① Assignment :

Describe the concept of Abstract data type (ADT) and how they differ from concrete data structures. Explain how ADTs are implemented in C++, including operations like push, pop, top, isEmpty, isFull and Peek.

Sol:-

Abstract :

An Abstract Data type (ADT) is a theoretical model that defines a set of operations and the semantics of those operations on a data structure, without specifying how the data structure should be implemented. It provides a high level description of what operations can be performed on the data and what constraints apply to those operations.

Characteristics :-

* Operations :- Defines a set of operations that can be perfo on data structure.
* Semantics :- Specifies the behaviour of each operation.
* Encapsulation :- Hides the implementation details, focusing on the interface provided to the user.

ADT for Stack :-

A stack is a fundamental data structure that follow the Last In First Out (LIFO) Principle. It supports the following operations.

Push:- Adds an elements to the top of the Stack.
Pop:- Removes and returns the element from the top of the Stack.
Peek:- Returns the element from the top of the Stack with -out removing it.
Is Empty:- Checks if the Stack is empty.
Is full:- Checks if the Stack is full.

The implementations using arrays and linked lists are ecitic ways of implementing the Stack ADT in C.

focuses on the operations and their behaviour, e Concrete data structures focus on how these tions are realized using specific Programming ucts (arrays are linked list).

arting the ADT from its implementation, you modularity, encapsulation, and flexibility in and using data structures in programs. This allows for easier maintenance, Code reuse, raction of the Complex operations.

## Implementation in C using Arrays:-

```
#include <stdio.h>
#define Max_Size 100
typedef struct {
    int Items[max_size];
    int top;
} StackArray;
int main() {
  Stack Array Stack;
  Stack.items[++stack.top] = 10;
  Stack.items[++stack.top] = 20;
  Stack.items[++stack.top] = 30;
if (stack.top != -1) {
  Printf("Top element: %d \n", stack.items[stack.top]);
}else {
  Printf("Stack is empty! \n");
}
if (stack.top != -1) {
  Printf("Popped element: %d \n", Stack.items[stack.top--]);
}else {
  Printf("stack underflow! \n");
}
if (stack.top != -1) {
  Printf("Popped element: %.d \n", stack.items[stack.top--]);
}else {
  Printf("Stack underflow: \n");
}
```

```
if (stack.top != -1) {
    Print ("Top element after pops: %d\n", stack.item
        .top]);
}else {
    Print("Stack is empty!\n");
}
return 0;
}
```

## Implementation in C using linked list:

```
#include<stdio.h>
#include<stdio.h>
typedef struct Node{
    int data;
    struct Node* next;
} Node;
int main(){
    Node* top = NULL;
    Node* newnode = (Node*)malloc(sizeof(Node));
if (newnode == NULL) {
Printf("Memory allocation failed!\n");
    return 1;
}
newNode -> data = 10;
new Node -> next = top;
top = newNode;
newNode = (Node*) malloc (size of (Node));
if (newNode == NULL) {
    Printf("Memory allocation failed\n");
```

```
return 1;
}
newNode -> data = 30;
newNode -> next = top;
top = newNode;
if (top != NULL){
Printf("Top element : %d\n", top -> data);
}else {
  Printf("Stack is empty :\n");
}
if (top != NULL){
  Node* temp = top;
Printf("Popped element : %d\n", temp -> data);
  top = top -> next;
 free (temp);
}else{
  Printf("Stack underflow :\n")
if (top != NULL) {
   Printf("Top elements after pops : %d\n", top -> data);
}else{
  Printf("Stack is empty!\n");
}
while (top != NULL){
   Node* temp = top;
   top = top -> next;
   free (temp);
}
 return 0;
}
```

(8) The university announced the selected candidates register number for placement training. The student xxx, reg no. 20142010 wishes to check whether his name is listed or not. The list is not sorted in any order. Identify the searching technique that can be applied and explain the searching steps with the suitable Produre. List includes 20142005, 20142033, 20142011, 20142017, 20148010, 20148056, 20145003.

Linear Search:-

Linear Search work by checking each element in the list one by one until the desired element is found or the end of the is reached. It's a simple searching technique that doesn't require any pair sorting of the data.

Steps for linear Search:-

- Start from the first element
- Check if the current element is equal to the target element.
- If the current element is not the target, move to the next element in the list.
- Continue this process until either the target element is found or you reach the end of the list.
- If the target is found, return its position. If the end of the list is reached and the element has not been found, indicate that element is not present.

Procedure:-

Given the list:

-) Start at the first element of the list.

-) Compare '20142010' with '20142015' (first element), (second element), (fourth element) these are not equal.

-) Compare '20142010' with '20142010' (fifth element). They are equal.

-) The element '20142010' is found at the fifth position Index in the list.

```
#include <stdio.h>
int main() {
    int regnumbers[] = {20142015, 20142033, 20142011, 20142017, 20142010,
                        20142056, 20142003};
    int target = 20142010;
    int n = (sizeof numbers / sizeof regnumbers[0]);
    int found = 0;
    int i;
for (i=0; i<n; i++) {
    if (regnumbers[i] == target) {
        printf ("Registraction number found at index %d\n", target[i]);
    found=1;
    break; }}
if (! found){
    Printf(" Registration number %d not found in list\n", target
    }
return 0; }.
```

Explanation of the Code:

- The 'regnumbers' array contains the list of registration numbers.
- 'target' is the registration number we are searching for.
- 'n' is the total number of element in array.
- Iterate through each element of the array.
- If the current element matches the target, print its index and set the found flag to '1'.
- If the loop completes without finding the target, print that the registration number is not found.

Output:- Registration number 20142010 found at Index 4.

(3) Write Pseudocode for stack operations?

A) 1.) Initializestack();

Initialize necessary variables & structure to represent the stack.

2) Push (elements);

if stack is full:

Print "stack overflow"

3.) pop():-

if stack is empty:

Print ("stack underflow");

return null (or oppropriate error value)

@lse:

remove and return element from the top of the stack decrement end pointer.

5) Peek():

If Stack is empty:

Print "Stack is empty".

return null (appropriate error value)

Else:

return element at the top of the stack (without removing it, is empty():

return true if top is -1 (stack is empty).

Otherwise, return false.

6) is full:-

return true, if top is equal to maxsize-1 (stack is full)

otherwise, return false.

Explanation of the Pseudocode:-

- Intializes the necessary variables or data structures to represented a Stack.
- Adds an element to the top of the stack. Checks if the stack is full before pushing.
- Removes and returns the element from the top of the stack.

* Return the elements at the top of the Stack without removing it.

* Check if the stack is full by Comparing the top Pointer or equivalent variable to the maximum Size of the Stack.