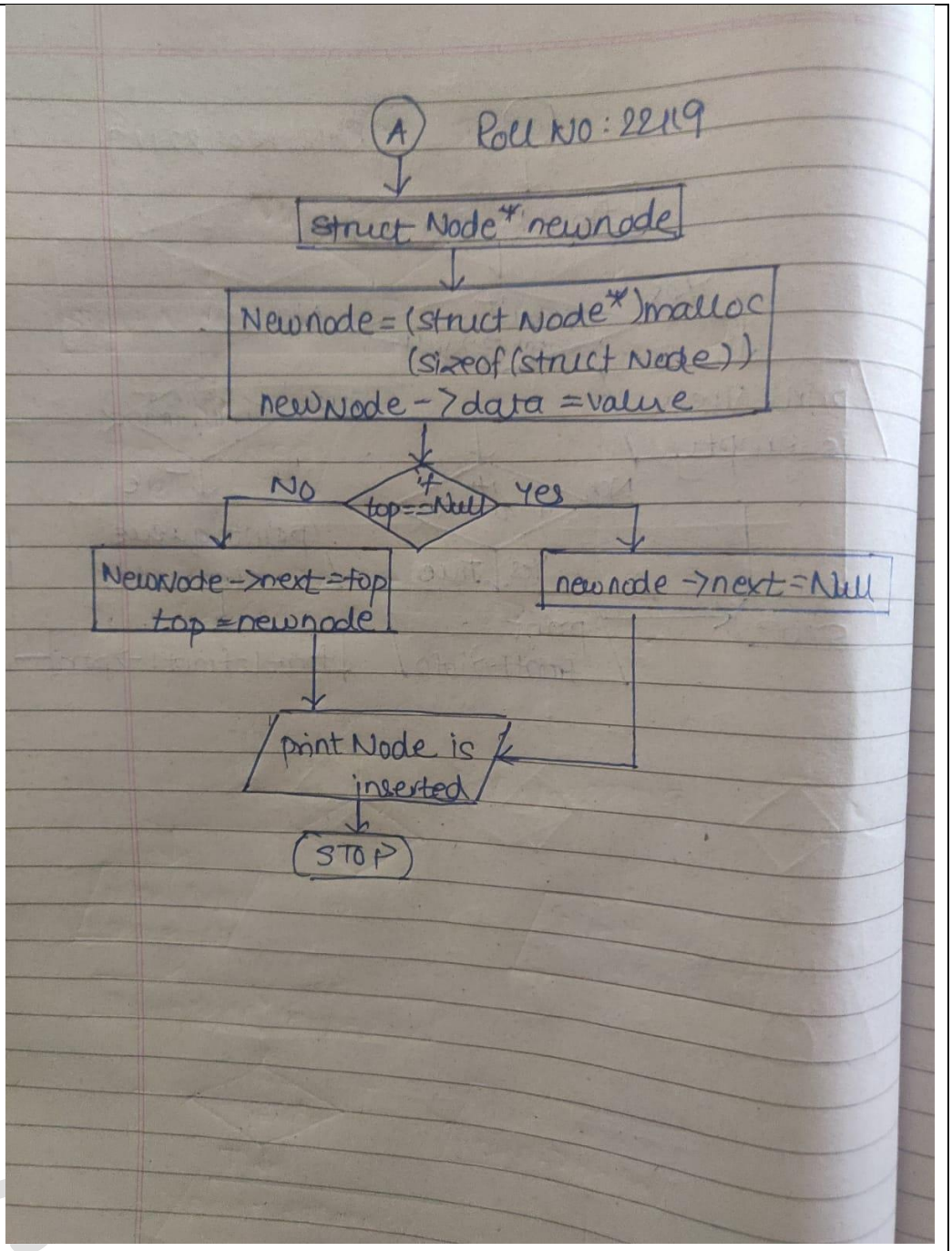
DEQUEUE:
Step 1: start
Step 2: set front1= front
Step 3: check if front 1=null, if yes print empty queue else go to step 4
Step 4: if front1->ptr != NULL go to step 5 else go to step 7
Step 5: set front1 = front1->ptr;
Print the dequeued value
Set free(front);
set front = front1;

