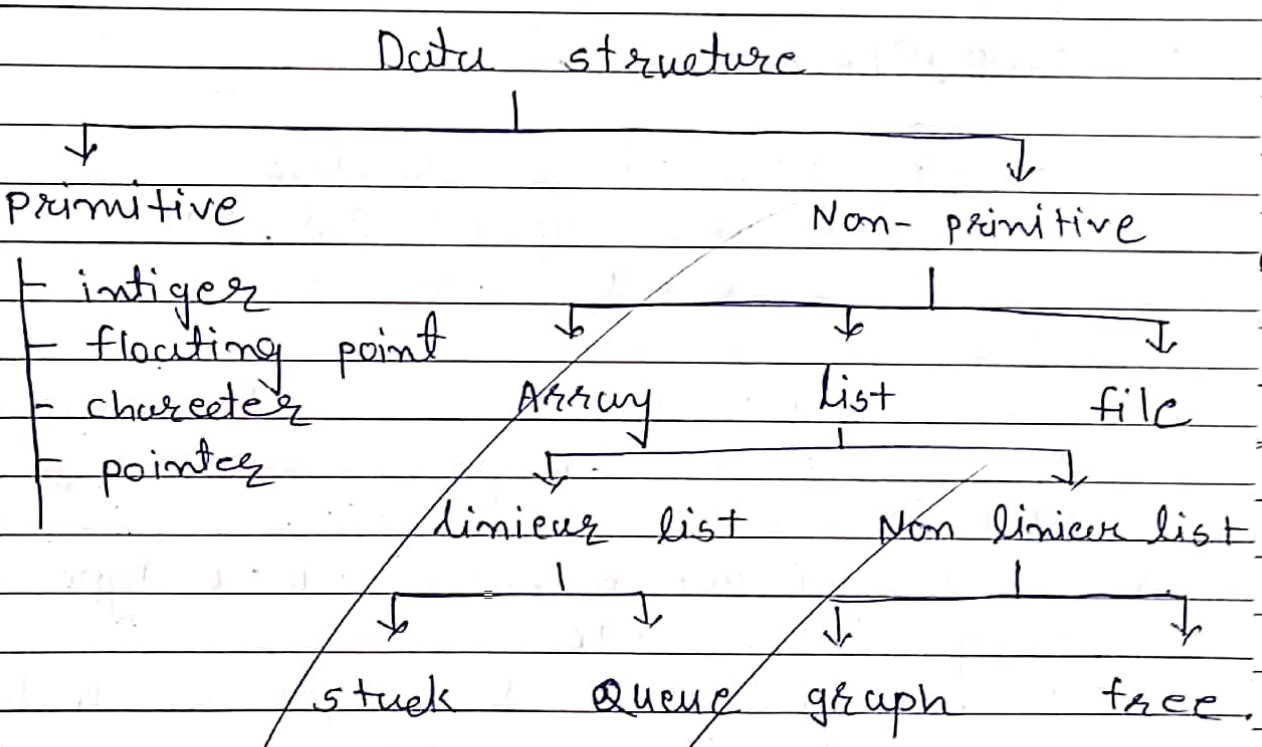
Page No.:
Date: / /

B

ASSIGNMENT - 1

Q) Define data structure, list out types of data structure and explain them in brief. Differentiate the following terms.
primitive & Non-primitive data structure.
linear and Non-linear data structure.

⇒ Data structure is representation of the logical relationship existing between individual elements of data.



⇒ Data structure are normally divided into two broad categories.

1) primitive data structure
2) Non-primitive data structure.

Page No.:
Date: / /

primitive data structure :-

⇒ primitive data structure are basic structures and are directly operated upon by machine instructions.

⇒ primitive data structure have different representation on different computers

⇒ Integer, float, character & pointers are examples of primitive data structure.

⇒ These data types are available in most programming languages as built in type.

- Integer
- float
- character
- pointer

Non-primitive data structure.

⇒ These are more sophisticated data structures

⇒ These are derived from primitive data structure.

⇒ The non-primitive data structure emphasize of a group of homogeneous or heterogenious data items

⇒ examples of Non primitive data type are array, list, file etc.

⇒ A non-primitive data type is further divided into linear & Non-linear data structure.

- Array
- list
- file.

linear Data structure :-

⇒ A data structure is said to linear, if its elements are connected in linear fashion by means of logically or in sequence memory location.

⇒ There are two ways to represent a linear data structure in memory.
- static memory allocated
- Dynamic memory allocation.

⇒ The possible operation on linear data structure are : Traversal, Deletion, insertion searching, sorting, & Merging.

⇒ example of linear data structure are stack and queue.

Non linear data structure :

⇒ Non-linear data structures are those data structure in which data items are not arranged in sequence.

⇒ examples of Non-linicar data structure are tree and graph.

