DATA STRUCTURES MID-1 IMPORTANT Q/A

Unit-1

1.What is a data structure? Why do we need data structures and list out some data structures?

A. A data structure is basically a group of data elements that are put together under one name, and which defines a particular way of storing and organizing data in a computer so that it can be used efficiently.

Data structures are used in almost every program or software system. Some common examples of data structures are arrays, linked lists, queues, stacks, binary trees, and graphs.

1.Arrays- An array is a collection of similar data elements. These data elements have the same data type. The elements of the array are stored in continuous memory locations and are referenced by an index.

syntax: datatype name[size];

Example-int marks[10]

Arrays are generally used when we want to store large amount of similar type of data. But they have the following limitations:

1.Arrays are of fixed size.

2.Data elements are stored in contiguous memory locations which may not be always available.

3.Insertion and deletion of elements can be problematic because of shifting of elements from their positions.

2.Linked list- A linked list is a very flexible, dynamic data structure in which elements (called nodes) form a sequential list.

In a linked list, each node is allocated space as it is added to the list. Every node in the list points to the next node in the list. Therefore, in a linked list, every node contains the following two types of data:

1.The value of the node or any other data that corresponds to that node

2.A pointer or link to the next node in the list

The last node in the list contains a NULL pointer to indicate that it is the end or tail of the list.

3.Queues- A queue is a first-in, first-out (FIFO) data structure in which the element that is inserted first is the first one to be taken out. The elements in a queue are added at one end called the rear and removed from the other end called the front. Like stacks, queues can be implemented by using either arrays or linked lists. Every queue has front and rear variables that point to the position from where deletions and insertions can be done, respectively.

If we want to add one more value then the rear would be incremented by 1 and the value would be stored at the position pointed by the rear. If we want to delete an element from the queue, then the value of front will be incremented. Deletions are done only from this end of the queue.

A queue is full when rear = MAX–1, where MAX is the size of the queue, that is MAX specifies the maximum number of elements in the queue.

An underflow condition occurs when we try to delete an element from a queue that is already empty. If front = NULL and rear = NULL, then there is no element in the queue.

4.Stacks- A stack is a linear data structure in which insertion and deletion of elements are done at only one end, which is known as the top of the stack. Stack is called a last-in, first-out (LIFO) structure because the last element which is added to the stack is the first element which is deleted from the stack.

top is used to store the address of the topmost element of the stack. It is this position from where the element will be added or deleted. There is another variable MAX, which is used to store the maximum number of elements that the stack can store. If top = NULL, then it indicates that the stack is empty and if top = MAX–1, then the stack is full.

A stack supports three basic operations: push, pop, and peep. The push operation adds an element to the top of the stack. The pop operation removes the element from the top of the stack. And the peep operation returns the value of the topmost element of the stack (without deleting it).

overflow occurs when we try to insert an element into a stack that is already full.

An underflow condition occurs when we try to delete an element from a stack that is already empty.

5.Trees- A tree is a non-linear data structure which consists of a collection of nodes arranged in a hierarchical order. One of the nodes is designated as the root node, and the remaining nodes can be partitioned into disjoint sets such that each set is a sub-tree of the root.

A binary tree consists of a root node and left and right sub-trees, where both sub-trees are also binary trees. Each node contains a data element, a left pointer which points to the left sub-tree, and a right pointer which points to the right sub-tree. The root element is the topmost node which is pointed by a ‘root’ pointer. If root = NULL then the tree is empty.

6.Graphs-

A graph is a non-linear data structure which is a collection of vertices (also called nodes) and edges that connect these vertices. A graph is often viewed as a generalization of the tree structure, where instead of a purely parent-to-child relationship between tree nodes, any kind of complex relationships between the nodes can exist.

2.Discuss about searching and implement Binary search technique?

A. Searching- It is used to find the location of one or more data items that satisfy the given constraint.

Implementation of binary search-

Binary Search is **a searching algorithm for finding an element's position in a sorted array**. In this approach, the element is always searched in the middle of a portion of an array. Binary search can be implemented only on a sorted list of items. If the elements are not sorted already, we need to sort them first.

```
// Binary Search in C

#include <stdio.h>

int binarySearch(int array[], int x, int low, int high) {
  // Repeat until the pointers low and high meet each other
  while (low <= high) {
    int mid = low + (high - low) / 2;

    if (array[mid] == x)
      return mid;

    if (array[mid] < x)
      low = mid + 1;

    else
      high = mid - 1;
  }

  return -1;
}
```

```
int main(void) {
  int array[] = {3, 4, 5, 6, 7, 8, 9};
  int n = sizeof(array) / sizeof(array[0]);
  int x = 4;
  int result = binarySearch(array, x, 0, n - 1);
  if (result == -1)
    printf("Not found");
  else
    printf("Element is found at index %d", result);
  return 0;
}
```

3. Compare and contrast linear and binary search algorithms.

A.