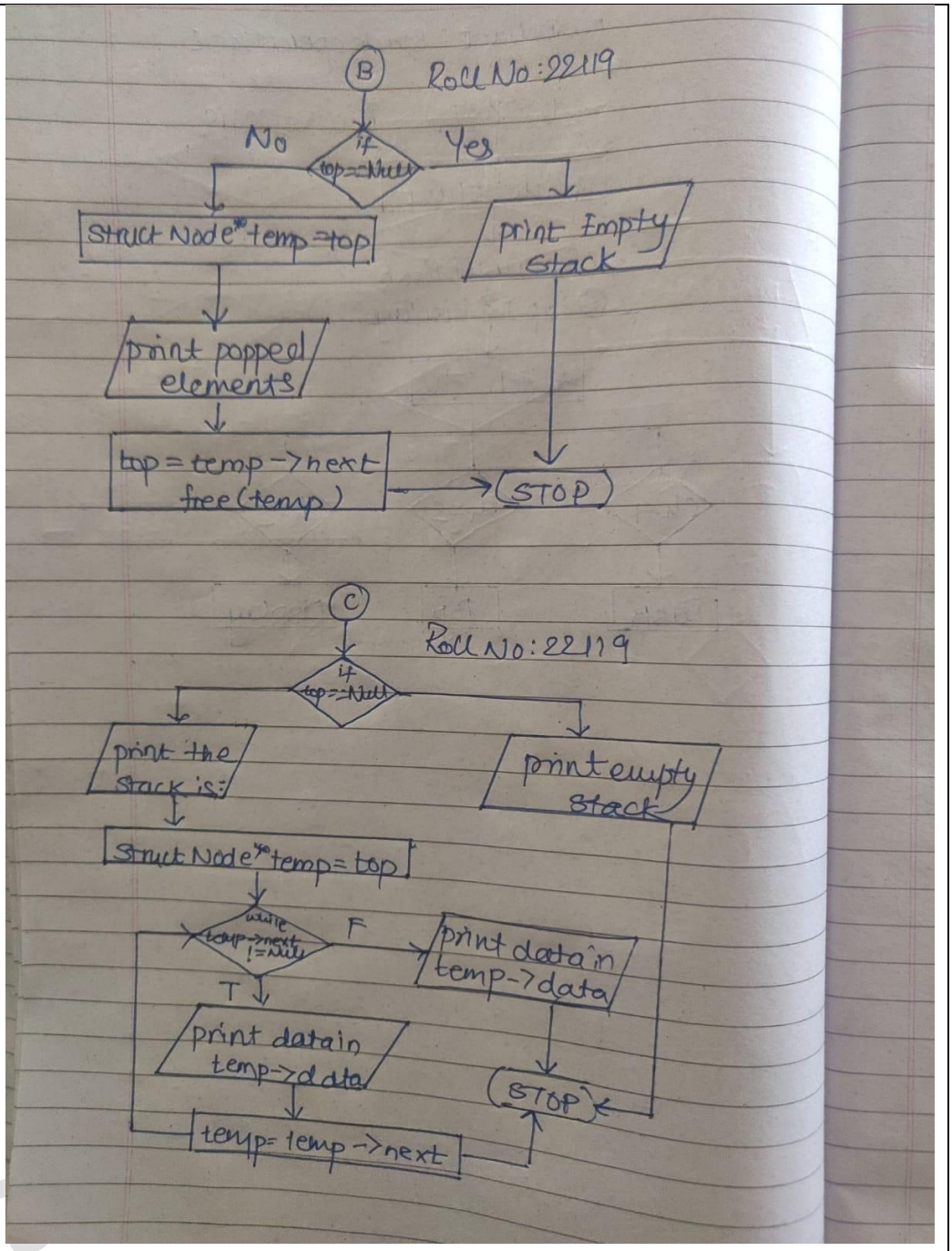
step 6: print the dequeued value
set free(front);
set front = NULL;
set rear = NULL
step 7: decrement count
step 8: stop

