D Y PATIL GROUP

Assignment No :- 7

- Title :- Program for singly linked list.
- objective :- To study and implement singly linked list.
- Problem statement :- Second year computer Engineering class, set A of student like vanilla Ice-cream and set B of students like butterscotch ice-cream. Write c++ program to store two sets using linked list. compute and display.

a) set of students who like both vanilla and butterscotch.

b) set of students who like either vanilla or butterscotch or not both.

c) Number of students who like neither vanilla nor butterscotch.

- outcome :- student will be able to use linked list data structure and its operations.
- Theory :- A linked list is a linear collection of data elements, called nodes, each pointing to the next node by means of pointer. It is a data structure consisting of group of nodes which together represent a sequence. Under the simplest form, each node is composed of data and a reference (in other words, a link)

PUNE

DYPSOETP, PUNE

to the next node in the sequence. This structure allows for efficient insertion or removal of elements from any position in sequence during iteration. More complex variants add additional links, allowing efficient insertion or removal from arbitrary element references.

- Terminologies of linked list :-

1) Header Node : Header node is option node in the list. The node contains the information about the node structure of node & number of nodes present in list.
2) Head node : This is the first node in linked list.
3) Tail node : This is the last node in linked list.
4) Head pointer : This pointer holds the address of first node.

- Advantages of Linked list :-

1) They are dynamic in nature which allocates the memory when required.
2) Insertion and deletion operations can be easily implemented.
3) stacks and queues can be easily executed.
4) Linked list reduces the access time.

- Disadvantages of linked lists

1) The memory is wasted, as pointers requires extra

memory for storage.

2) No element can be accessed randomly ; it has to access each node sequentially.

3) Reverse Traversing is difficult in linked list.

- Applications of linked list

1) Linked list are used to implement stacks, queues, graphs etc.

2) Linked lists let you insert elements at the beginning and end of list.

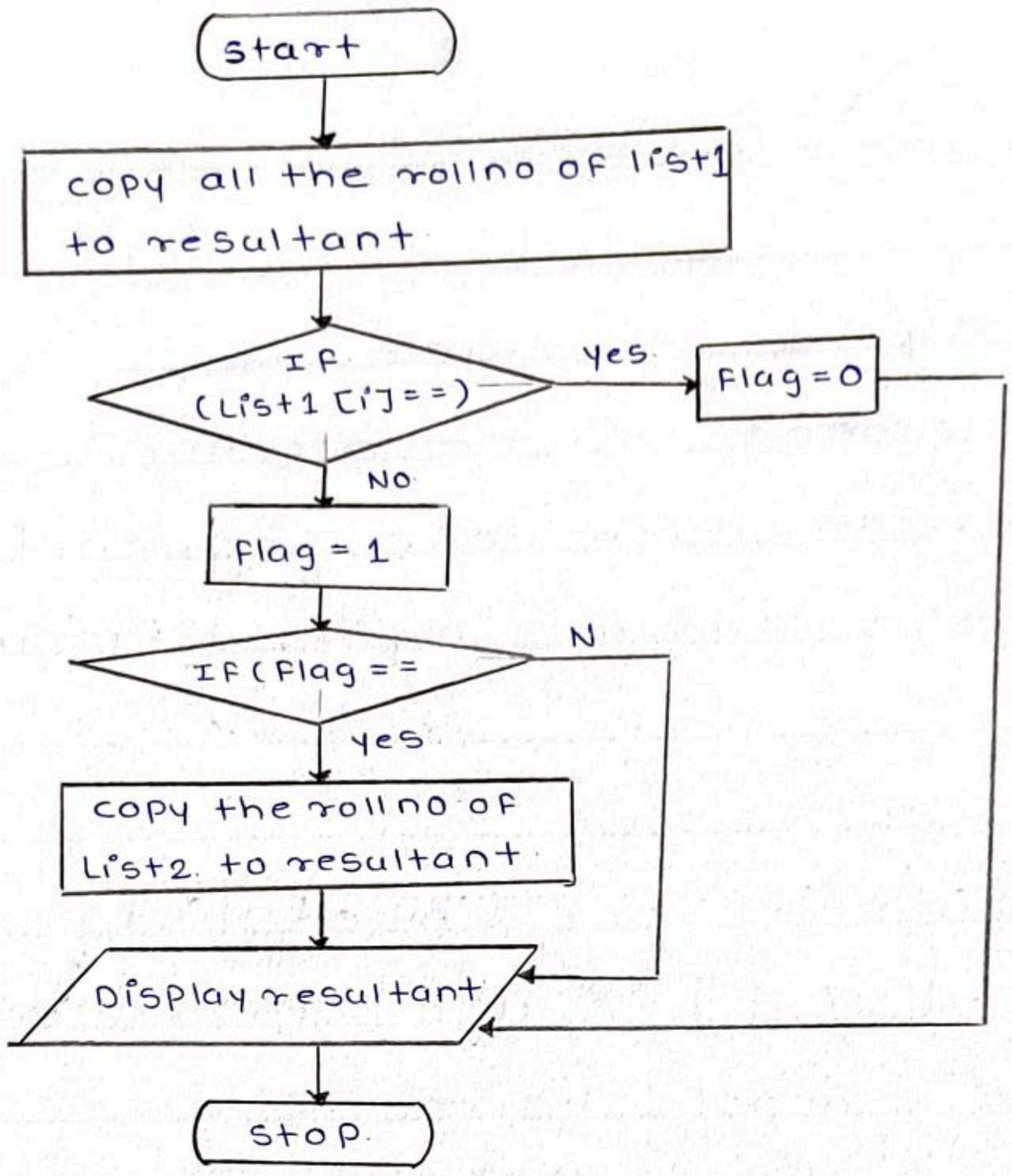
3) In linked lists we don't need to know the size in advance.

