Data Structure: NAME:- G. Rahul
Assignment - II: Regno: 192311244

1 Describe the concept of Abstract data type (ADT) and how they differ from concrete data structures. Design an ADT for a stack and implement it using arrays and linked list in C. Include operations like PUSH, POP, PEEK, isEmpty, or other structures.

Abstract data type (ADT):

An abstract data type (ADT) is a model for data structures that defines the data and the operations that can be performed on the data without specifying how the data will be search or how the operations will be implemented.

Differences between ADTs and Concrete Data Structures:

* Abstract vs Implementation
* Interface vs Realization
* Flexibility
* Encapsulation

Stack - ADT:

A Stack is a linear data structure that follows the last in, first out (LIFO) principle. The primary operations are:

* PUSH: Add an element to the top of the Stack.
* POP: Remove the top element from the Stack.
* PEEK: Retrieve the top element without removing it.

ISEMPTY: Check if the Stack is empty

ISFULL: Check if the Stack is full (scala
-vant for array-based implementation)

Advantages of ADT:

By separating the ADT from its implementation, you achieve modularity, encapsulation, and flexibility in designing and using data Structures in programs.

Implementation in C using linked list:

```
#include <stdio.h>
#include <stdlib.h>
typedef Stouct Node {
        int data;
      Stouct node *next;
} node;
int main() {
      Node *top = NULL;
      Node *newNode = (Node*) mallac
                        (sizeof(node));
if (newnode == NULL) {
   printf ("memory allocation");
   return 1;
}
new node -> data = 10;
new node -> next = top;
top = top->next
    Free (temp);
} else {
   printf ("Stack overflow").
if (top != null) {
     printf ("top element after pops:");
} else {
   print ("Stack is empty");
```

```
}
while (top != null)
    Node *temp = top;
    top = top->next;
    free (temp);
}
return 0;
}
```

Implementation in C Using Arrays:

```
#include <stdio.h>
#define MAX_SIZE 100
typedef struct {
    int items [MAX_SIZE];
    int top;
} StackArray;
int main() {
    StackArray stack;
    stack.top = -1;
    stack.items [++stack.top] = 10;
    stack.items [++stack.top] = 20;
    stack.items [++stack.top] = 30;
if (stack.top != -1) {
    printf ("Top element");
} else {
    printf ("Stack is empty");
}
if (stack.top != -1) {
    printf ("Popped element")
} else {
    printf ("Stack Underflow");
}
if (stack.top != -1) {
    printf ("Popped element");
} else {
    top = newnode;
```

```
newnode = (Node*) malloc (Size of (Node));
if (newnode = NULL) {
    Printf ("Memory allocation");
    return1;
  }
  new node -> data = 80;
  new node -> next = top;
  top = newnode;
  newnode = (node *) malloc (Size of (Node));
  if (new Node == null) {
     printf ("Memory allocation");
    return 1;
   }
newnode -> data = 30;
 newnode -> next = top;
 top = newnode;
if (top != null) {
    printf ("Top element")
  } else {
  printf ("Stack is empty");
  }
  if (top != null) {
    Node *temp = top;
    printf ("popped element");
  }
  if (stack - top1 = -1) {
    printf ("Top element");
  } else {
    printf ("Stack is empty");
  }
 return 0;
}
```

5) The University announced the selected candidate register number for placement training. The student xxy, 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 append by explain the searching steps with suitable procedure. List includes 20142015, 20142033, 20142011, 20142017, 20142056, 20142003.

Steps to implement Linear Search:-

1. Initialize the list: define an array in that contains registration numbers.
2. Specify the target: define the registration number you want to search
3. Iterate through list: use a loop to iterate through each element in array
4. Comparison: Check if the current element contains matches the target registration numbers.

Code:-

```
#include <stdio.h>
#define Size 7
int main() {
    int reg_nums[Size] = {20142015, 20142033,
    20142011, 20142017, 20142010, 20142003};
    int target = 20142010;
    int i;
    for (i=0; i<Size; i++) {
        if (reg_numbers[i] == target) {
        printf("Registration number %d is found",
                    target);
                return 0;
            }
    }
```

used To Reduce the

```
printf("Registration number %d is not
        found", target);
return 0;
}
```

Output:- Registration number 20163010 is found.