Basis of comparison	Linear search	Binary search
Definition	The linear search starts searching from the first element and compares each element with a searched element till the element is not found.	It finds the position of the searched element by finding the middle element of the array.
Sorted data	In a linear search, the elements don't need to be arranged in sorted order.	The pre-condition for the binary search is that the elements must be arranged in a sorted order.
Implementation	The linear search can be implemented on any linear data structure such as an array, linked list, etc.	The implementation of binary search is limited as it can be implemented only on those data structures that have two-way traversal.
Approach	It is based on the sequential approach.	It is based on the divide and conquer approach.

Size	It is preferrable for the small-sized data sets.	It is preferrable for the large-size data sets.
Efficiency	It is less efficient in the case of large-size data sets.	It is more efficient in the case of large-size data sets.
Worst-case scenario	In a linear search, the worst- case scenario for finding the element is O(n).	In a binary search, the worst-case scenario for finding the element is $O(\log_2 n)$.
Best-case scenario	In a linear search, the best-case scenario for finding the first element in the list is O(1).	In a binary search, the best-case scenario for finding the first element in the list is O(1).
Dimensional array	It can be implemented on both a single and multidimensional array.	It can be implemented only on a multidimensional array.

4.Write a C program for to sort the given elements using quick sort with an example?

A. A Sorting technique that sequences a list by continuously dividing the list into two parts and moving the lower items to one side and the higher items to the other.

```
//QUICK SORT
#include<stdio.h>
void quicksort(int number[25],int first,int last){
   int i, j, pivot, temp;
   if(first<last){
      pivot=first;
      i=first;
      j=last;
      while(i<j){
         while(number[i]<=number[pivot]&&i<last)
         i++;
         while(number[j]>number[pivot])
         j--;
         if(i<j){
            temp=number[i];
            number[i]=number[j];
```

```
            number[j]=temp;
          }
        }
        temp=number[pivot];
        number[pivot]=number[j];
        number[j]=temp;
        quicksort(number,first,j-1);
        quicksort(number,j+1,last);
    }
}
int main(){
    int i, count, number[25];
    printf("How many elements are u going to enter?: ");
    scanf("%d",&count);
    printf("Enter %d elements: ", count);
    for(i=0;i<count;i++)
    scanf("%d",&number[i]);
    quicksort(number,0,count-1);
    printf("Order of Sorted elements: ");
    for(i=0;i<count;i++)
    printf(" %d",number[i]);
    return 0;
}
```

5.Explain Merge Sort with example.

A. A sorting technique that sequences data by continuously merging items in the list.

```
#include<stdlib.h>
#include<stdio.h>
// Merge Function
void merge(int arr[], int l, int m, int r)
{
int i, j, k;
int n1 = m - l + 1;
int n2 = r - m;
int L[n1], R[n2];
```

```
for (i = 0; i < n1; i++)
L[i] = arr[l + i];
for (j = 0; j < n2; j++)
R[j] = arr[m + 1+ j];
i = 0;
j = 0;
k = l;
while (i < n1 && j < n2)
{
if (L[i] <= R[j])
{
arr[k] = L[i];
i++;
}
else
{
arr[k] = R[j];
j++;
}
k++;
}
while (i < n1)
{
arr[k] = L[i];
i++;
k++;
}
while (j < n2)
{
arr[k] = R[j];
j++;
k++;
}
}
```

6.Sort the following set using selection sort technique:
45, 78, 36, 90, 16, 59, 82, 61

A. #include<stdio.h>

int min(int *arr,int i,int n)

{

int j,ind=i,mini=arr[i];

for(j=i;j<n;j++)

{

```
            if(mini>arr[j])
            {
                mini=arr[j];
                ind=j;
            }
        }
        return ind;
}
void selection_sort(int *arr,int n)
{
        int i,mini,temp;
        for(i=0;i<n;i++)
        {
            mini=min(arr,i,n);
            temp=arr[i];
            arr[i]=arr[mini];
            arr[mini]=temp;
        }
}
int main()
{
        int arr[100],i,n;
        scanf("%d",&n); //8
        for(i=0;i<n;i++)
        {
            scanf("%d",&arr[i]);// 45 78 36 90 16 59 82 61
        }
        selection_sort(arr,n);
```

```
        for(i=0;i<n;i++)
        {
                printf("%d ",arr[i]);//16 36 45 59 61 78 82 90
        }
}
```

7.Define time complexity and space complexity and discuss about big o notation? And specify Time complexity for Linear Search, Binary Search, Quick Sort, Bubble Sort, Merge Sort.

A. Time complexity- The time complexity of an algorithm is basically the running time of a program as a function of the input size.

Space complexity- Space complexity of an algorithm is the amount of computer memory that is required during the program execution as a function of the input size.

Big 'O' notation- The Big O notation is used to express the upper bound of the runtime of an algorithm and thus measure the worst-case time complexity of an algorithm.