Following are the basic operations on linked list :-

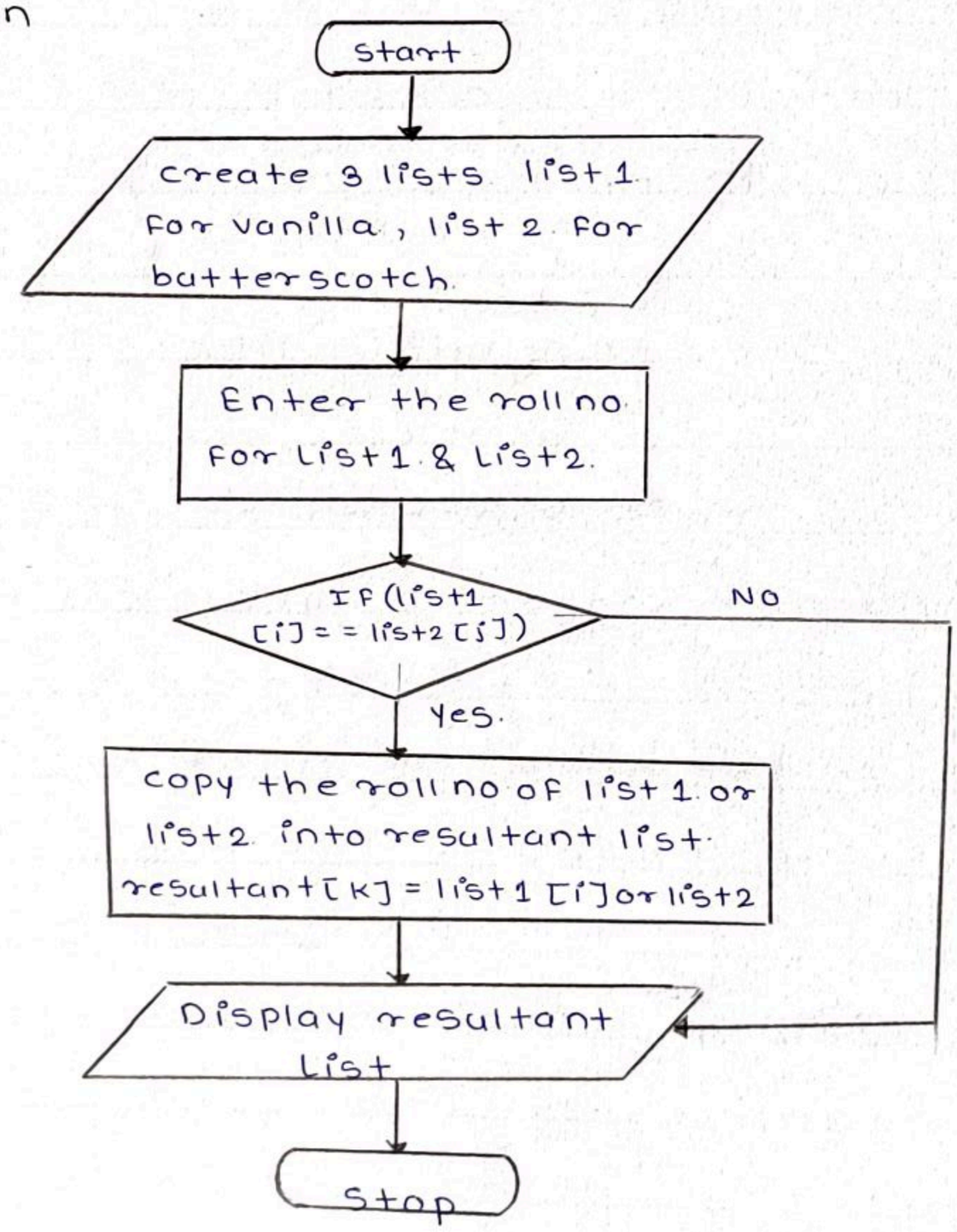
1) create :- This function creates the new node allocate memory dynamically if using new keyword. After creation of node it will add the node at first or at last if list is contains nodes.

2) Print / display :- function takes the start node (as pointer) as argument. If pointer = NULL, then there is no element in list. Else, print the data value of node (pointer -> data) & move to next node by recursively calling

- Union



- Intersection



the print function with pointer -> next sent as an argument.

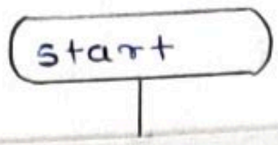
- Algorithm

1) Start
2) Define node structure by using class or structure
3) Define head & tail pointer assign to null inside the constructor.
4) Define methods for create(), getnode(), append(), delete_node(), display(), union(), Intersection(), Difference();
5) create()
While(1)
Begin
Enter any more node to be added (y/n)
If (ans == n) then break
Else
NewNode = GetNode();
Append(NewNode);
Union();
Intersection();
Difference();
End

- Algorithm for getting data of node()
Input : pointer to the linked list.
output : Newly created node newnode
Begin

DYPSOETP, PUNE

- union



NewNode = new Node;

Enter data for new node.

Assign link field to NULL;

Return that node to create function.

End

- Algorithm for appending node()

Input :- pointer to the linked list

output :- linked list with a added node.

if (head == = NULL)

then

head = NewNode;

tail = NewNode;

Else

Begin

tail -> link = NewNode;

tail = NewNode;

End.

- Algorithm to find intersection of two linked list

Input : pointers to two linked list.

output : New linked list with common nodes of two linked list.

Node * temp1 = list1.head, *temp2 = list2.head;

While (temp1 != Null)

begin

while (temp2 != Null)

begin

if (list[i] = = list[j])then

Then add that rollno into the resultant list
End
Display resultant list.

- Algorithm to find Union of two linked list

Input :- pointers to the two linked list.
output :- New linked list with all nodes of two linked list.

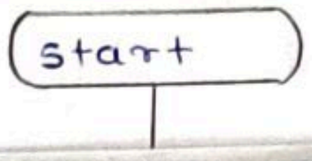
Initialization flag = 0;
Node *temp1 = list1.head, *temp2 = list2.head;
copy the rollno of vanilla list into resultant list.
While (temp2 != Null)
begin
While (temp1 != Null)
begin
If (list2[i] == list1[j]) then
set flag = 1; break;
If (flag == 0) then
Then add remaining roll no. to resultant list
End
Display resultant list.

- Algorithm to find Difference of two linked list

Input: pointers to 2 linked list.
output: New linked list with nodes of either linked list.

Initialization flag = 1;
Node *temp1 = list1.head, *temp2 = list2.head;
While (temp1 != Null)

- union



Begin

While (temp2 != Null)

Begin

If (list[i] == list2[j])

Flag = 0;

Break;

If (Flag == 1)

copy list2 rollno to resultant list.

End

Display the resultant list;

- Algorithm for display of node()

Input :- pointer to the linked list

output :- Elements of the list.

Node * temp = head;

if (temp == NULL) then

print list is empty

Else Begin

While (temp != NULL)

Begin

Display node data

Increment the pointer i.e.

temp = temp → link;

End

- Test cases

	Steps	Input	Expected Result	Actual Result	status
ID1	Enter duplicate number	{1,2,2,3}	Message to be displayed duplicate elements	Message displayed.	pass
ID2	Enter negative roll number	{-1,2,3}	Message to be displayed -ve elements	Message displayed	fail
ID3	Enter two sets	{1,2,3} {1,4,5}	{1,2,3,4,5} Union	{1,2,3,1, 4,5}	Fail
ID4	Enter two sets	{1,2,3} {1,4,5}	{2,3} Difference A-B	{1,2,3}	Fail

- conclusion :-

Successfully implemented all singly linked list operation.

11/11/22

Assignment NO - 8

- Title :- Program for storing binary number using doubley linked lists.

- Objective :- • To store binary numbers using doubly linked list.
  - To find 1's and 2's complement of a binary number.
  - To add two binary numbers.

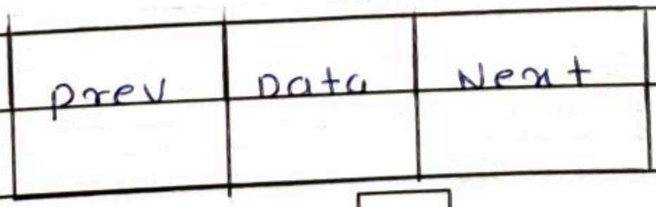
- Problem statement :- Write c++ program for storing binary number using doubly linked list. Write functions.
  a) to compute 1's and 2's complement.
  b) add two binary numbers.

- outcome :- Student will able to use
  - 1's and 2's complement of a binary number.
  - Result of addition of two binary numbers.

- Theory

Doubly linked list :- Doubly linked list is a collection of nodes and each node is having one data field, one previous link field and one next link field.

ex :-



ex:- Find the one's complement of value and then add 1 to

original value	1 1 0 0 0 0 1 0	1 0 0 0 0 0 0 0	1 0 1 0 1 0 1 1
	↓	↓	↓
one's complement	0 0 1 1 1 1 0 1	0 1 1 1 1 1 1 1	0 1 0 1 0 1 0 0
	↓	↓	↓
Add 1	0 0 1 1 1 1 0 1	0 1 1 1 1 1 1 1	0 1 0 1 0 1 0 0
	+ 1	+ 1	+ 1
	0 0 1 1 1 1 1 0	1 0 0 0 0 0 0 0	0 1 0 1 0 1 0 1

The elements can be accessed using both previous link as well as next link. one field is required to store previous link hence node takes more memory in DLL. More efficient access to elements.

