	PUNE INSTITUTE OF COMPUTER TECHNOLOGY PUNE - 411043	
	Department of Electronics & Telecommunication	
	ASSESMENT YEAR: 2020-2021	CLASS: SE 5
	SUBJECT: DATA STRUCTURES	
EXPT No: 4	LAB Ref: SE/2020-21/	Starting date: 14/11/2020
	Roll No: 22119	Submission date: 21/11/2020
Title:	Singly Linked List Operations	

Prerequisites:	DEVC++ IDE
	Knowledge about singly linked lists and operations on it
Objectives:	Learn and implement the Singly Linked List using the concept of self-referential structure
	Apply concept of dynamic memory allocation functions to change size during runtime.
	Perform and verify various operation with data handling using linked data structure.

Theory:

A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations. The elements in a linked list are linked using pointers. Linked List is a sequence of links which contains items. Each link contains a connection to another link. Linked list can be visualized as a chain of nodes, where every node points to the next node. Linked List contains a link element called first. Each link carries a data field(s) and a link field called next. Each link is linked with its next link using its next link. Last link carries a link as null to mark the end of the list

Algorithm	1) Start 2) Ask the user for what he wants to do. 3) If the user chooses to create a single linked list call the create() function. **a) CREATE:** 1. We first allocate the struct Node* memory to a pointer using malloc. 2. If the start pointer is NULL then the newly created node will point to start pointer. 3. If not then the pointer to this new node is stored in the node which is already saved in the start pointer. 4. If the user chooses to display the linked list then call the display() function. **b) DISPLAY:** 1. If the start pointer is null then print linked list is empty. 2. Else print the info stored in the info of the start pointer and then run a while loop pointing out to the next node stored in the next field of the pointer. 3. And then print the info stores in the node. 4. If the user chooses to insert a new node in the beginning then call the insert_begin() function. **c) INSERT_BEGIN:** 1. If the start pointer is null then this node pointer will be saved as the start pointer. 2. Else the next field in the currently entered node will save the value of the start pointer. 3. If the user chooses to insert a node at the end then call the insert_end() function. **d) INSERT_END:** 1. If the start pointer is NULL then the current node is saved at start pointer. 2. Else we will run a while loop to find which node has its next field NULL and then the pointer to the new node is saved in the next field. 3. If the user chooses to insert a node at a specific position then call the insert_position() function. **e) INSERT_SPECIFIC POSITIOIN:** 1. If position u want to insert the node is 0 then the start pointer will point to this node. 2. Else run a while loop till u find the pointer of the node having position (pos-1) and let the next field of this pointer be the newly entered node and the next field of it point to the newly entered node. 3. If the user wants to delete the beginning node then call the delete_begin() function. **f) DELETE_BEGIN:** 1. Check whether list is Empty (head == NU LL) 2. If it is Empty then, display 'List is Empty' and terminate the function. 3. If it is Not Empty then, define a Node pointer 'temp' and initialize with

head.

4. Check whether list is having only one node (temp → next == NULL)
5. If it is TRUE then set head = NULL and delete temp (Setting Empty list conditions)
6. If it is FALSE then set head = temp → next, and delete temp.
7. If the user chooses to delete the end item of the linked list then call delete_end() function.

g) DELETE_END:

1. Check whether list is Empty (head == NULL)
2. If it is Empty then, display 'List is Empty!!! Deletion is not possible' and terminate the function.
3. If it is Not Empty then, check if there is only one node and then let the pointer value of that node be null(i.e start = null)
4. Else run a while loop to find the node whose next field is empty and make the next field of the previous node null.
5. And use the free() function to free the memory allocated to the node.