Time complexity-Amount of time taken by a set of code or algorithm to process or run as a function of the amount of input. In other words, the time complexity is how long a program takes to process a given input.

Linear search-A sequential search is made over all items one by one. Every item is checked and if a match is found then that particular item is returned, otherwise the search continues till the end of the data collection. The time complexity of linear search **O(n).**

Binary search-Binary search looks for a particular item by comparing the middle most item of the collection. If a match occurs, then the index of item is returned. If the middle item is greater than the item, then the item is searched in the sub-array to the left of the middle item.

The time complexity for binary search **O(log n)**.

Quick sort- Quicksort is a sorting algorithm based on the divide and conquer approach where An array is divided into subarrays by selecting a pivot element (element selected from the array).While dividing the array, the pivot element should be positioned in such a way that elements less than pivot are kept on the left side and elements greater than pivot are on the right side of the pivot. The left and right subarrays are also divided using the same approach. This process continues until each subarray contains a single element. At this point, elements are already sorted. Finally, elements are combined to form a sorted array. The average case time complexity of quicksort is **O(n*logn)**.

Bubble sort- This sorting algorithm is comparison-based algorithm in which each pair of adjacent elements is compared and the elements are swapped if they are not in order. This

algorithm is not suitable for large data sets as its average and worst case complexity are of **O(n2)** where n is the number of items.

Merge sort- It divides the input array into two halves, calls itself for the two halves, and then it merges the two sorted halves. The merge() function is used for merging two halves. The time complexity of MergeSort is **O(n*Log n).**

Unit-2

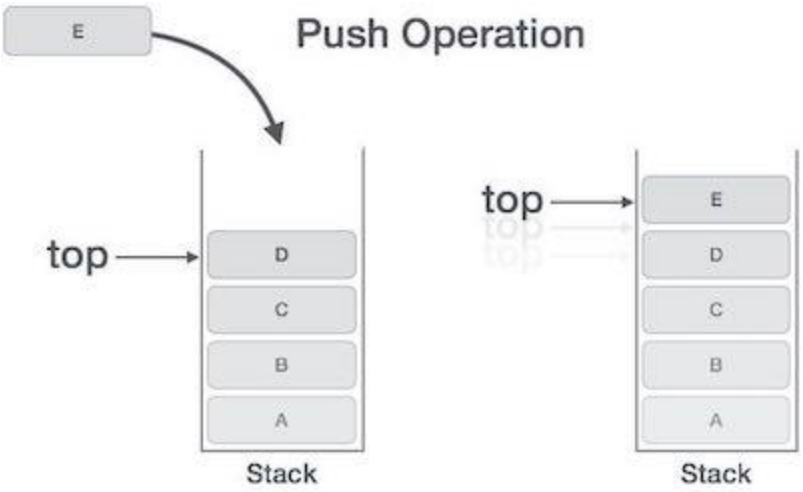
1. Define Stack? What are the operations of the stack? Write the routine to push a element into a stack.

A. A stack is a linear data structure in which insertion and deletion of elements are done at only one end, which is known as the top of the stack. Stack is called a last-in, first-out (LIFO) structure because the last element which is added to the stack is the first element which is deleted from the stack. stacks can be implemented using arrays or linked lists. top is used to store the address of the topmost element of the stack. It is this position from where the element will be added or deleted. There is another variable MAX, which is used to store the maximum number of elements that the stack can store. If top = NULL, then it indicates that the stack is empty and if top = MAX–1, then the stack is full.

Operations on stack:
A stack supports three basic operations: push, pop, and peep. The push operation adds an element to the top of the stack. The pop operation removes the element from the top of the stack. And the peep operation returns the value of the topmost element of the stack (without deleting it). overflow occurs when we try to insert an element into a stack that is already full. An underflow condition occurs when we try to delete an element from a stack that is already empty.

Routine to push an element into stack:
The process of putting a new data element onto stack is known as a Push Operation. Push operation involves a series of steps −

- **Step 1** − Checks if the stack is full.
- **Step 2** − If the stack is full, produces an error and exit.
- **Step 3** − If the stack is not full, increments **top** to point next empty space.
- **Step 4** − Adds data element to the stack location, where top is pointing.
- **Step 5** − Returns success.



2. What are the applications of stack? Specify methods to implement stack in C?

A. Applications of Stack in Data Structure:

1.Evaluation of Arithmetic Expressions-

An arithmetic expression consists of operands and operators. In addition to operands and operators, the arithmetic expression may also include parenthesis like "left parenthesis" and "right parenthesis".

Operators	Associativity	Precedence
^ exponentiation	Right to left	Highest followed by *Multiplication and /division
*Multiplication, /division	Left to right	Highest followed by + addition and - subtraction
+ addition, - subtraction	Left to right	Lowest

Notations for Arithmetic Expression

There are three notations to represent an arithmetic expression:

1.Infix Notation

2.Prefix Notation

3.Postfix Notation

Infix Notation

In infix notation each operator is placed between the operands.