- Advantages :-

1) Reversing the Doubly linked list is very easy.
2) It can allocate or reallocate memory easily during its execution.
3) As with a singly linked list, it is the easiest data structure to implement.
4) The traversal of these doubly linked list is bidirectional which is not possible in a singly linked list.
5) Deletion of nodes is easy as compared to a singly linked list. A singly linked list deletion requires a pointer to the node and previous node to be deleted but in the doubly linked list, it only required the pointer which is to be deleted.

- Disadvantages :-

1) It uses extra memory when compared to the array and singly linked list.
2) Since, elements in memory are stored randomly,

DYPSOETP, PUNE

therefore the elements are accessed sequentially no direct access is allowed.

- Algorithm :-

Insertion at Beginning :-

1) start
2) input Data to be inserted.
3) create a new node.
4) New Node ⟶ data = data New Node
   - Lpoint = NULL
5) IF START IS NULL New Node ⟶ Rpoint = NULL
6) Else New Node ⟶ Rpoint = START
   START ⟶ Lpoint = New Node
7) START = New Node.
8) STOP.

Insertion at End :-

1) start
2) Input Data to be inserted.
3) create a New Node
4) New Node ⟶ Data = Data
5) New Node ⟶ Rpoint = NULL
6) IF (START equal to NULL)
   a) START = New Node
   b) New Node ⟶ Lpoint = NULL

7) Else

a) TEMP = START

b) While (TEMP → Next not equal to NULL)

c) TEMP = TEMP → Next

d) TEMP → Rpoint = New Node

e) New Node → lpoint = TEMP

8) STOP

- conclusion :-

Successfully implemented to store binary number using doubly linked list.

18/11/22

Assignment NO-9

- Title :- Use of stack data structure
- objective :- To study & implement stack to check whether given expⁿ is well parenthesized or not
- Problem statement :- In any language program mostly syntax error occurs due to unbalancing delimiter such as (), {}, or []. Write c++ program using stack to check whether given expⁿ is well parenthesized or not.
- outcomes :- Student will be able to check whether given expⁿ is well parenthesized or not, using stack.
- Theory :- A stack can be implemented by means of Array, structure, pointer & linked list. stack. Here, we are going to implement stack using arrays which makes it fixed size stack implementation.
- Basic operation

Push() → Pushing an element on stack.

Pop() → removing an element from stack.

gettop() → get top data element of stack, without removing it.

isfull() → check if stack is full.

isempty() → check if stack is empty.

- Algorithm of gettop() function

begin procedure gettop.

return stack [top]

end procedure.

- Algorithm of isFull() function

```
char stack [MAXSIZE];
begin procedure isFull
if top equals to MAXSIZE
return true
else
return false
end if
end procedure
```

- Algorithm of isempty() function

```
begin procedure isempty
if top less than 0
return true
else.
return false
endif
end procedure.
```

- Algorithm of PUSH() operation

```
begin procedure push : stack, data
if stack is full
return null endif
top ← top + 1
stack [top] ← data
end procedure.
```

- Algorithm for POP() operation.

```
begin procedure pop : stack if stack is empty
return null
```

endif data ← stack [top]

top ← top-1 return data end procedure

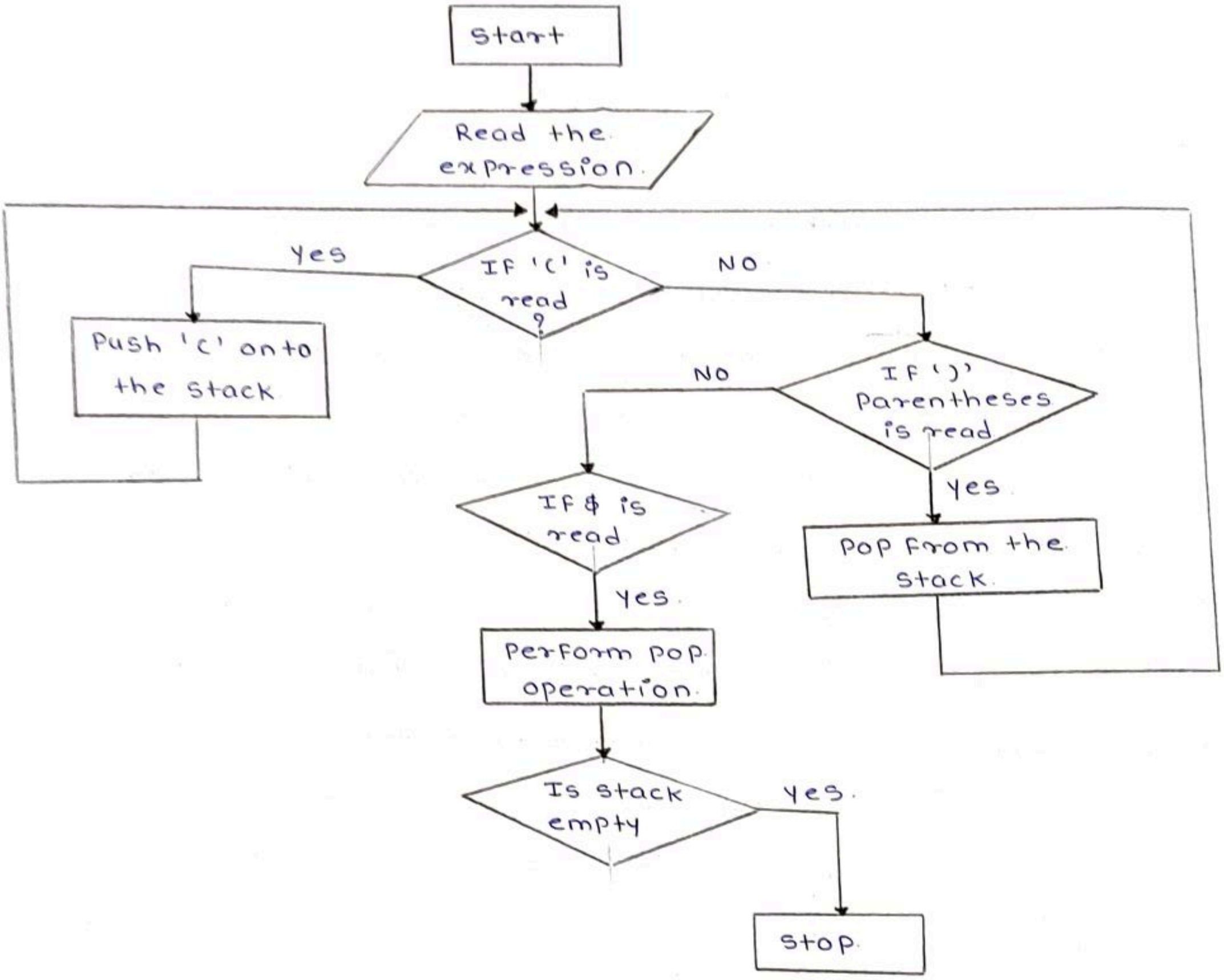
- Algorithm

1) Declare stack S.
2) Now traverse expn string expn.
   a) If current character is starting bracket ('(' or '{' or '[') then push it to stack.
   b) If current character is closing bracket (')' or '}' or ']') then pop from stack.
3) After complete traversal, if there is some string bracket left in stack then "not balanced".