6. If the user chooses to delete a node at a specific position then call the delete_pos() function.

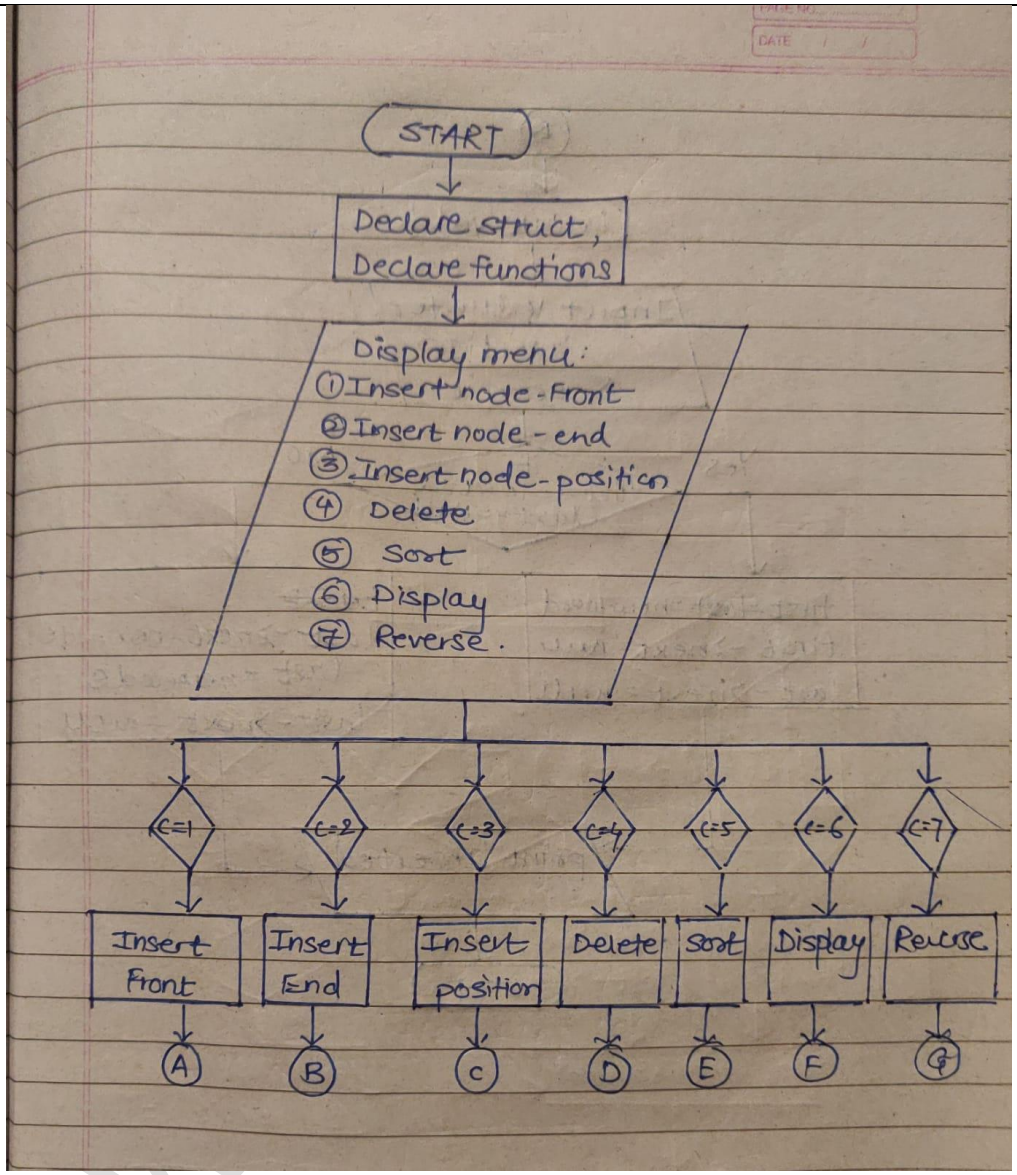
h) DELETE_SPECIFIC POSITION:

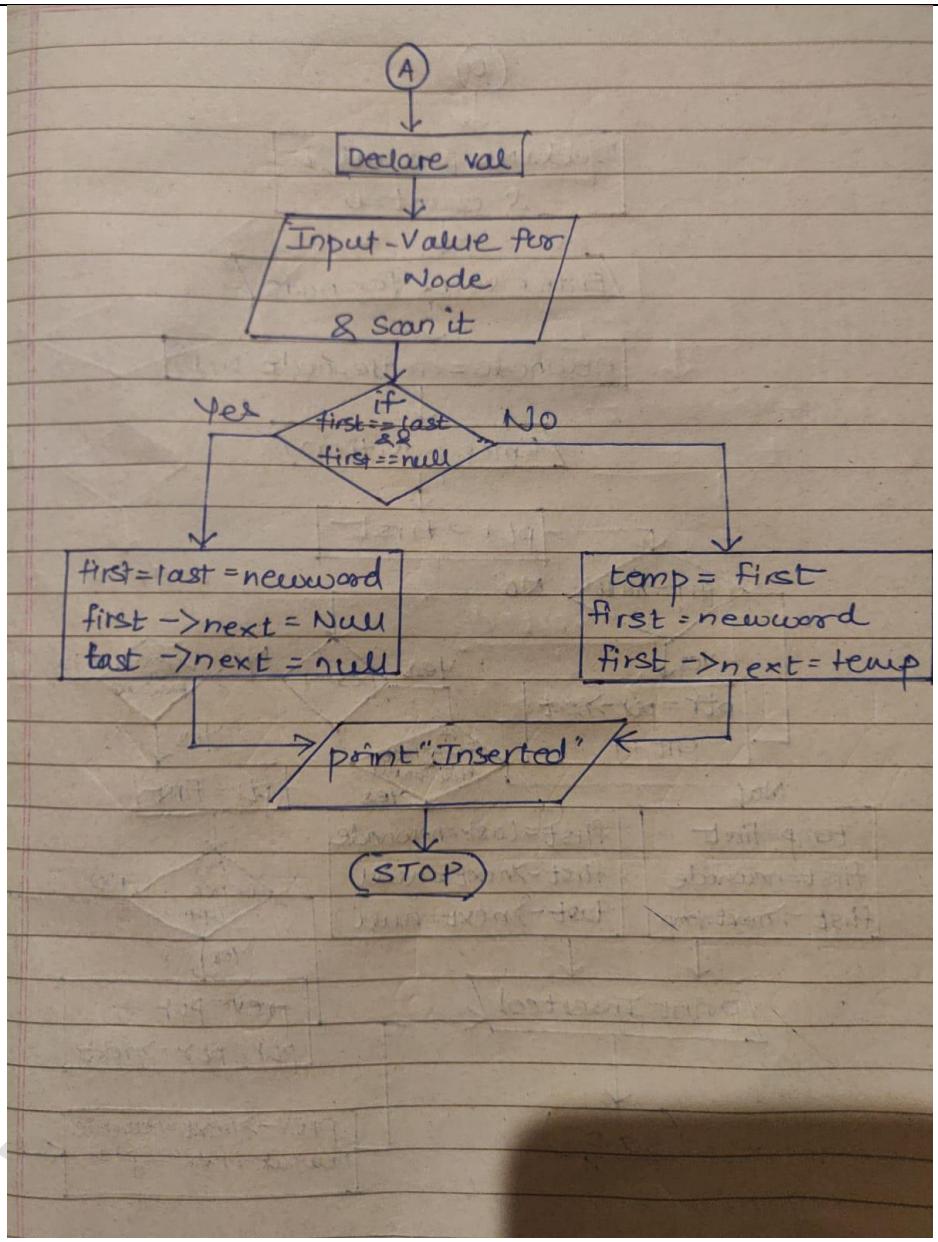
1. Check if start is null and print 'List is Empty'
2. Take input from user for the position of the node to be deleted.
3. Run a while loop to find the pointer of the node at the given position.
4. Let the pointer stored in the next field of this pointer now be the pointer of this node itself.
5. And now use the free() function to free the space stored at the removed pointer.
6. If user chooses to revert the SSL then call the reverse function()