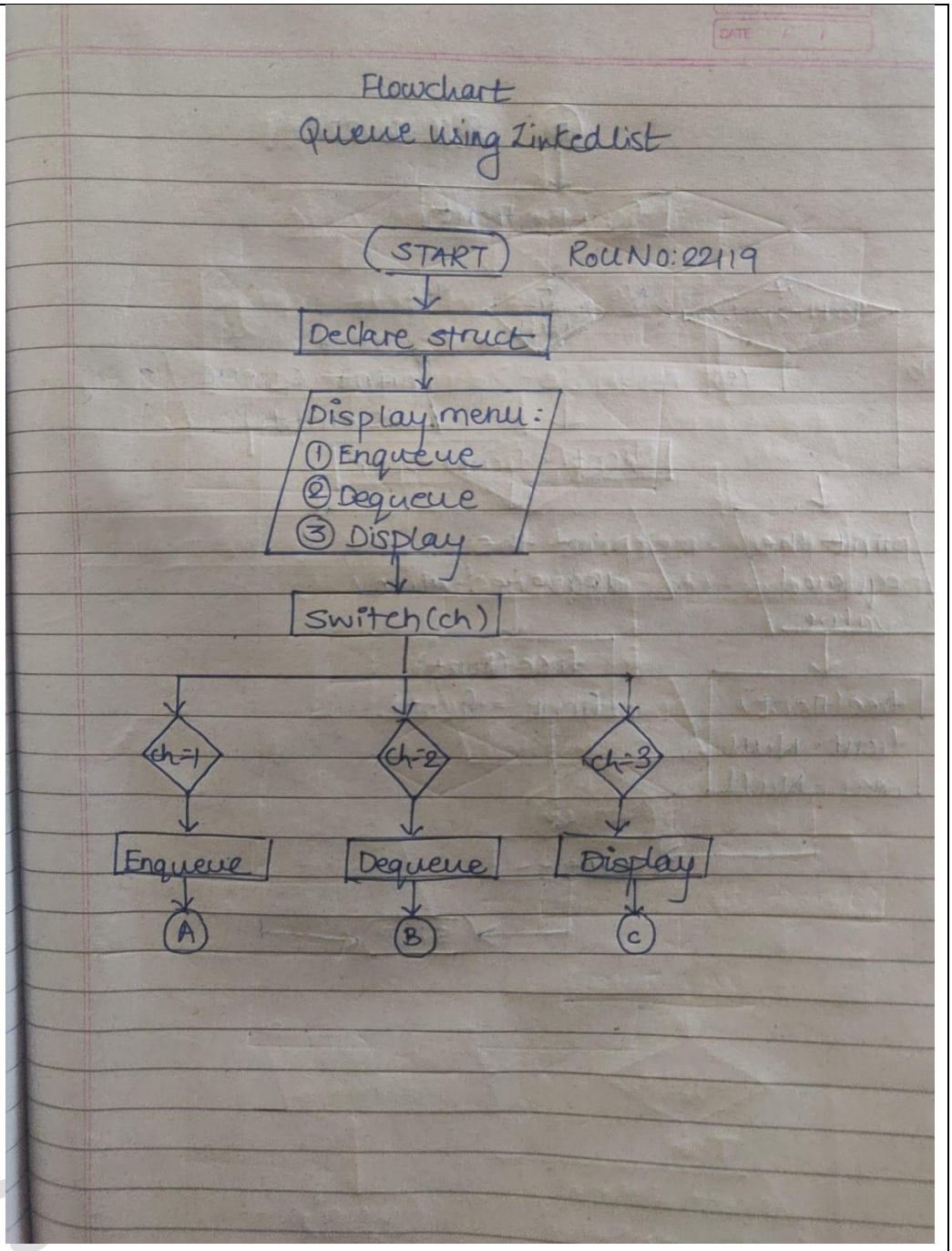
DISPLAY:
step 1: start
step 2: set front1 = front
step 3: check if front1==null and rear == null, if yes print empty queue
step 4: while front1 is not equal to rear, print the values
step 5: front1 = front1->ptr
step 6: if front1 == rear print queue
step 7: stop