Steps	Input	Expected Result	Actual Result	status
ID1	I/p infix expn	{(a+b)/[d-e]}.	Well formed Parentheses	Well formed Parentheses	pass
ID2	I/p infix expn	{(a+b)/[d-e]	Not well formed Parentheses	Not well Formed Parentheses	Fail
ID3	I/p infix expn	[(a+b)]	Not well formed Parentheses	Not well Formed Parentheses	Fail

- conclusion :- Successfully implemented a program to check well-formed parentheses using stack data structure.

25/11/22


Start
Read the expression.
IF 'C' is read ?
Yes
Push 'C' onto the stack
NO
IF ')' Parentheses is read
NO
Pop from the stack
Yes
IF $ is read
Yes
Perform pop operation
Is stack empty
Yes
stop

Assignment NO :- 10

- Title :- Use of stack data structure.
- objective :- To study and implement conversion of infix to postfix notation and its evaluation.
- Problem statement :- Implement c++ program for expression conversion as infix to postfix and its evaluation using stack based on given condition.
  1) operands and operator, both must be single character.
  2) Input postfix expression must be in a desired format.
  3) only '+', '-', '*', '/' operators are expected.
- outcome :- Student will be able to convert infix to postfix and evaluate it, using stack.
- Theory :- A stack is container of objects that are inserted and removed according to the last-in-first-out (LIFO) principle. In the pushdown stacks only two operations are allowed : PUSH the item into the stack and POP the item out of stack. A stack is 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.

- Basic operations :-

push() - pushing (storing) an element on the stack.

pop() - removing (accessing) an element from the stack.

To use a stack efficiently we need to check status of stack through following functionality,

gettop() - get the top data element of the stack without removing it.

isFull() - check if stack is full.

isEmpty() - check if stack is empty.

The top pointer provides top value of stack without actually removing it.

- Infix notation : x + y

operators are written in-between their oprands. This is the usual way we write expressions.

- Postfix notation (also known as "Reverse Polish notation") : x y +

operators are written after their oprands. The infix expression given above is equivalent to A B C + * D /

- Prefix notation (also known as "Polish notation"): + x y

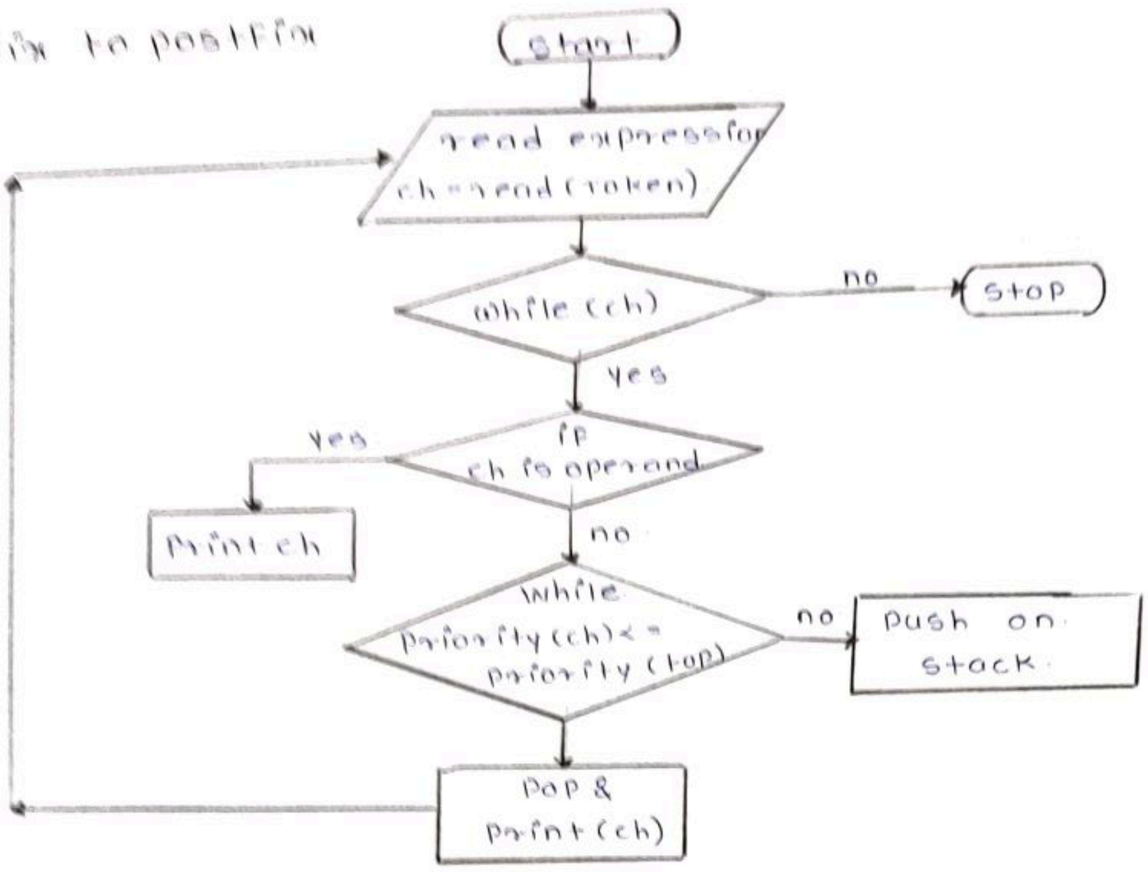
operators are written before their operands. The expression given above are equivalent to

/* A + B c D

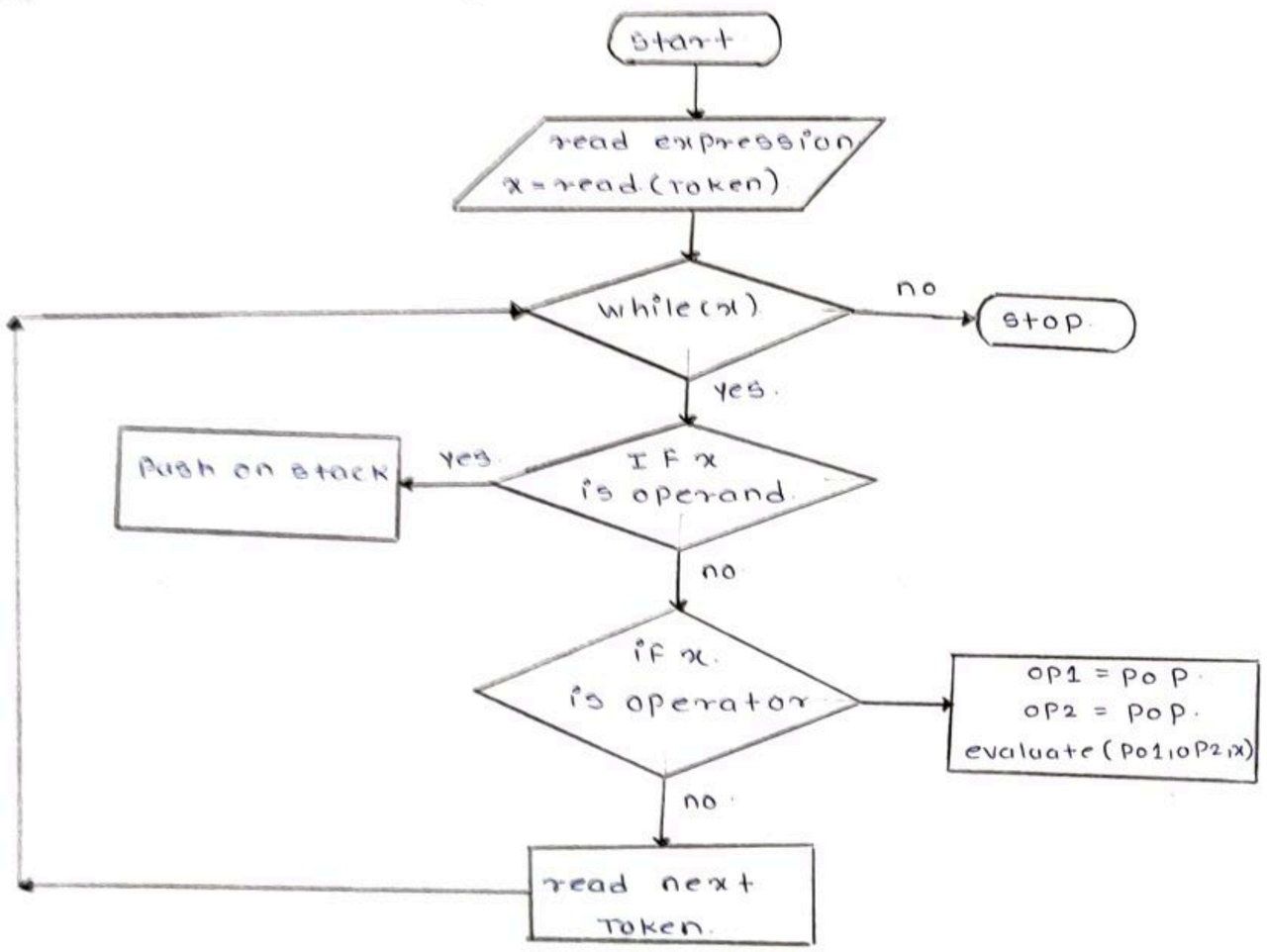
- Algorithm : conversion INFIX TO POSTFIX

```
stack s
char ch
char element
While (Token are available)
{
 ch = Read (Token);
 if (ch is operand)
 {
    Print ch;
 }
 else
 {
  While (priority (ch) <= priority (Top most stack))
  {
   element = pop (s);
   print (ele);
  }
  push (s, ch);
 }
}
while (! Empty (s))
{
 element = pop (s);
```