Example: A + B, (C - D) etc.

All these expressions are in infix notation because the operator comes between the operands.

Prefix Notation

The prefix notation places the operator before the operands.

Example: + A B, -CD etc.

All these expressions are in prefix notation because the operator comes before the operands.

Postfix Notation

The postfix notation places the operator after the operands.

Example: AB +, CD+, etc.

All these expressions are in postfix notation because the operator comes after the operands.

2.Backtracking

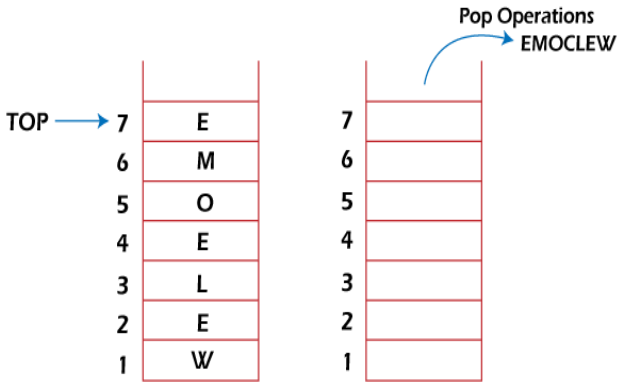
Backtracking is another application of Stack. It is a recursive algorithm that is used for solving the optimization problem.

3.Delimiter Checking

A delimiter is one or more characters that separate text strings. Common delimiters are commas (,), semicolon (;), quotes (", '), braces ({}), pipes (|), or slashes (/ \ ). When a program stores sequential or tabular data, it delimits each item of data with a predefined character.

4.Reverse a Data

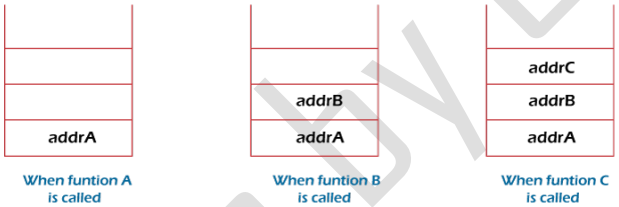
A Stack can be used to reverse the characters of a string. This can be achieved by simply pushing one by one each character onto the Stack, which later can be popped from the Stack one by one. Because of the **last in first out** property of the Stack, the first character of the Stack is on the bottom of the Stack and the last character of the String is on the Top of the Stack and after performing the pop operation in the Stack, the Stack returns the String in Reverse order.



5.Processing Function Calls

A will only be completed after function B is completed and function B will only be completed after function C is completed. Therefore, function A is first to be started and last to be completed. To conclude, the above function activity matches the last in first out behavior and can easily be handled using Stack.

Consider addrA, addrB, addrC be the addresses of the statements to which control is returned after completing the function A, B, and C, respectively.



Different states of stack

Implementation:

There are two ways to implement a stack:

- Using array
- Using linked list

3. Convert Given Expression into Prefix expression.

(a) (A-B) * (C+D)

3 a (A − B) * (C + D)

A. Infix to prefix

→ First reverse the given expression

(D + C) * (B − A)

→ Apply infix to postfix

[O.S → Output String]

STEPS

1) ~~D~~ (

2) D O.S = D

3) + → (+

4) C O.S = D C

5)) O.S = D C +

6) *  (Adding it to stack)