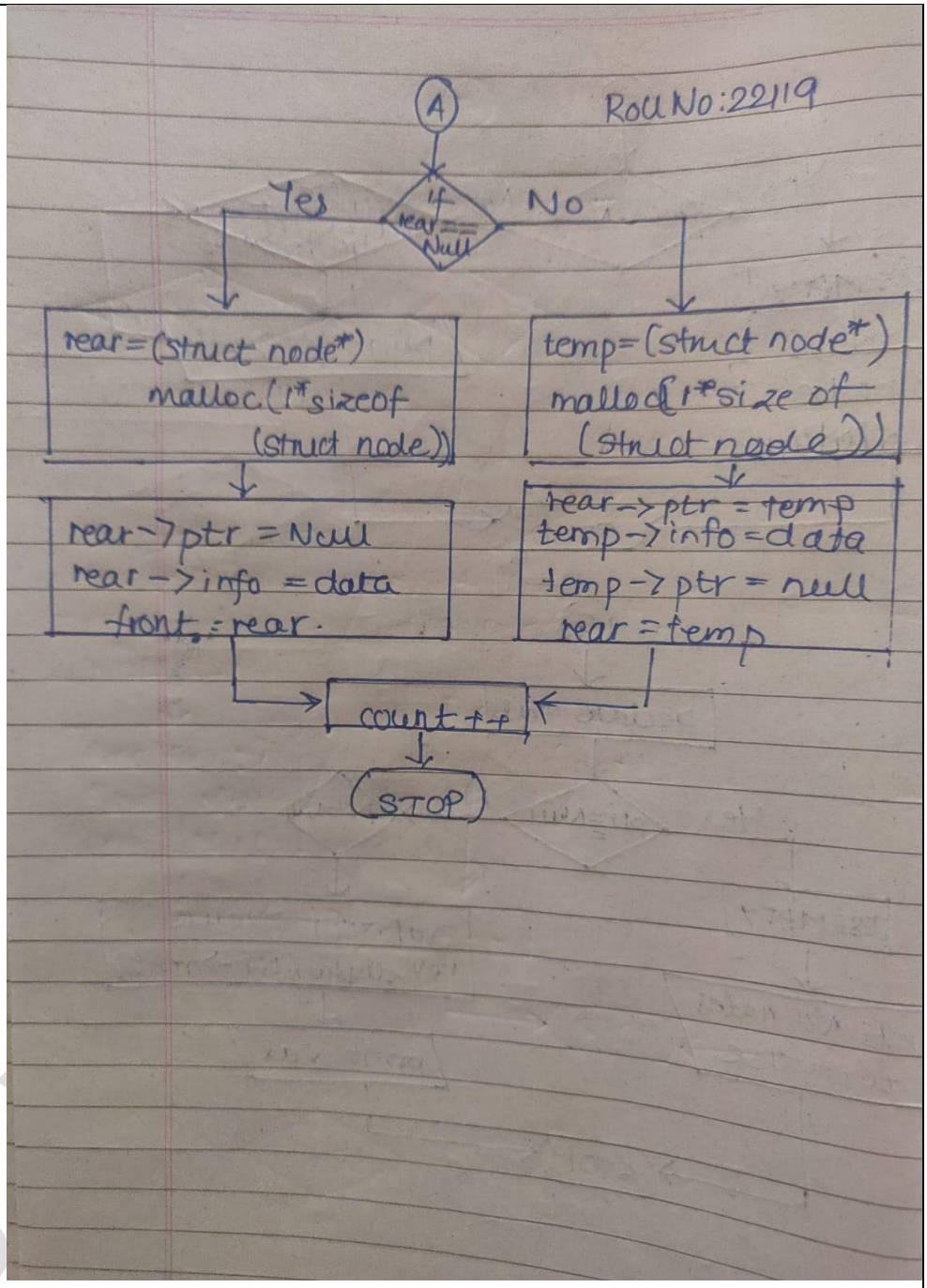
Flow-chart

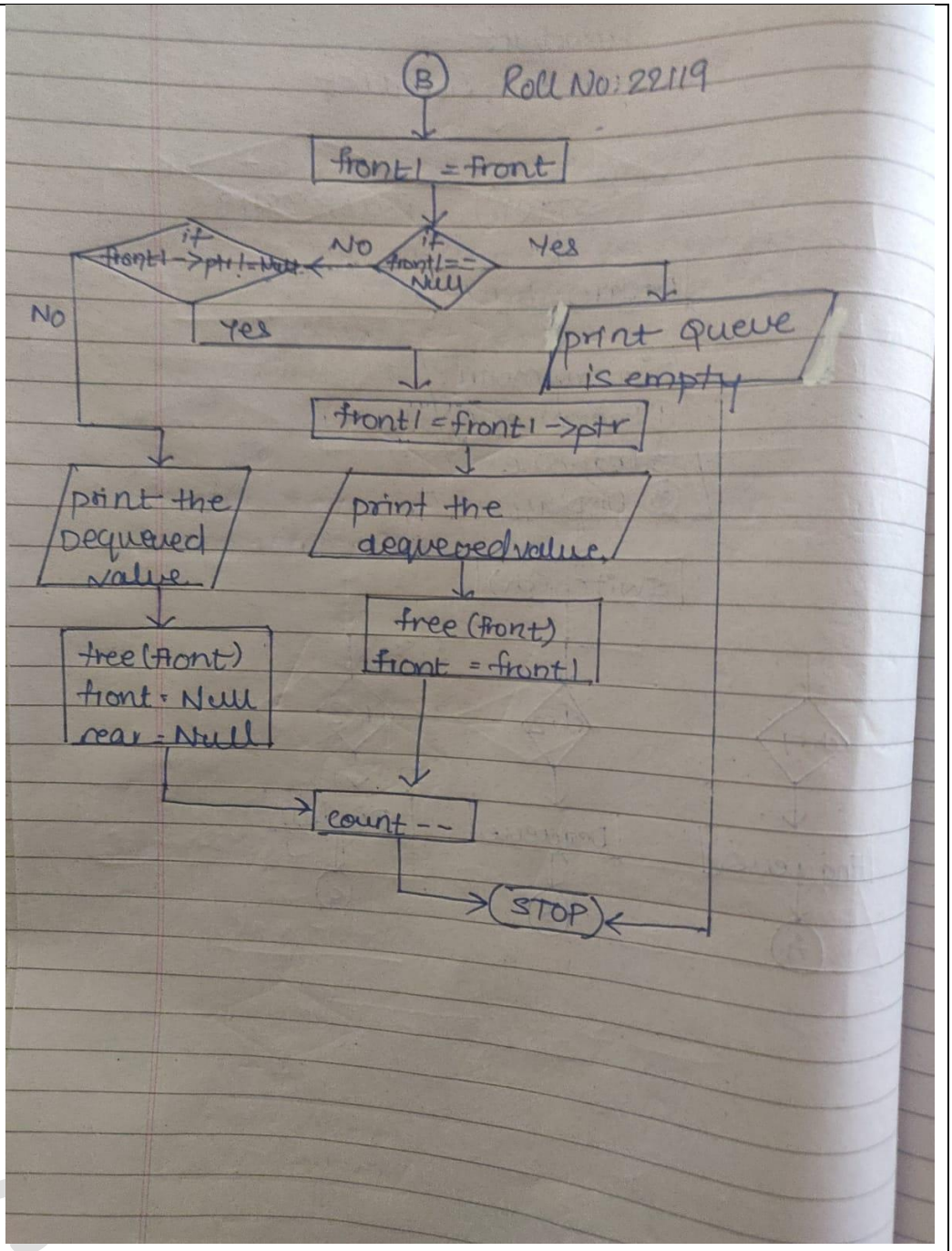


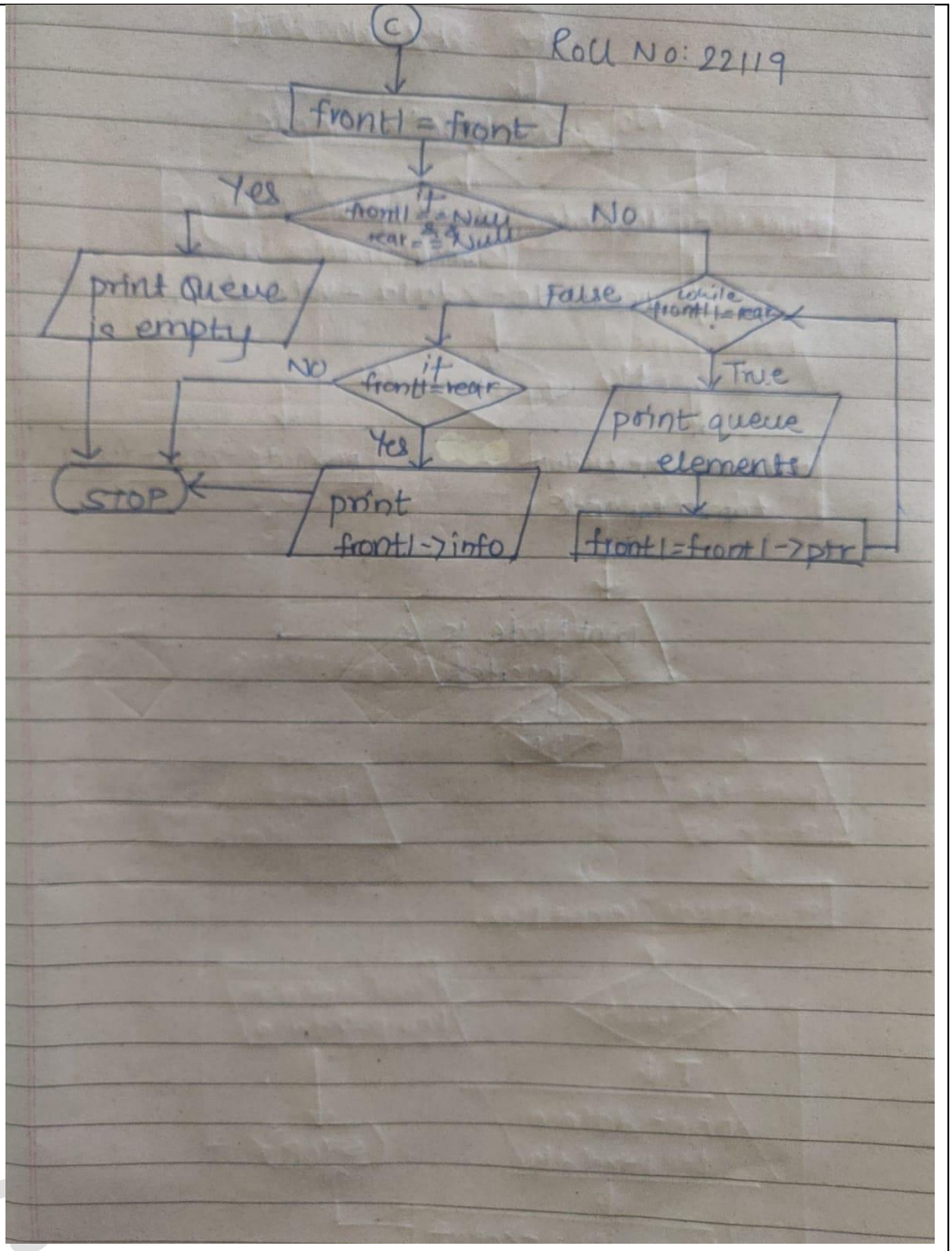

A
Roll NO : 2219
Struct Node* newnode
Newnode = (struct Node*) malloc
(sizeof (struct Node))
newNode -> data = value
if
top == Null
No
Yes
NewNode -> next = top
top = newnode
newnode -> next = Null
print Node is
inserted
STOP


B
Roll No: 22119
if top==Null
No
Yes
Struct Node* temp = top
print Empty Stack
print popped elements
top = temp -> next
free(temp)
STOP
C
Roll No: 22119
if top==Null
print the stack is:
print empty stack
Struct Node* temp = top
while temp->next != Null
F
print data in temp->data
T
print data in temp->data
STOP
temp = temp -> next


Flowchart
Queue using Linkedlist
