Assignment - 1

Name :- CH. Aravind

Reg No :- 192325093

Course Code :- CSA0389

Course Name :- Data Structure for stack overflow.

Describe the Concept of ADT and how they differ from concrete data structure. Design ADT for a stack & implement it using array & linked list in C.

(A) Abstract Data Type (ADT) :- An Abstract data type is a theoretical model that defines a set of operations and the semantics of those operations on a data structure, without specifing how the data structure should be implemented. It provideds a high level description of what operations can be performed on the data and what constrains apply to those operations.

Characteristics of ADTs :-

Operations : Defines a set of operations that can be performed on the D.S

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 D.S that follows the last. In, first out (LIFO) principal. It supports the following operations.

Concrete D.S :- The implementing using arrays & linked lists and specific ways of implementing the stack ADT in 'c'.

How ADT differ from Concrete D.S :- ADT focuses on operations and their behaviour, while concrete d.s focuses on how those operations are realized using a programming constructs (arrays & linked lists).

ADVANTAGES of ADT :- By seperating the ADT from its implimentation you achive modularity, encapulsation & flexibility in designing and using d.s in programs.

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 ;
"          ["          "]" = 30 ;
"          ["          "]" = 40 ;
if (Stack.top != -1) {
    printf ("Top element : %d ", Stack.item[stack.top]) ;
```

```
else {
printf(" stack is empty");
}
if (stack.top >= -1) {
    printf(" Popped element : %d " stack.items [stack.top--]);
}else {
    printf (" stack underflow") ;
} if (stack.top != -1){
    printf(" popped element : %d " stack.item[stack.top--]);
}else {
   top = new Node;
new Node = (Node*) malloc (size of (Node)) ;
    if (new Node == Null) {
    printf (" Memory allocation faiked")
    return 1;
}
 new Node -> data = 20;
 NewNode -> next = top;
 top = newNode;
$newnode = (Node*) malloc (size of (Node)) ;
 if (newnode == Null) {
printf (" Memory allocation failed")
 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 %d", temp -> data);
top = top -> next;
    free (temp);
} else {
 printf ("Stack underflow");
if (top != NULL) {
   printf ("Top element %d", top -> data);
} else {
   printf ("stack is empty");
while (top != NULL) {
  Node* temp = top;
  top = top -> next;
  free (temp);
}
return 0;
```

Explanation of Pseudocode :-

① Initialize the neccessary variable or d.s to represent a stack.

② Adds an element to the top of the stack.

③ Removes and returns the element from the top of the stack.

④ Returns the element at the top of the stack without removing it.

⑤ checks if the stack is empty by inspecting the top pointer (or) equivalent variable.

⑥ checks if the stack is full by comparing the top pointer (or) equivalent variable to the maximum size of the stack.

Push :- Adds an element to the top of the stack.

POP :- Removes an elements & return the element from the top of the stack.

Peek :- Returns the elements from the top of the stack without removing it.

is Empty :- checks if the stack is empty.

is full :- checks if the stack is full.

Implemantation in C using linked list :

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node {
    int data;
    struct Node * next;
} Node;
int main() {
    Node* newNode = (Node*) malloc (sizeof (Node));
if (newnode == NULL) {
    printf ("Memory allocation ");
    return 1;
}
NewNode -> data = 10;
NewNode -> next = top;
otherwise, return false.
isfull;
return true,
if (top == MaxSize -1)
    printf (" Stack full ")
else
    paint ("stack empty")
return 0;
}
```

cedure :-

ven the list : 20142015, 20142033, 20142011, 20142017, 20142056, 20142023

Start at the first element of the list.

② Compare '20142010' with '20142015' (first element) '20142033' (second element), '20142011' (third element) these are not same.

③ Compare 20142010 with 20142010 (fifth element), they are equal.

④ The element '20142010' is found at the fifth element position index in (4) list.

C Code for Linear Search :-

```
#include<stdio.h>
int main() {
int regNum[] = {20142015, 2014, 2033, 20142011, 20142017,
                20142010; 20142056, 20142033};
 int target = 20142010;
 int n = sizeof(regNum/sizeof(regNum[0]));
 int found = 0;
 int i;
for (i=0; i<n; i++) {
 if (regNum[i] == target ) {
printf("Registration Num %d found at index %n" target,i);
 found =1;
```

```
        break;
    }
}
if (!found) {
    printf("Registration Num %d not found in list.\n", target);
}
return 0;
}
```

Explanation :-

① The 'regNum' array contains the list of registration num.
② Target is the registration Number we are searching for.
③ 'n' is the total number of elements in array.
④ Iterate through each element of the array.
⑤ If the element matches the target print its index & set the found flag to '1'.
⑥ If the loop completes without finding the target, print that the registration number is not found.

output :- Registration Number 20142010 found at index 4.