7) (

8) B OS = DC + B

9) (−

10) A OS = DC + BA

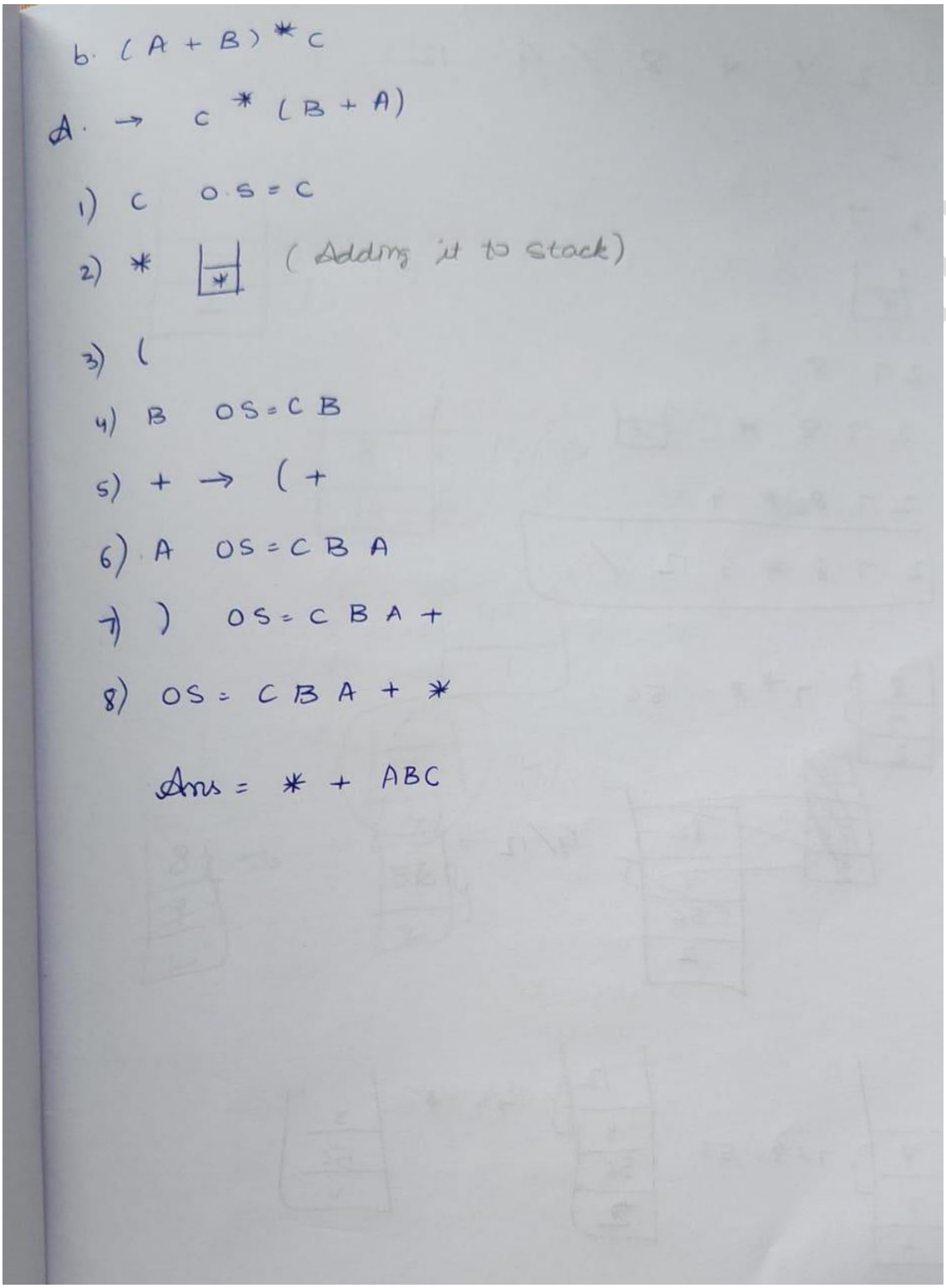
11))

12) OS = DC + BA − *

→ Finally reverse the answer (O.S)

⇒ * − AB + CD

(B) (A+B) * C



4.Define Polish notation. And evaluate postfix expressions using stacks.