1) Infix to postfix



2) Evaluate Postfix



```
print (element);
}
```

- Algorithm : Evaluate POSTFIX expression

```
Initialize (stack s)
x = Read Token(); //Read Token.
While (x)
{
if (x is operand)
   push (x) onto stack s.
  if (x is operator)
   {
   operand 1 = pop (stack s);
   operand 2 = pop (stack s);
   Evaluate (operand 1, operand 2, operator x);
   }
x = Read Next Token(); // Read Token.
}
```

- Test cases :-

steps	Input	Expected Result	Actual Result	state
TD1 Input a infix expression	(a+b)/(d-e)	ab+de-/	ab+de-/	pass
TD2 Input a infix expression	(a+b)/(d-e)	ab+de-/	ab+del-	Fail

1) Infix to postfix

start

D Y PATIL GROUP

- conclusion :-

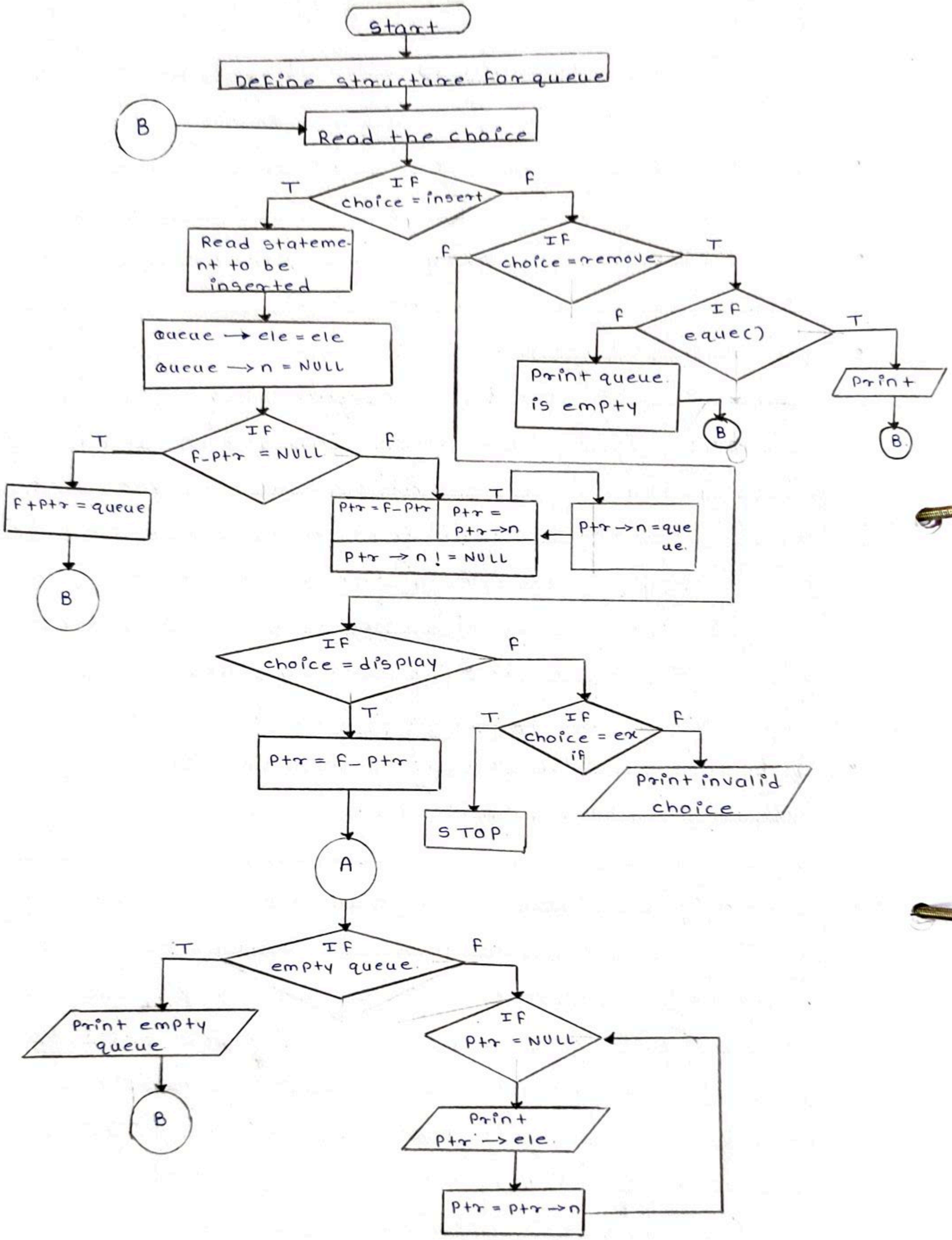
successfully implemented a program of Infix to postfix conversion using stack.

02/12/22

Assignment NO - 11

- Title :- Use of queue data structure
- objective :- To study & implement queue & its operations.
- problem statement :- queues are frequently used in computer programming & typical example is creation of job queue by an operating system. If operating system does not use priorities, then jobs are processed in order they enter the system. Write c++ program for simulating job queue. Write functions to add job & delete job from queue.
- outcome :- student will be able to implement queue & its operations.
- Theory :- Queue is also an abstract data type or linear data structure, in which 1st element is inserted from one end called REAR (also called tail), & the deletion of existing element takes place from the other end called as FRONT (also called head). This makes queue as FIFO data structure, which means that element inserted 1st will also be removed 1st.