2) Discuss best case, average case and worst case time analysis with example.

Best case Analysis

⇒ In the best case analysis we calculate lower bound on running time of an algorithem we must know the case that causes min no of operation to be executed. in the linear search problem the best case occurs when x is present at the first location. the number of operation in worst case is const. so time complexity is the best case

Average case Analysis

⇒ In average case analysis, we take all possible inputs and calculate computing time for all of the input. sum all the calculated values and divide the sum by total number of inputs we must know distribution of cases. for the linear search problem, let us assume that all cases are uniformly distributed, so we sum all the cases and dive the sum by (n+1).

Worst case Analysis:

⇒ In the worst case analysis, we calculated upper bound on running time of an algorithem we must know the case that causes maximum number of operation to be executed. for linear search. the worst case happens when the element to be serched is not present in the array. when x is

not present, the search() function compares it with all the elements of array one by one. Therefore, the worst case time complexity of linear search would be.

3) what does abstract data type means?

⇒ An abstract data type is the way we look at the data structure, focusing on what it does and ignoring how it does its job. The word 'abstract' in the context of data structures means considered apart from the detailed specification or implementation.

⇒ In C, an abstract data type can be considered without regard to its implementation. It can be thought of as a description of the data structure with a list of operations that can be performed on the data within the structure.

⇒ The end-user is not considered concerned about how the details of how the methods carry out their tasks. They are only aware of the method that are available to them and are only concerned about calling these methods and getting the results. They are not concerned about how they work.

4) Write an algorithem to implemeit PUSH POP & CHANGE operations on stuck.

PUSH algorithem.

1) [check for stuck overflow]

if (TOP >= N)
then write ('STACK OVERFLOW')
Return

2) [TOP Increment]
TOP ← Top + 1

3) [Insert element]
S[TOP] ← x

4) [finished]
Return.

POP algorithem.

1) [check for underflow on stuck]
if TOP = 0
then write ('STACK UNDERFLOW ON POP')
take action in response to underflow
exit.

2) [pointer decrement]
TOP ← TOP - 1

3) [Return formerr top element of stuck]
Return (S (top + 1))

PUSH

CHANGE algorithem.

1) [check for stack underflow]
if TOP - i + 1 ≤ 0
then write ('STACK UNDERFLOW ON CHANGE')

2) [Change i^{th} element from top of stack]
S[TOP - i + 1] ← x

3) [finished]
Return.

5) convert infix to prefix / postfix format showing stack status after every step in tabular form.

i) ((A + B) * C - D ^ E ^ (F * G)) postfix

Next Char	content of stack	polish Notation	Rank.
(	(	-	-
(	((	-	-
A	((A	-	-
+	((+	A	1
B	((+B	A	1
)	(	AB +	1
*	(*	AB +	1
C	(* C	AB +	1
-	(-	AB + C *	1
D	(- D	AB + C *	1

Next Char	content of stack	polish Notation	Rank
^	(- ^	A B + C * D	2
E	(- ^ E	A B + C * D	2
^	(- ^ ^	A B + C * D E	3
(	(- ^ ^ (	A B + C * D E	3
F	(- ^ ^ (F	A B + C * D E	3
*	(- ^ ^ (*	A B + C * D E F	4
G	(- ^ ^ (* G	A B + C * D E F	4
)	(- ^ ^	A B + C * D E F G *	4
)	-	A B + C * D E F G * ^ ^ -	2

ii) ((A + B * D + E / (F + G * D) + C) -postfix

Next Char.	Content of stack	polish Notation	Rank.
(	(	-	-
(	((	-	-
A	((A	-	-
+	((+	A	1
B,	((+ B	A	1
*	((*	A B +	1
D	((* D	A B +	1
+	(+	A B + D *	1
E	(+ E	A B + D *	1
/	(+ /	A B + D * E	2
(	(+ / (	A B + D * E	2
F	(+ / (F	A B + D * E	2
+	(+ / (+	A B + D * E F	3
G	(+ / (+ G	A B + D * E F	3
*	(+ / (+ *	A B + D * E F G	4
D	(+ / (+ * D	A B + D * E F G	4
)	C + /	A B + D * E F G D * +	3

+	(+	AB + D * E F G D * + / +	1
C	(+ C	AB + D * E F G D * + / +	1
)	-	AB + D * E F G D * + / + C +	1

iii) A / B $ C + D * E / F - G + H Prefix

⇒ After inverting expression # H + G - F / E * D + C $ B / A #

Next Char	content of stack	polish Notation	Rule
#	#	-	-
H	# H	-	-
+	# +	H	1
G	# + G	H	1
-	# -	H G +	1
F	# - F	H G +	1
/	# - /	H G + F	2
E	# - / E	H G + F	2
*	# - *	H G + F E /	2
D	# - * D	H G + F E /	2
+	# +	H G + F E / D * -	1
C	# + C	H G + F E / D * -	1
$	# + C $	H G + F E / D * - C	2
B	# + $ B	H G + F E / D * - C	2
/	# + /	H G + F E / D * - C B $	2
A	# + / A	H G + F E / D * - C B $	2
#	-	H G + F E / D * - C B $ A / +	1

prefix : + / A $ B C - * D / E F + G H

iv) a - b / c * d + e * f / g prefix.