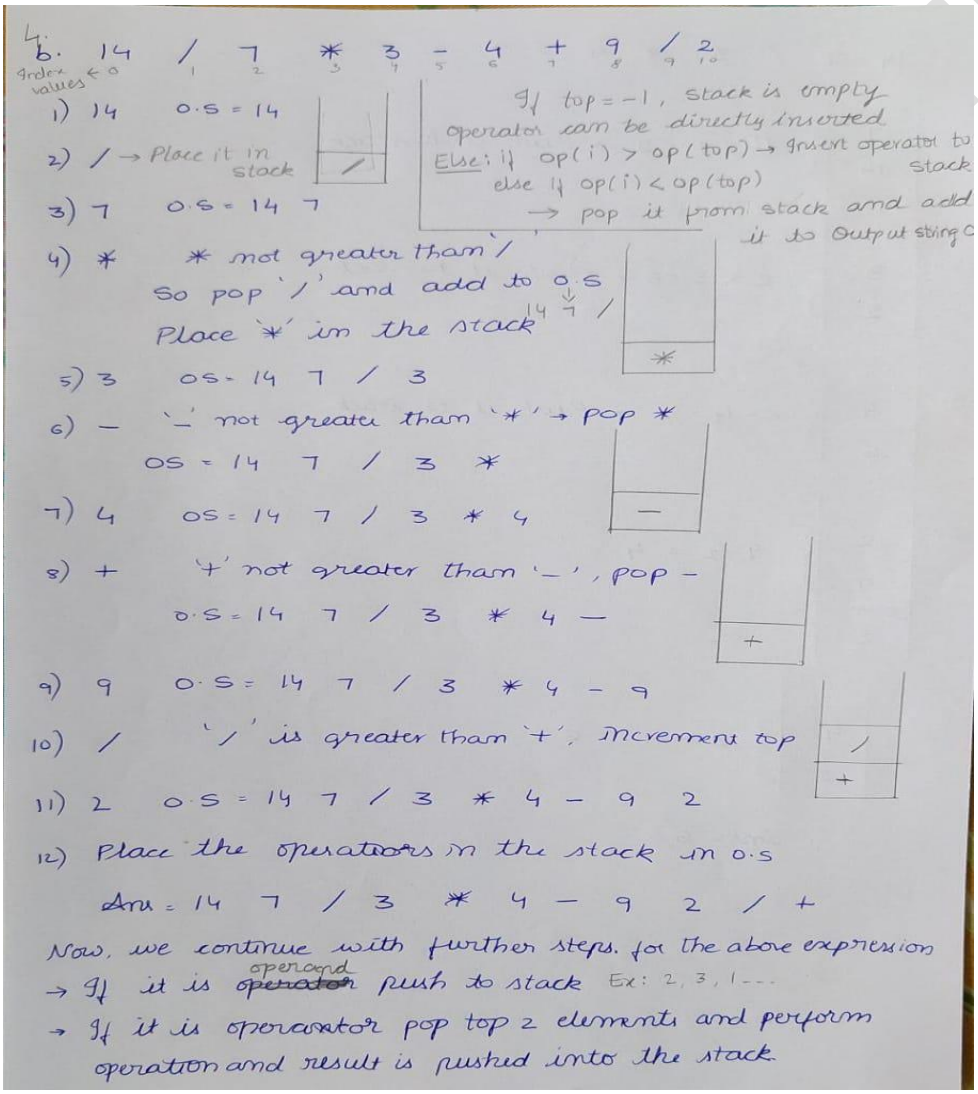
(a) 2 7 * 8 / 4 12

Define polish notation-

Polish notation is a notation form for expressing arithmetic, logic and algebraic equations. Operators are placed on the left of their operands. If the operator has a defined fixed number of operands, the syntax does not require brackets or parenthesis to lessen ambiguity. Polish notation is also known as prefix notation, prefix Polish notation, normal Polish notation, Warsaw notation and Lukasiewicz notation.

We got some errors in this answer so, we can't present it. If possible, we update separately afterwards

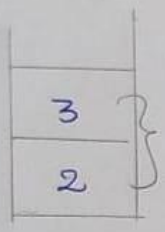
(b) 14 / 7 * 3 – 4 + 9 / 2



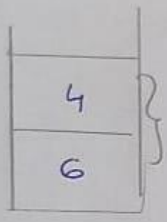
14 7 / 3 * 4 - 9 2 / +

① 

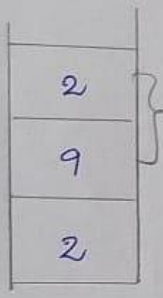
14/7 = 2 → Push it to stack

② 

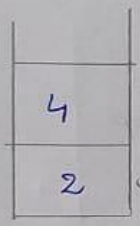
2 x 3 = 6 → Push it to stack

③ 

6 - 4 = 2 → Push it to stack

④ 

9/2 = 4.5

⑤ 

2 + 4.5 = 6.5

∴ Ans = 6.5

5. Define queue. What are the operations of a queue? What are the types of queues? List out Applications of Priority queues.

A. A queue is a first-in, first-out (FIFO) data structure in which the element that is inserted first is the first one to be taken out. The elements in a queue are added at one end called the rear and removed from the other end called the front. Like stacks, queues can be implemented by using either arrays or linked lists.

Operations of a queue-

Peek() Operation

This function helps in extracting the data element where the front is pointing without removing it from the queue. The algorithm of Peek() function is as follows-

- Step 1: Check if the queue is empty.
- Step 2: If the queue is empty, return "Queue is Empty."
- Step 3: If the queue is not empty, access the data where the front pointer is pointing.
- Step 4: Return data.

isFull() Operation

This function checks if the rear pointer is reached at MAXSIZE to determine that the queue is full. The following steps are performed in the isFull() operation -

- Step 1: Check if rear == MAXSIZE - 1.
- Step 2: If they are equal, return "Queue is Full."
- Step 3: If they are not equal, return "Queue is not Full."

isNull() Operation

The algorithm of the isNull() operation is as follows -

- Step 1: Check if the rear and front are pointing to null memory space, i.e., -1.
- Step 2: If they are pointing to -1, return "Queue is empty."
- Step 3: If they are not equal, return "Queue is not empty."

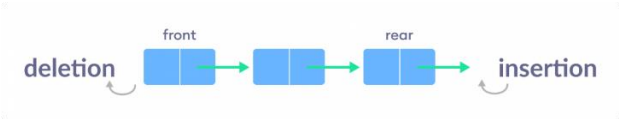
Types of queues:

There are three different types of queues:

- Simple Queue
- Circular Queue
- Priority Queue

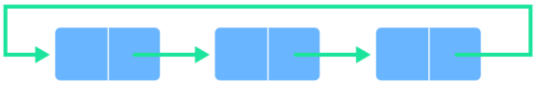
Simple Queue-

insertion takes place at the rear and removal occurs at the front. It strictly follows the FIFO (First in First out) rule.