The process to add an element into queue is called Enqueue & the process of removal of an element from queue is called


Start
Define structure for queue
B
Read the choice
IF choice = insert
T
F
Read statement to be inserted
IF choice = remove
F
T
Queue → ele = ele
Queue → n = NULL
IF equ()
f
T
Print queue is empty
B
Print
B
IF f_ptr = NULL
T
f
f_ptr = queue
B
ptr = f_ptr
ptr = ptr→n
ptr → n != NULL
ptr → n = queue
IF choice = display
f
T
ptr = f_ptr
IF choice = exit
T
f
STOP
Print invalid choice
A
IF empty queue
T
f
Print empty queue
B
IF ptr = NULL
Print ptr → ele
ptr = ptr → n

Dequeue.

Basic operations :-

enQueue(value) - Inserting value into queue.

In queue data structure, enQueue() is function used to insert new element into queue. In queue, the new element is always inserted at rear position. The enQueue() function takes place one integer value as parameter & inserts that value into queue. We can use the following step to insert an element into queue...

Step 1 :- check whether queue is FULL. (rear == SIZE-1)

Step 2 :- If it is FULL, then display "Queue is FULL!! Insertion is not possible!!!" & terminate function.

Step 3 :- If it is NOT FULL, then increment rear value by one (rear++) & set queue[rear] = value.

deQueue() - Deleting a value from queue.

In queue data structure, deQueue() is a function used to delete an element from the queue. In a queue, the element is always deleted from front position. The deQueue() function does not take any value as parameter. We can use the following steps to delete an element from the queue...

Step 1 :- check whether queue is Empty (front == rear)

Test ID	Steps	Input	Expected Result	Actual result	Status
ID1	Enqueue(x) If Full()	x	Queue is Full	Queue is Full	Fail
ID2	Enqueue(x) If empty()	x	1 element added in queue.	1 element added in queue	Pass
ID3	dequeue() If empty()	-	error	Queue is empty.	Fail
ID4	dequeue() If Full()	-	1 element deleted from the queue	1 element deleted From the queue	Pass

Step 2 :- If it is Empty, then display "Queue is Empty!!! Deletion is not possible!!!" and terminate the function.

Step 3 :- 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)

- ALGORITHM

Algorithm : Procedure isfull
if rear equals to MAXSIZE
  return true
else
  return false
endif
endprocedure
Algorithm : Procedure isempty
if front is less than MIN OR front is greater than rear
  return true
else
  return false
endif
end procedure.

Algorithm : Procedure enqueue (data)
if queue is full
return overflow
end if
rear ← rear + 1
queue [rear] ← data
return true
end procedure

Algorithm : Procedure dequeue
if queue is empty
return overflow
end if
data = queue [front]
front ← front + 1
return true
end procedure

- conclusion :-

Successfully implemented different operations on queue.

05/12/22

D Y PATIL GROUP

Assignment NO - 12

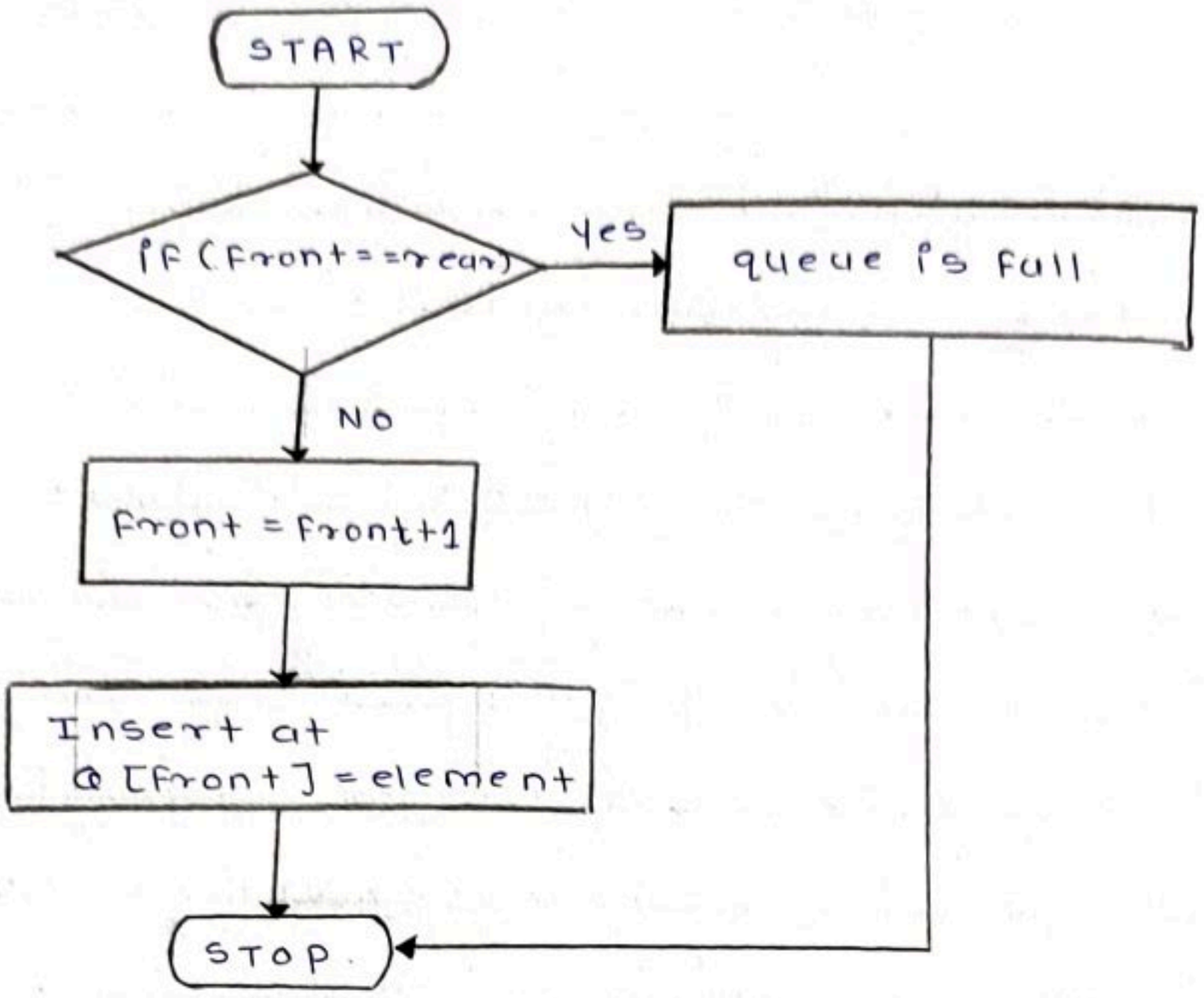
- Title :- Use of queue Data structure.
- objective :- To study & implement deque and its operation.
- Problem statement :- A double-ended queue (deque) is a linear list in which addition & deletion may be made at either end. obtain a data representation mapping a deque into 1-D array. Write c++ program to simulate deque with functions to add & delete elements from either end of the queue.
- outcome :- student will be able to implement deque & its operations.
- Theory

Deque (usually pronounced like "deck") is an irregular acronym of double-ended queue. Double-ended queues are sequence containers with dynamic size that can be expanded or contracted on both ends (either its back or front)

operations of deque :-

1) Deque() creates a new deque that is empty. It needs no parameter & returns empty queue.
2) addFront(item) adds a new item to the front of dequette. It needs item & returns nothing.

- Flowchart
- Insert element from front



3) removeFront() removes the front item from deque. It needs no parameters & returns items. The deque is modified.

4) addRear(item) adds a new item to rear of deque. It needs item and returns nothing.