⇒ # g / f * e + d * c / b - a #

Next char	content of stack	polish Notation	Rank
#	#	-	-
g	# g	-	-
/	# /	g	1
f	# / f	g	1
*	# *	g f /	1
e	# * e	g f /	1
+	# +	g f / e *	1
d	# + d	g f / e *	1
*	# + *	g f / e * d	2
c	# + * c	g f / e * d	2
/	# + /	g f / e * d c *	2
b	# + / b	g f / c * d c *	2
-	# -	g f / e * d c * b / +	2
a	# - a	g f / e * d c * b / +	1
#	-	g f / e * d c * b / + a -	1

prefix ⇒ - a + / b * c d * e / f g

v) (((A - (B + C)) * D) $ (E + F)) - postfix

Next ch	con of stack	polish Notation	Rank
(	(	-	-
(	((	-	-
(	(((	-	-
A	(((A	-	-
-	(((-	A	1
(	(((-(	A	1
B	(((-(B	A	1
+	(((-(+	A B	2
C	(((-(+C	A B	2

)	(((-	ABC+	2
)	((	ABC+-	1
*	((*	ABC+-	1
D	((*D	ABC+-	1
)	(	ABC+-D*	1
$	($	ABC+-D*	1
(	($(	ABC+-D*	1
E	($(E	ABC+-D*	1
+	($(+	ABC+-D*E	2
F	($(+F	ABC+-D*E	2
)	($	ABC+-D*EF+	2
)	-	ABC+-D*EF+$	1

6) Consider the following queue, where queue is a circular queue having 6 memory cells.
Front = 2 , Rear = 4
Queue : — A C D — —
Describe queue as following operation take place.

i) F is added
ii) Two letters are deleted
iii) R is added.
iv) S is added.
v) one letter is deleted.

— A C D — —
↑ ↑
F R

i) — A C D F — f is added.
↑ ↑
f R

ii) — — — D F — two are deleted.
↑ ↑
f R

iii) — — — D f R R is added.
↑ ↑
f R

iv) S — — D f R S is added.
↑ ↑
R f

v) S — — — f R one is deleted.
↑ ↑
R f

7) Write an algorithem for evalition of postfix expression & evaluate the following expression showing status of stack in tabular form.

i) 5 4 6 + * 4 9 3 / + *

Char scanned	content of stack	operation on stack	operation.
5	5	push 5	
4	5, 4	push 4	
6	5, 4, 6	push 6	
+	5, 10	pop 6, 4 push 10	6 + 4 = 10
*	50	pop 10, 5 push 50	10 * 5 = 50
4	50, 4	push 4	
9	50, 4, 9	push 9	
3	50, 4, 9, 3	push 3	
/	50, 4, 3	pop 3, 9 push 3	9 / 3 = 3
+	50, 7	pop 4, 3 push 7	3 + 4 = 7
*	350	POP 7, 50 Push 350	50 * 7 = 350

ii) 7 5 2 + * 4 1 1 + / -

char scanned	content of stack	operation on stack	operation.
7	7	push 7	-
5	7, 5	push 5	-
2	7, 5, 2	push 2	-
+	7, 7	pop 2, 5 push 7	2+5=7
*	49	pop 7, 7 push 49	7*7=49
4	49, 4	push 4	
1	49, 4, 1	push 1	
1	49, 4, 1, 1	push 1	
+	49, 4, 2	pop 1, 1 push 2	1+1=2
/	49, 2	pop 4, 2 push 2	4/2=2
-	47	pop 2, 49 push 47	49-2=47

iii) 2 $ 3 + 5 * 2 $ 2 - 6 / 6 ⇒ 2 3 $ 5 2 2 $ * + 6 6 / -

char	content of stack	operation in stack	operation
2	2	push 2	
3	2, 3	push 3	
$	8	pop 2, 3 push 8	2^3=8
5	8, 5	push 5	
2	8, 5, 2	push 2	
2	8, 5, 2, 2	push 2	
$	8, 5, 4	pop 2, 2 push 4	2^2=4

*	6, 20	pop 4, 5 push 20	4 * 5 = 2
+	26	pop 6, 20 push 26	20 + 6 = 26
6	26, 6	push 6	
6	26, 6, 6	push 6	
/	26, 1	pop 6, 6 push 1	6 / 6 = 1
-	27	pop 26, 1 push 27	26 - 1 = 27

9) write an algorithem to perform various operation (insert, delete, display for simple queue.