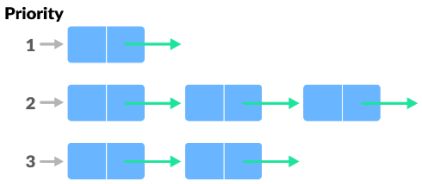
Circular Queue-

last element points to the first element making a circular link.



Priority Queue-

A priority queue is a special type of queue in which each element is associated with a priority and is served according to its priority. If elements with the same priority occur, they are served according to their order in the queue.



6. Define double ended queue? What are the methods used to implement queue in C? List out applications of queue? Distinguish between stack and queue.

A. In a double ended queue, insertion and removal of elements can be performed from either from the front or rear. Thus, it does not follow the FIFO (First In First Out) rule.

Enqueue: Addition of an element to the queue. Adding an element will be performed after checking whether the queue is full or not.

Dequeue: Removal of an element from the queue. An element can only be deleted when there is at least an element to delete i.e. rear > 0.

Front: Get the front element from the queue i.e. arr[front] if queue is not empty.

Display: Print all element of the queue. If the queue is non-empty, traverse and print all the elements from index front to rear.

The following are some of the most common applications of queue in data structure:

- Managing requests on a single shared resource such as CPU scheduling and disk scheduling
- Handling hardware or real-time systems interrupts
- Handling website traffic
- Routers and switches in networking
- Maintaining the playlist in media players

Difference between stack and queue

Stacks	Queues
Stacks are based on the LIFO principle, i.e., the element inserted at the last, is the first element to come out of the list.	Queues are based on the FIFO principle, i.e., the element inserted at the first, is the first element to come out of the list.
Insertion and deletion in stacks takes place only from one end of the list called the top.	Insertion and deletion in queues takes place from the opposite ends of the list. The insertion takes place at the rear of the list and the deletion takes place from the front of the list.
Insert operation is called push operation.	Insert operation is called enqueue operation.
Delete operation is called pop operation.	Delete operation is called dequeue operation.
In stacks we maintain only one pointer to access the list, called the top, which always points to the last element present in the list.	In queues we maintain two pointers to access the list. The front pointer always points to the first element inserted in the list and is still present, and the rear pointer always points to the last inserted element.
Stack is used in solving problems works on recursion.	Queue is used in solving problems having sequential processing.

Unit-3

1.Define Linked list? Discuss about Types of Linked lists?

A. A linked list is a very flexible, dynamic data structure in which elements (called nodes) form a sequential list. In a linked list, each node is allocated space as it is added to the list. Every node in the list points to the next node in the list. Therefore, in a linked list, every node contains the following two types of data:

The value of the node or any other data that corresponds to that node

A pointer or link to the next node in the list

Types of Linked lists-

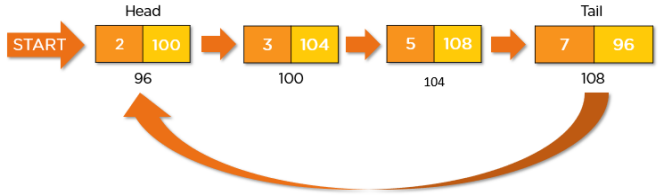
Singly Linked List

A singly linked list is a unidirectional linked list. So, you can only traverse it in one direction, i.e., from head node to tail node.

Doubly Linked List

A doubly linked list is a bi-directional linked list. So, you can traverse it in both directions. Unlike singly linked lists, its nodes contain one extra pointer called the previous pointer. This pointer points to the previous node.

Circular Linked List



A circular Linked list is a unidirectional linked list. So, you can traverse it in only one direction. But this type of linked list has its last node pointing to the head node. So while traversing, you need to be careful and stop traversing when you revisit the head node.

2.Explain about how operations are performed on single linked list?

A. Ina Singly Linked List, the following operations are done:

- Insertion
- Deletion
- Display

Insertion

The procedures below can be used to add a new node at the start of singly linked list:

Step 1: Generate a newNode using the supplied value.

Step 2: Verify that the list is empty (head == NULL).

Step 3: Set newNode→next = NULL and head = newNode if it's empty.

Step 4: Set newNode→next = head and head = newNode if it is not empty.

Deletion

Deleting First Node of Singly Linked List

To delete a node from the single linked list's beginning, apply the procedures below:

Step 1: Check to see if the list is empty (head == NULL).

Step 2: If the list is empty, display the message 'List is Empty! Deletion is not feasible!', and the function is terminated.

Step 3: If it is not empty, create a temp Node reference and initialize it with head.

Step 4: Verify that the list only has one node (temp next == NULL).

Step 5: If TRUE, change head to NULL and remove temp (Setting Empty list conditions).

Step 6: If the answer is FALSE, set head = temp next and delete temp.

Display

The components of a single linked list can be displayed using the methods below:

Step 1: Determine whether or not the list is empty (head == NULL).

Step 2: If the list is empty, show "List is Empty!!!" and exit the method.

