Assignment-1

Name: S.Niharika

Course code: CSA0389

Course name: Data Structure for stack Overflow

Register number: 192311176

Date: 31-07-24

① Describe the concept of Abstract data type (ADT) and how they differ from concrete data structures. Design on ADT for a stack and implement it using arrays and linked list in C. Include operations like Push, Pop, Peek, is empty, is full and Peek

Abstract Data Type (ADT):-

An abstract data type is a theoretical model that defines a set of operations and the semantics (behaviour) 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 of ADTS:-

- operations: defines a set of operations that can be performed on the data structures
- 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 follows the last in first out (LIFO) principal. It supports the following operations

- Push : Adds an element 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 without removing it
- is empty : checks if the stack is empty
- is full : checks if the stack is full

Concrete Data structures :-

The implementations using arrays and linked lists are specific ways of implementing ADT inc

How ADT differ from concrete data structures :-

ADT focuses on the operations and their behaviour, while concrete data structures focus on how those operations are realized using specific programming constructs

Advantages of ADT :-

By separating the ADT from its implementation, you achieve modurality, encapsulation, flexibility in the designing and using data structures 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;
        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{
    top=newNode;
newNode = (Node* ) malloc (sizeof (Node));
if (newNode == NULL) {
    Printf (" Memory allocation failed \n *);
    return 1;
}
   newNode → data=20;
   new Node → Next=top;
    top = newNode;
newNode = (Node* )malloc (sizeof (Node));
 if (newNode==NULL){
      Printf (" Memory allocation failed! \n") ;
return 1;
}
   newNode → data = 30;
   newNode → next=top;
    top = newNode;
  newNode = (Node*)malloc (Sizeof (Node));
  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);
```

```
newNode->data = 10;
newNode->next = top;
top = newNode;
newNode = (Node*) malloc (sizeof (Node));
if (newNode == NULL){
        Printf ("Memory allocation failed \n");
        return 1;
}
newNode->data = 20;
newNode->next = top;
top = newNode;
newNode = (Node*)malloc (sizeof (Node));
if (newNode == NULL) {
        Printf ("Memory allocation failed: \n");
return 1;
}
  newNode->data = 30;
  newNode->next = top;
   top = newNode;
if (~~ne~~top != NULL) &
        Printf ("stack is empty: \n");
} else {
      Printf ("Top element: %d\n", top->data);

if (top != NULL){
            Node* temp = top;
            Printf ("Popped element : %d \n", temp->data);
```

```
Printf ("stack underflow:\n");
}
if (stack.top1==-y){
        Printf(" Top element after PoPs:%d\n", stack.items
                [stack.top2];
}else{
        Printf ("stack is empty:\n");
}
  return 0;
 }
```

Implementation in C using linked list :-

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

Procedure :-

Given the list :

'20142015, 20142033, 20142011, 20142017, 20142010, 20142056, 20142003'

1) start at the 1st element of the list

2) compare '20142010' with '20142015' (1st element), '20142033' (2nd element), '20142011' (3rd element), '20142017' (4th element) these arenot equal

3) compare '20142010' with '20142010' (5th element). They are equal

4) The element '20142010' is found at the 5th position in list

code for linear search :-

```
#include <stdio.h>
int main()
int regNumbers[] = {20142015, 20142033, 20142011, 20142017,
                    20142010, 20142056, 20142003}
int target = 20142010;
int n = sizeof(regNumbers) / sizeof(regNumbers[0]);
int found = 0;
int i;
for(i=0; i<n; i++){
        if(regNumbers[i] = target){
Printf("Registration number %d 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 :-

1) The 'regnumbers' array contains the list of registration numbers
2) 'target' is the reg. number we are searching for
3) 'n' is the total no. of elements in array
4) Iterate through each element of the array
5) If the current element matches the 'target', print its index and set the 'found' flag to '1'.
6) If the loop completes without finding the target, print that the registration number isn't found

output :- Registration number 20142010 found at index 4.

The University announced the selected candiates 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 procedure. List includes 20142015, 20142033, 20142011, 20142017, 20142010, 20142056, 20142003

Linear Search:-

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

Steps for linear search:-

1) start from the 1st element
2) check if the current element is equal to the next
3) If the current element is not the target, move to the next element in the list
4) Continue this process until either the target element is found or you reach the end of the list
5) If the target is found, return its position. If the end of the list is reached and the element hasnot been found, indicate that element is not present

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

write pseudocode for stack operations

1) Intialize stack():

Intialize necessary variable or structures to represent the stack

2) Push (elements) :-

```
if (stack is full):
  Print "stack overflow"
else :
    add element to the top of the stack
    increment top pointer
```

3) Pop ():

```
if stack is empty:
    Print ("stack underflow")
    return null (or appropriate error value)
else:
    remove and return element from top of stack
    decrement end pointer
```

4) Peek ():

```
if stack is empty:
    Print ("stack is empty")
    return null (or appropriate error value)
else :
    return element at the top of stack
```