5) removeRear() removes the rear item from deque. It needs no parameters & return item. The deque is modified.

6) isEmpty() tests to see whether deque is empty. It needs no parameter & returns boolean value.

7) size() returns the number of items in deque It needs no parameter & returns an integer.

- Algorithm to add element into Dequeue.

Assumptions : pointer f, r & initial values are -1, -1.

Q[] is an array

max represent the size of a queue

enq - front

step 1 :- start

step 2 :- check the queue is full or not as if (f<>

step 3 :- If false update pointer f as f = f-1

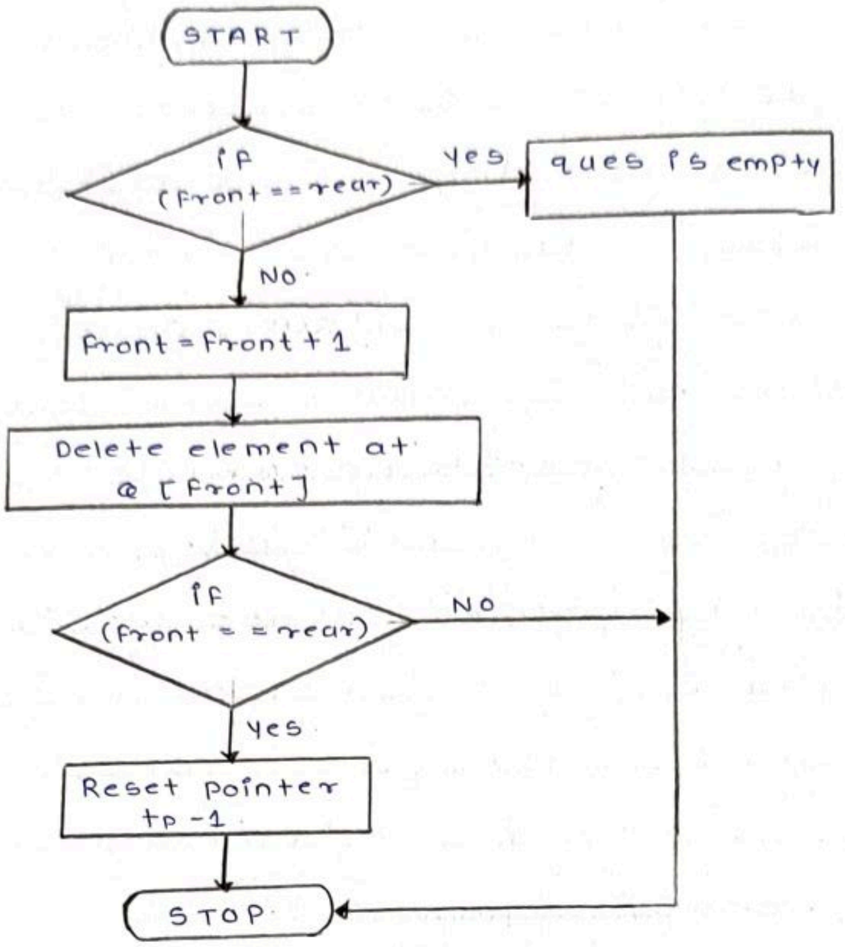
step 4 :- Insert element at pointer f as Q[f] = elem

step 5 :- stop

enq - back

step 1 :- start

- Delete element from front



Step 2 :- check the queue if full or not (r == max-1)

Step 3 :- If false update pointer r as r = r+1

Step 4 :- Insert element at pointer r as q[r] = element.

Step 5 :- Stop.

- Algorithm to delete an element from Dequeue

deq-front

Step 1 :- Start

Step 2 :- check queue is empty or not as if (f == r)
If yes queue is empty.

Step 3 :- If false update pointer f as f = f+1 &
delete element at position f as = q[f]

Step 4 :- If (f == r) reset pointer f & r as f = r = -1.

Step 5 :- Stop.

deq-back

Step 1 :- start

Step 2 :- check queue is empty or not if (f == r)

Step 3 :- If false delete element at position r as
element = q[r].

Step 4 :- Update pointer as r = r-1.

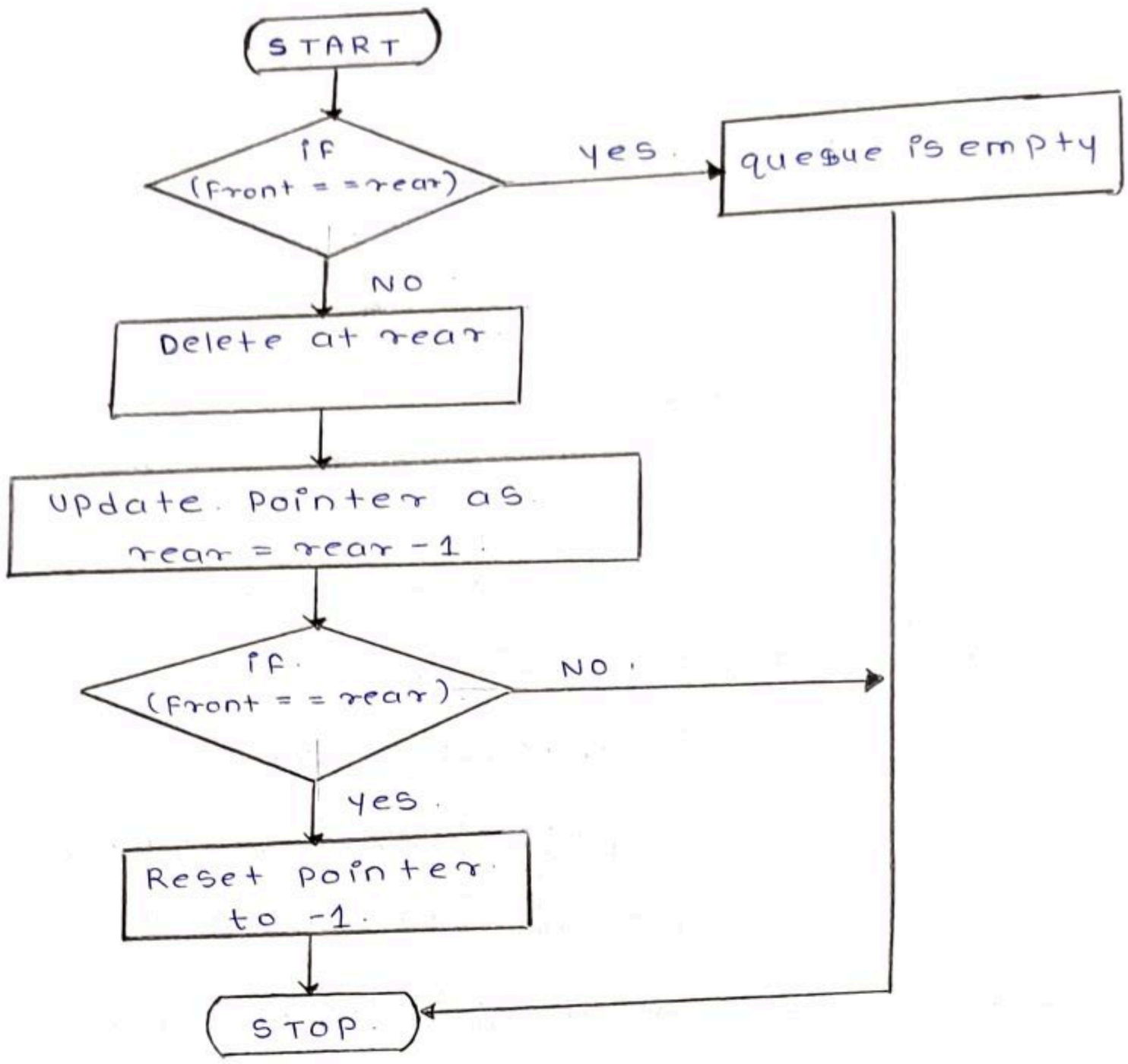
Step 5 :- If (f == r) reset pointer f and r as
f = r = -1

Step 6 :- Stop

- conclusion :- Successfully implemented deque and its operation.

