PART A solutions ॥ श्री गणेशाय नमः ॥

① Algorithm to delete data from queue and retrieve later

→ 1) check queue is empty or not return queue empty
2) make a temp variable to store data
3) // ~~for~~ temp[ ] = front;
3) increment front by 1. and print msg element deleted
4) return stored data.

② ptr to fxn concept

→ ptr pointing to a particular fxn in code called ptr to fxn.

ex. #include <stdio.h>
~~int main()~~
~~void~~
int sum(int a, int b)
{
return(a+b);
}
int main()
{ int *s;
int (*s)(int, int) = sum;
int r = s(10,20);
pf("%d", r); → 30
}

Dynamic memory management functions.

1) malloc 2) calloc 3) re-alloc 4) free

malloc()

→ used to allocate single block of specified size

ex ptr = (cast type *) malloc (n * sizeof(datatype));

calloc()

→ used to allocate specified memory blocks.

ex ptr = (cast-type*) calloc(n, size);

n: no. of Block

realloc

~~if ptr is~~

→ used to reallocate memory of a pointer by other.

ex. ptr = realloc(ptr, memory to allocate);

free

→ used to delete ptrs.

→ free(ptr)

→ Algo to destroy a queue

1) check given queue is empty or not if its empty → print: queue is empty.
2) if not, ~~increment pointer~~ make a ~~ptr~~ temp variable and store front pointer data in temp. variable
3) increment front by 1 and print element deleted msg
4) in end free(temp);

5) repeat step (3,4) untill front reach end of queue.
6) reset front ~~size~~ to 0
7) end

comparison bet linked list and Array

Array	LL
1. contiguos location	1. various locations.
2. not dynamic	2. dunamic
3. memory allocat at compile time	3. runtime Allocat.
4. use less memory	4. use more memory
5. elements accessed easily	5. cant access easily
6. elemut insertion hard	6. insertion of new elemut is much easy.

what is BT, and its terminology

→ Tree who has atmost 2 child

(1) root → node at top
(2) leaf → no child
(3) internal → atleast one child
(4) degree → no. of children
(5) indegree → no. of nodes pointing given node
(6) outdegree → no. of nodes given node is pointing

explain basic stack operations

```
push(int c)
{ if(top==size-1)
    pf("stack full");
  else
    stack[top++] = c;
}
```

```
delete
pop()
{ if(top==-1)
    pf(empty);
  else
    pf(deleted %d", &stack[top]);
    top--;
}
```

```
display
show()
{ if(top==-1)
    pf("empty");
  else
    for(i=0; i<=top; i++)
    { pf("%d", stack[i]);
    }
}
```