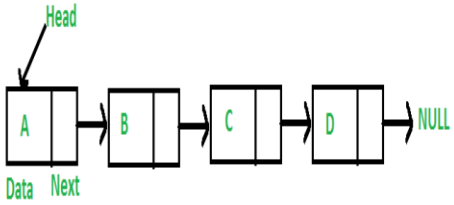
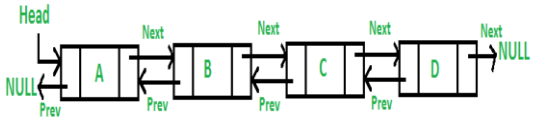
Step 3: If it is not empty, create a 'temp' Node pointer and initialize it with head.

Step 4: Continue to display temp →data with an arrow (—>) until the temp reaches the last node.

Step 5: Finally, show temp →data with an arrow pointing to NULL (temp →data —> NULL).

3.Differentiate Single and double linked list? And Discuss about Applications of linked list.

A.

Singly linked list (SLL)	Doubly linked list (DLL)
SLL nodes contains 2 field -data field and next link field.	DLL nodes contains 3 fields -data field, a previous link field and a next link field.
	
In SLL, the traversal can be done using the next node link only. Thus traversal is possible in one direction only.	In DLL, the traversal can be done using the previous node link or the next node link. Thus traversal is possible in both directions (forward and backward).
The SLL occupies less memory than DLL as it has only 2 fields.	The DLL occupies more memory than SLL as it has 3 fields.
Complexity of insertion and deletion at a given position is O(n).	Complexity of insertion and deletion at a given position is O(n / 2) = O(n) because traversal can be made from start or from the end.
Complexity of deletion with a given node is O(n), because the previous node needs to be known, and traversal takes O(n)	Complexity of deletion with a given node is O(1) because the previous node can be accessed easily
We mostly prefer to use singly linked list for the execution of stacks.	We can use a doubly linked list to execute heaps and stacks, binary trees.

Singly linked list (SLL)	Doubly linked list (DLL)
When we do not need to perform any searching operation and we want to save memory, we prefer a singly linked list.	In case of better implementation, while searching, we prefer to use doubly linked list.
A singly linked list consumes less memory as compared to the doubly linked list.	The doubly linked list consumes more memory as compared to the singly linked list

Applications of linked list-

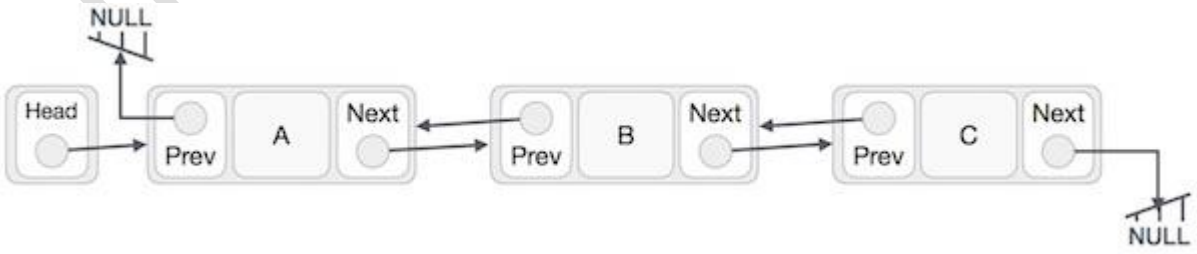
- Symbol table creation
- Mailing list
- Memory management
- Linked allocation of files
- Multiple precision arithmetic etc.

4. Explain the concept of doubly linked list with an example.

A. **D**oubly **L**inked **L**ist (DLL) contains an extra pointer, typically called *previous pointer*, together with next pointer and data which are there in singly linked list.

- **Link** − Each link of a linked list can store a data called an element.
- **Next** − Each link of a linked list contains a link to the next link called Next.
- **Prev** − Each link of a linked list contains a link to the previous link called Prev.
- **LinkedList** − A Linked List contains the connection link to the first link called First and to the last link called Last.

Doubly Linked List Representation



following are the important points to be considered.

- Doubly Linked List contains a link element called first and last.
- Each link carries a data field(s) and two link fields called next and prev.
- Each link is linked with its next link using its next link.
- Each link is linked with its previous link using its previous link.
- The last link carries a link as null to mark the end of the list.

Basic Operations

Following are the basic operations supported by a list.

- **Insertion** − Adds an element at the beginning of the list.
- **Deletion** − Deletes an element at the beginning of the list.
- **Insert Last** − Adds an element at the end of the list.
- **Delete Last** − Deletes an element from the end of the list.
- **Insert After** − Adds an element after an item of the list.
- **Delete** − Deletes an element from the list using the key.
- **Display forward** − Displays the complete list in a forward manner.
- **Display backward** − Displays the complete list in a backward manner.

Example-

A music player which has next and previous buttons.

-by Crazyhumanbeing

.....This paper may or may not contain ERRORS so please read CAREFULLY.....

Write matter(length) with respective to marks.