06/12/22

- Delete element from rear



- TEST CASES :-

Test ID	steps	Input	Expected Result	Actual Result	Status
ID1	Insert element from front of the deque	10	10,20,30,40	10,20,30,40	Pass
ID2	Delete element from front of deque.	20	30,40	30,40	Pass
ID3	Delete element from rear of the deque	40	30	30,40	Fail

D Y PATIL GROUP

Assignment NO - 13

- Title :- Use of queue data structure
- objective :- To study & implement circular queue using array.
- outcome :- student will be able to use circular queue data structure & can perform its operations.
- Problem statement :- Pizza parlour accepting max. M orders. orders are served in 1st come 1st served basis. order once placed can't be cancelled. Write c++ program to simulate system using circular queue using array.
- Theory :-

What is circular queue :- circular queue is a linear data structure in which operations are performed based on FIFO principle & the last position is connected back to 1st position to make circle. It is also called 'Ring Buffer'.

The trouble of linear queue is that the rear of queue is at end of the array. Even if there are empty cells at beginning of array, bcoz you have removed them, you still can't insert new item bcoz Rear can't go any further.

To avoid problem of not being able to insert more items into queue even when it's not full, front & rear arrows wrap around to beginning of array. The results a circular queue.

A circular queue is queue but a particular implementation of queue. It is very efficient. In a circular queue, after rear reaches end of queue, it can be reset to zero. This helps in refilling empty spaces in betn.

→ Implementation of operation on circular queue.

1) (Front = 0) & (rear = capacity - 1)

2) Front = rear + 1

→ There are three scenarios

1) If queue is empty, then value of Front & rear variable will be -1, then both Front & rear are set to 0.

2) If queue is not empty, then value of rear will be index of last element of queue, then rear variable is incremented.

3) If queue is not full & value of rear variable is equal to capacity - 1 then rear is set to 0.

- Algorithm to insert an element into circular queue as Array :

1) check whether queue is Full.
i.e. check ((rear == SIZE-1 && Front == 0) || (rear == Front-1)).

2) If it is full then Display queue is Full.
If queue is not full then, check if (rear =

(rear == SIZE -1 && Front != 0)

If it is true then set rear = 0 & insert element.

Time complexity is O(1) as there is no loop in this operation.

- Algorithm to delete an element from circular queue as Array from front end.

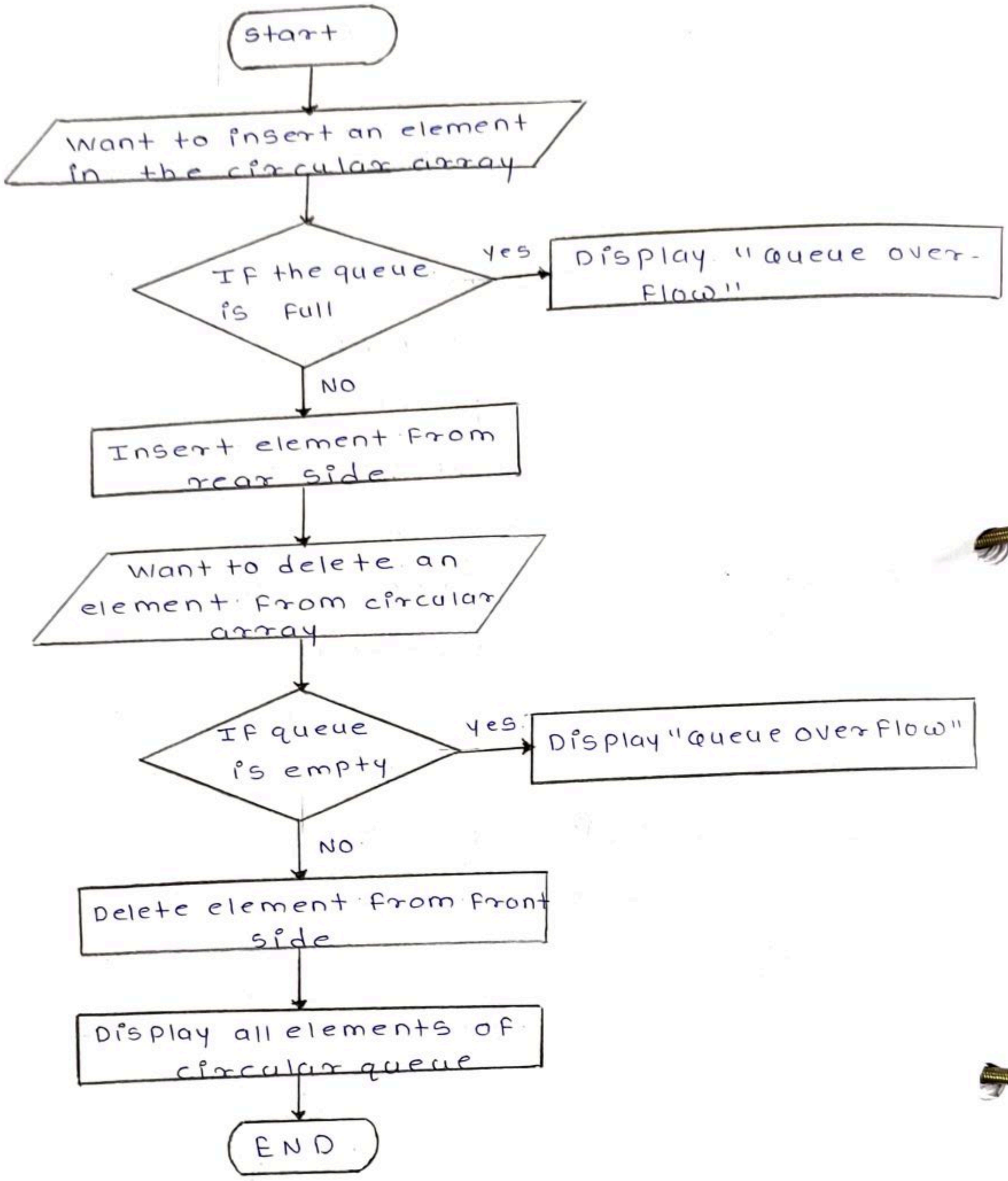
1) check whether queue is empty means check (front == -1)
2) If it is empty then display queue is empty.
3) check if (front == rear) if it is true then set front = rear = -1 else check if (front == size-1), if it is true then set front = 0 & return element.

- Time complexity is O(1) as there is no loop in this operation.
- Test cases :-

	steps	Input	Expected result	Actual result	status
TD1	To insert	insert an element into circular array.	Inserted successfully at the rear end.	If queue is full then not possible.	fail
TD2	To delete	delete the element from circular queue.	deleted successfully at front end	If queue is empty not possible	fail

- conclusion :- Implemented application of circular queue.

05/12/22


Start
Want to insert an element in the circular array
IF the queue is full
yes
Display "Queue over-Flow"
NO
Insert element from rear side
Want to delete an element from circular array
IF queue is empty
yes
Display "Queue overFlow"
NO
Delete element from front side
Display all elements of circular queue
END