ASSIGNMENT - 02

B. Ravi Kumar

192311246

CSA0689

DATA STRUCTURES.

(1) Describe the concept of abstract datatype (ADT) and how they differ from concrete datatype structures. Design an ADT for a stack and implement it using arrays and linked listing. Include operations like Push, Peek, Pop, is Empty, is full, and Peek.

Abstract data type (ADT):

An abstract data type is a there theoretical model that differs a set of operation and the semantics those operations on a data structure, without specifying how the data structure without specifying how the data structure should be implemented.

Characteristics of ADT's:

- operations: Defines a set of operations that can be performed on data structure.
- Semantics: specifies the behaviour of each operation.
- Encapsulation: Hides that implemented focusing on the interface provided to user.

ADT for stack:

A stack is fundamental data structure. data follows in first, the following operations.

- PUSH : Adds an element to top of stack.
- POP : Removes & returns the element from the top to bottom.
- PEEK : Returns the Element from the top of the stack.
- EMPTY : checks if the stack is Empty.
- Full : checks if stack is full.

Concrete data structure :

The implementation using arrays and linked List are specific ways of implement the stack in ADT.

Difference :

ADT focuses on operations and their behaviour, while data structures focus on howthose operations are realised using specific programme construction on linked lists.

Advantages of ADT :

By separating the ADT from its implementing you acheive modulation, encapsulation and flexibility in designing and using the data structures in programs. this encapsulation allows for easier maintainance of its complex operations.

```
newnode -> data = 10;
newnode -> next = top;
```

Implementation in C using arrays:

```
#include <stdio.h>
# define max.size 100
  type def stuck {
        Pnt  items [max-size];
       int  top;
}
  Stack main () {
       Stack array stuck;
        stack.top = -1;
       stack . items [++stack.top] = 10;
       stack . items [++ stack .top] = 20;
        stack . items [++stack.top] = 30;
if (stack . top 1 = -1) {
       Print ("TOP element : % d'/n". stack.items [stack top];
} else {.
       Print ("stack is empty : \n");
} else {.
       Print ("stack underflow : \n");
};if (stack . top! = -1) {.
```

```
    printf ("stack underflow : \n");
}
    if (stack . c == -1) {
        printf ("TOP element after POP : %d "n stack :
    } else {
        Print (" stack is empty : \n");
    }
        Print f (" stack is empty).
    } return 0;
    }.
```

Implementation in c using Linked list :

```
# include <stdio.h>
# include <stdio.h>
    type def struct node {
            int data ;
                struct node * next ;
    } Node ;
    int main () {
            node * top = Null ;
        Node * New node = [Node*] malloc(size of (node)) :
if (New node == Null) {
        Printf (" memory allocation. failed d \n");
        return ;
}
```

```
} else {
    top = newnode;
    newnode = (node.*) malloc (size of node);
    if (newnode = Num) {
        Print ("Memory allocation failed \n");
    return 1;
    newnode → data = 20;
    newnode → next = top;
    top = new node.
    new node = (node*) malloc (size of Node);
if (newnode = null) {
    Print ("Memory allocation failed :\n").
return 1;
    new node → data = 30;
    newnode → next = top;
    top = new node;
if (top != null) {
    Print ("TOP element : %d \n" top → data).
}
else {
    Print f ("Stack is empty : \n");
```

```
    TOP = top → next
    Free (temp);
} else
  {
    Print ("Stack underflow : \n");
if (top! = Null){
   Print ("TOP element after POP);
}
 Else..
 { Printf ("Stack is Empty ! \n");
   }
   while (top! = Null)
       {
         Node * temp = top;
         top = top → next
          Free (temp);
   }
   return 0;
}
```

Linear Search :

Linear search work. each element in the list one by one. until the desired element is found or the end of the list is reached it is a simple technique that doesn't require only data.

Steps for linear search :

1. start from first element.
2. check if Current element is equal to target element.
3. If the Current element is not target, element is found or you reach in list.
4. continue this Process until target element is found on reach up of list.

Procedure :

Given the list :

'2014 2015, 2014, 2023, 2014 2017, 2014, 2010, 2014, 2056, 2014 200.

→ start at list element of list

→ compare. '2014 2010' with '2014 2015' and '2014 2033'.

→ Compare '20142010' with '20142010' they are equal.

→ The element '20142010' is found at 8th position.

C code for Linear search:

```
#include <stdio.h>
int again() {
    int regnumbers[] = {2014, 2013, 2014, 2023,
    2014, 2017, 20142010, 20142050, 20142003};
    int target = 20142010;
    int n = sizeof(regnumbers) / sizeof(regnum
                                        (0)};
    int found = 0;
    int i;
    for (i=0; i<n; i++) {
        if (regnumbers[i] == target) {
Printf ("Registration number is found).
        found = i;
            break;
    }
}
if (i=found) {
    in list Print ("Reg num")
}
return 0;
}
```

Write Pseudocode of stack operation.

1) Initialize stack. Initialize necessary variable & structure to replacement the stack.

2) Push element :

If stack is full :

Print " stack overflow ".

else :

add element to top of stack.

increment top pointer.

3) Pop() :

if stack is empty :

Print (" stack overflow ") :

return and return element from top of stack decrement and pointer.

4) Peek () :

if stack is empty :

Print " stack is empty "

return null.

else :

return element at top of stack.

5) Empty ():

Return true if top is -1. [stack is Empty]

otherwise, Return False.

6) Full ():

Return True, if top is equal to axis a<-1.

(Stack is full).

otherwise, Return False.

Explanation of Pseudocode:

- Initialize the necessary variable is or data structure to represent stack.
- Adds an element on top of the stack. if the stack is full before Pushing.
- Return the elements at top of stack without removing it. check stacks is Empty before Peeking.
- check if the stack is full by comparing the top Pointer of Equivalent variable to the maximum of stack.

———o———