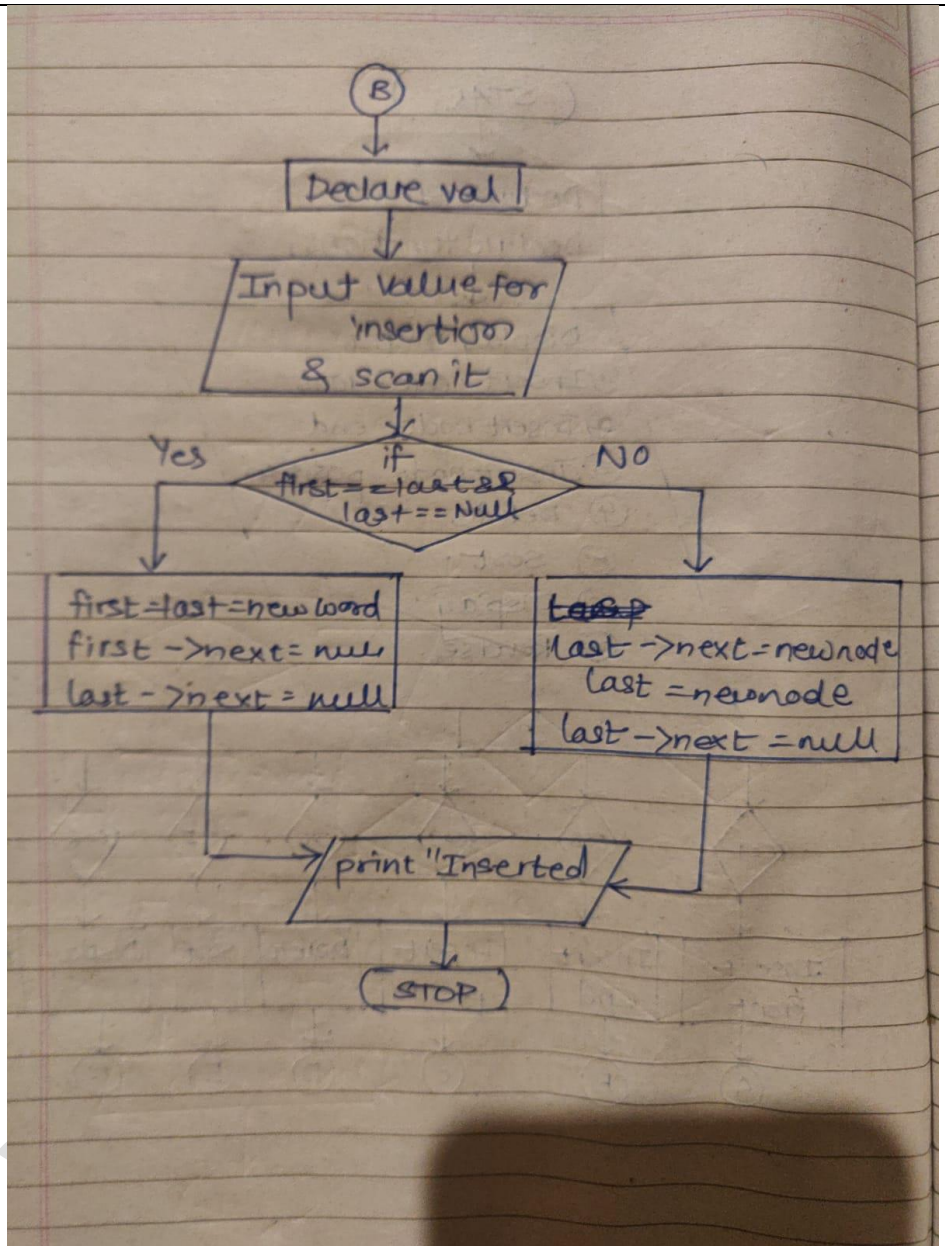
i) REVERSE:

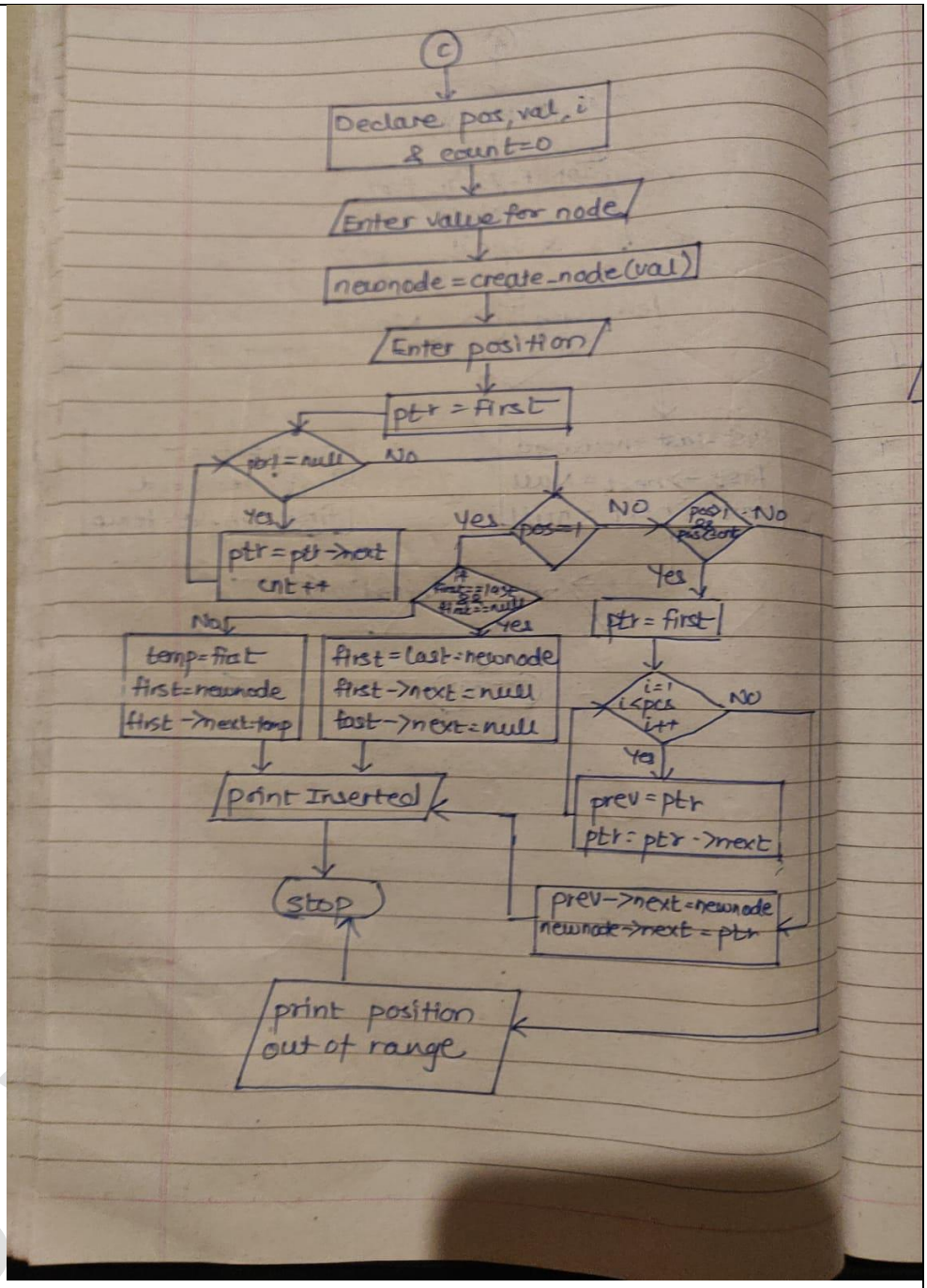
1. Initialize three pointers prev as NULL, curr as head and next as NULL
2. Iterate through the linked list. In loop, do following.
3. Before changing next of current, store next node next = curr->next.
4. Now change next of current, This is where actual reversing happens curr->next = prev
5. Move prev and curr one step forward prev = curr, curr = next
6. If the user chooses to exit the menu then call the exit() function.
7. END