Q INSERT

1) [overflow]
if R >= N
then write ('OVERFLOW')
Return.

2) [Increment REAR pointer]
R ← R+1

3) [Insert element] Q[R] ← Y

4) [Is front pointer properly set]
if F = 0
then F ← 1 Return

Q DELETE :- 1) [Underflow]
if F = 0
then write ('UNDERFLOW')
Return (0)

2) [Decrement element] y ← a[f]

3) [Queue empty ?]
if f = R
then f ← R ← 0
else f ← f+1

4) [Return element]
Return y

a DISPLAY :-

1) [check for empty]
if f > R
then write ('queue is empty')

2) [Display content]
for (i = FRONT ; i <= REAR ; i++)
write (a[i])

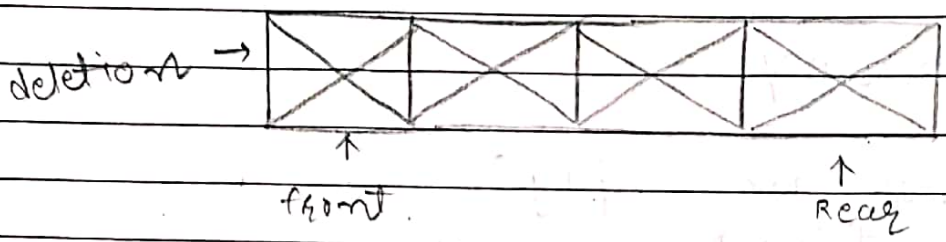
3) [Return statement]
Return.

10) compare simple queue v/s circular queue. write an algorithem to implement insert / delete operation into a circular queue using array representation of queue.

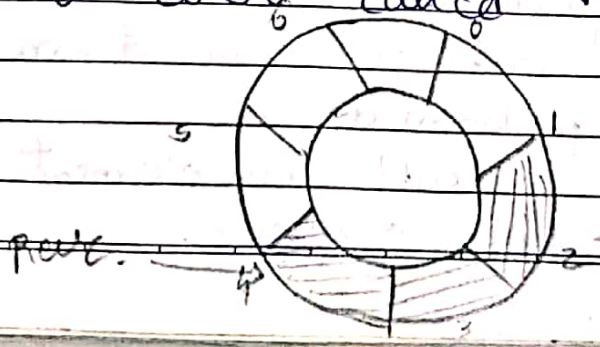
Queue → It follows FIFO pattern
→ front is the end of queue from where deletion is performed.
→ Rear is the end of queue from where insertion is performed.
→ The process to add an element in a queue is called enqueue.

Page No.:
Date: / /

→ The process of removal of an element in a queue is called Dequeue.
→ The familar traditional examples of a queue is checkout line at supermarket cash register where the first person in the line is usually the first to checkout.



Circular queue :→ A more suitable method of representing simple queue which prevents an excessive use of memory to arrange the element Q[1], Q[2] --- Q[n] in a called circular fashion with Q[1] following Q[n], this is called circular queue.
→ circular queue is a linear data structure. It follows FIFO principle.
→ In circular queue, the last node is connected back to the first node to make a circle.
→ circular linked list follow the ~~first~~ FIFO
→ elements are added at the rear end & the elements are deleted from the front end of queue
→ Both the front & the rear pointer point to the beginning of array.
→ It is also called as 'Ring buffer.



consider a circular queue: F is at 2, R is at 4, queue — A C D — —

i) F is added
ii) two letters are deleted
iii) R is added
iv) S is added
v) one is deleted.

— A C D — —
  ↑ F ↑ R

F is added : — A C D F —
  ↑ F ↑ R

two deleted : — — — D F —
  ↑ F ↑ R

R added : — — — D F R
  ↑ F ↑ R

S added : S — — D F R
  ↑ R ↑ F

one is deleted : S — — — F R
  ↑ R ↑ F

6) enlist and briefly explain various application of stack.

→ Recursion
→ keeping track of function calls.
→ evaluation of expression
→ Reversing character.
→ servicing hardware interupts
→ solving combinatorial peoplems using backtracking.

expression emulation.

⇒ stack is used to evalute prefix, postfix, infix

expression conversion

⇒ An expression can be represented in prefix, postfix, notation.

Infix expression: it is the general notion used for representing expression.

prefix expression :- operator are followed by operands

eg. infix :- a + b c

postfix :- a b c +

prefix :- + a b c

All infix operations will be converted into postfix notation with the help of stack in any problem.

parsing :-

⇒ Many compilers use a stack for parsing the syntax of expression program stacks etc. before transfering into the stack and them bestly invoked function will first return the value by poping the stack.

fuction calls

⇒ A stack is useful for the operating system to store local variables used inside a fn block so that they can be discarded once the control comes out of fn block.