START
RollNo:22119
Declare struct
Display menu:
① Enqueue
② Dequeue
③ Display
Switch(ch)
ch=1
ch=2
ch=3
Enqueue
Dequeue
Display
A
B
C


A
Roll No: 22119
if rear == Null
Yes
No
rear=(struct node*) malloc(1*sizeof (struct node))
temp=(struct node*) malloc(1*size of (struct node))
rear->ptr = Null
rear->info = data
front = rear
rear->ptr = temp
temp->info = data
temp->ptr = null
rear = temp
count++
STOP


B
Roll No: 22119
front1 = front
if front1 == Null
Yes
print queue is empty
No
if front1->ptr != Null
No
yes
front1 = front1->ptr
print the dequeued value
free(front)
front = front1
print the dequeued value
free(front)
front = Null
rear = Null
count --
STOP


Roll No: 22119
front1 = front
if front1 != Null & rear != Null
Yes
No
print Queue is empty
STOP
if front1 = rear
NO
Yes
print front1->info
while front1 != rear
False
True
print queue elements
front1 = front1->ptr

ERROR	No errors occurred
REMEDY	none
CONCLUSION:	
	1) learnt to represent a stack and a queue using linked list 2) learnt to apply operations on stack and queue using linked list 3) User defined functions used to perform operations like push, pop, enqueue, dequeue, Delete, etc
REFERENCES:	
	1) Seymour Lipschutz, Data Structure with C, Schaum's Outlines, Tata McGrawHill 2) Yedidyah Langsam – Data structures using C and C++ - PHI Publications (2nd Edition). 3) Yashavant Kanetkar, Data Structures Through C, BPB Publication, 2nd Edition

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