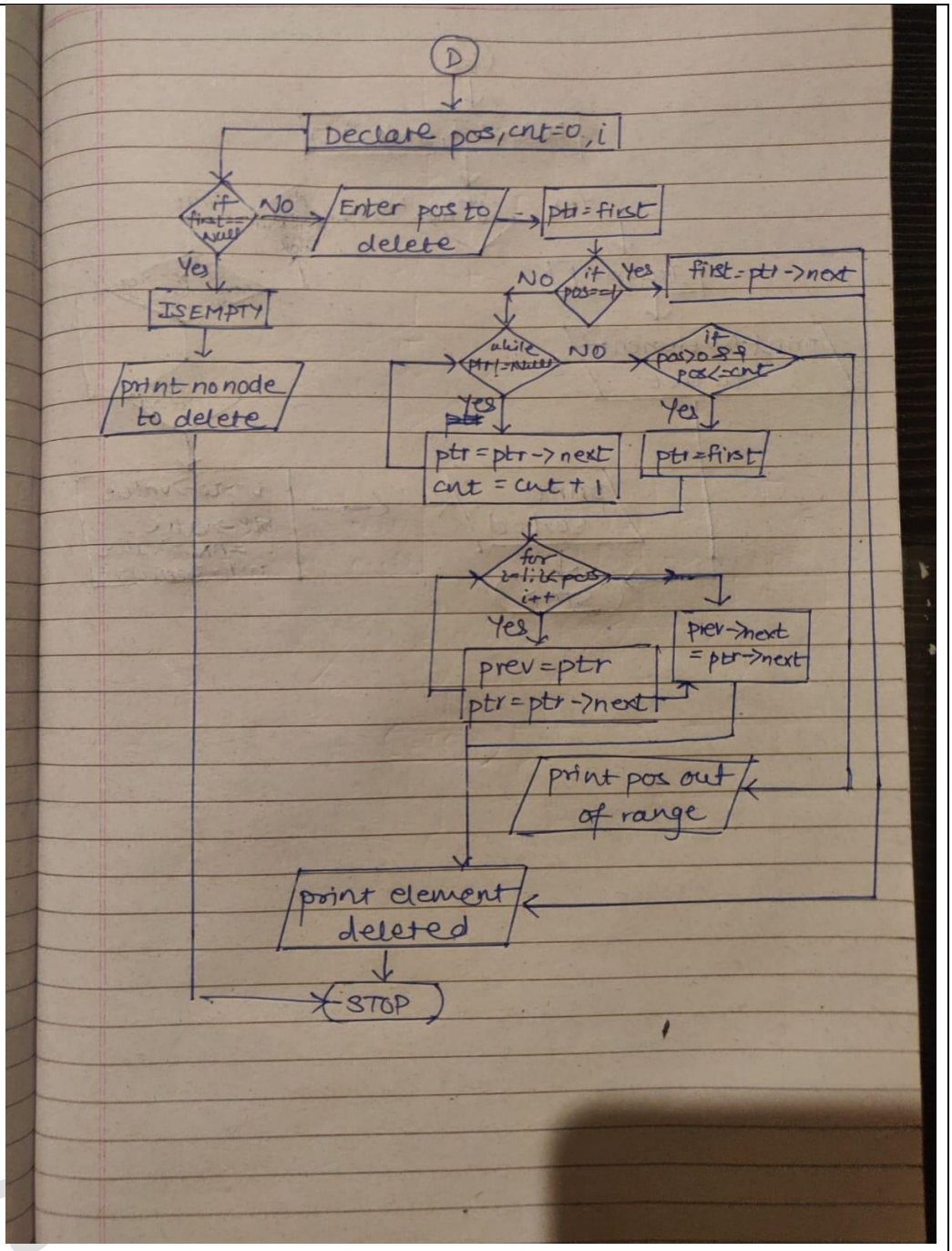
Flow-chart

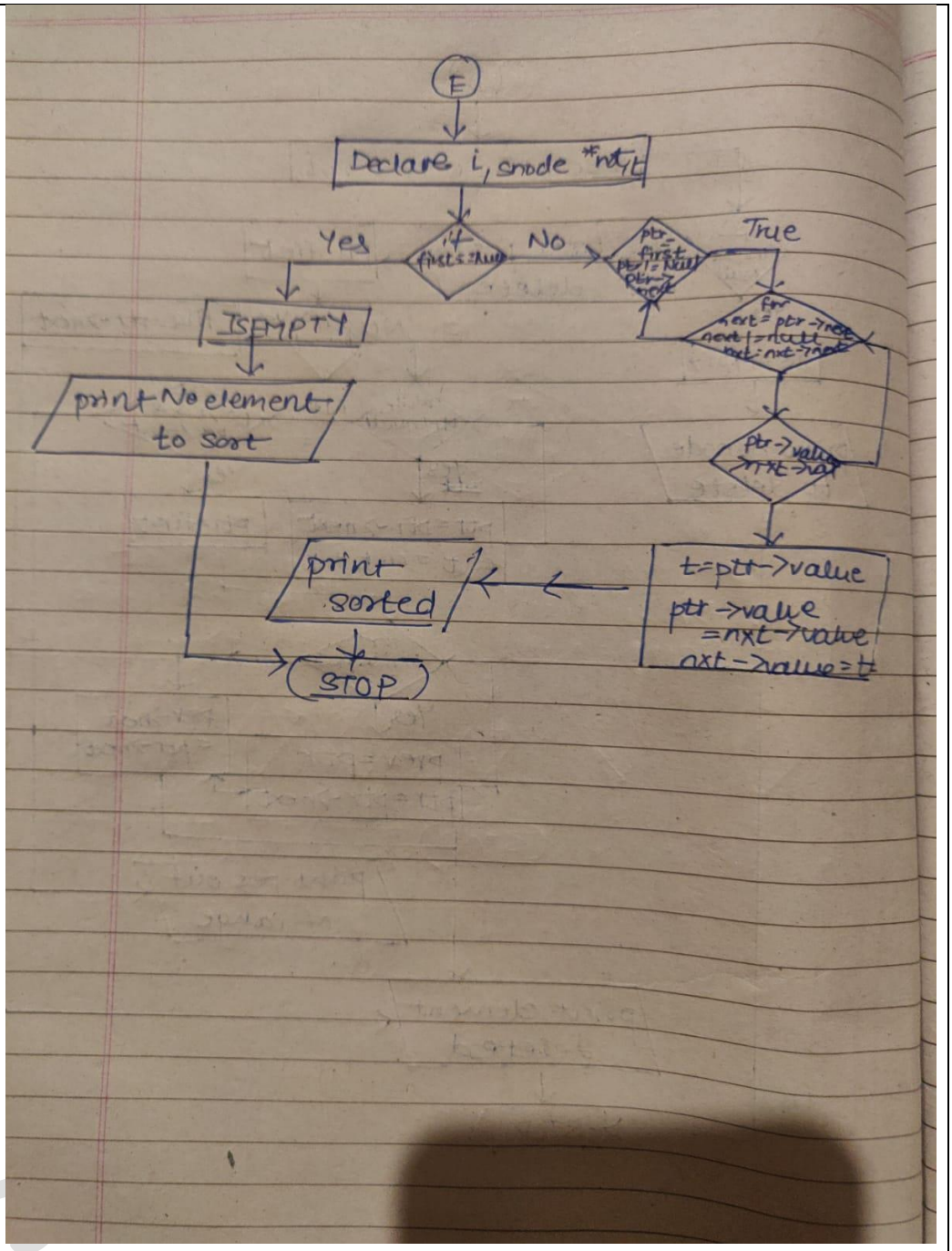


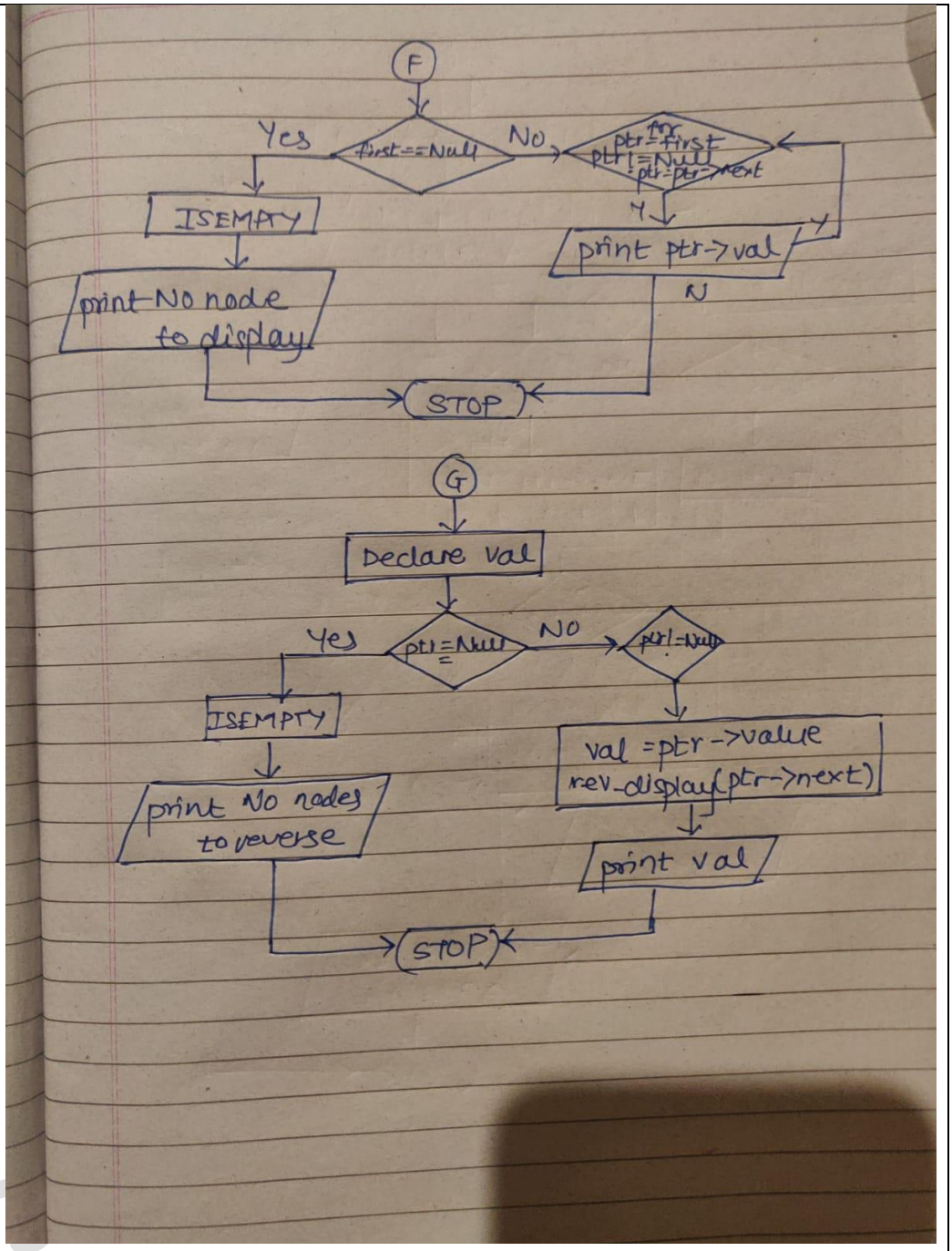

A
Declare val
Input-Value for Node & Scan it
if first==last && first==null
Yes
No
first=last=newword
first->next=Null
last->next=null
temp=first
first=newword
first->next=temp
print"Inserted"
STOP


B
Declare val
Input value for insertion & scan it
if
first==last==Null
Yes
NO
first=last=new node
first->next=null
last->next=null
last->next=newnode
last=newnode
last->next=null
print "Inserted"
STOP


c
Declare pos, val, i & count=0
Enter value for node
newnode = create_node(val)
Enter position
ptr = first
ptr != null
No
Yes
ptr = ptr->next
cnt++
pos==1
Yes
NO
pos>1 && pos<cnt
No
Yes
ptr = first
temp=first
first=newnode
first->next=temp
first = last=newnode
first->next=null
last->next=null
i=1 i<pos i++
NO
Yes
prev = ptr
ptr = ptr->next
print Inserted
stop
prev->next=newnode
newnode->next = ptr
print position out of range


D
Declare pos, cnt=0, i
if first==NULL
NO
Enter pos to delete
ptr = first
Yes
ISEMPTY
print no node to delete
NO
if pos==1
Yes
first = ptr->next
while ptr!=NULL
NO
if pos>0 && pos<=cnt
Yes
ptr = ptr->next
cnt = cnt + 1
Yes
ptr = first
for i=1; i<pos; i++
Yes
prev = ptr
ptr = ptr->next
prev->next = ptr->next
print pos out of range
print element deleted
STOP


E
Declare i, snode *nxt, t
if first = NULL
Yes
No
ISEMPTY
print No element to sort
ptr = first
ptr != NULL
ptr = ptr->next
True
ptr->value > nxt->val
t=ptr->value
ptr->value =nxt->value
nxt->value=t
print sorted
STOP


F
first==Null
Yes
No
ISEMPTY
print No node to display
ptr=first
ptr!=Null
ptr=ptr->next
print ptr->val
N
STOP
G
Declare val
ptr==Null
Yes
No
ptr!=Null
ISEMPTY
print No nodes to reverse
val=ptr->value
rev_display(ptr->next)
print val
STOP

ERROR	No errors occuered
REMEDY	none
CONCLUSION:	
	1) Hence, singly linked list was implemented using theoretical knowledge 2) Dynamic memory allocation method was implemented to perform given tasks 3) Operations on linked list were performed successfully
REFERENCES:	
	1) Ellis Horowitz, Sartaj Sahani, "Fundamentals of Data Structures", Galgotia books. 2) Richard F. Gilberg and Behrouz A. Forouzan, Data Structures A Pseudo code approach with C, cengage learning, 2nd edition. 3) Yashvant Kanetkar-Understanding Pointers in C BPB publications 3rd Edition. 4) E Balgurusamy – C and data structures, 2003 Edition TMH.

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