Assignment-1

Name: K. Nitheesh Kumar
Regno: 19232509.

1. Describe the concept of abstract data type (ADT) & how they differ from concrete data structure. Design an ADT for a stack and implement it using arrays.

Sol. Abstract data type (ADT) :-

An abstract data type (ADT) is a theoritical 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.

Characteristics of ADT's :-

→ operations :- Defines a set of operations that can be performed on the DS.

→ Semantics :- Specifies the behaviour of each operations

→ Encapsulation :- Hides the implementation details, focusing on the interface provide to the user.

ADT for stack :-

A stack is a fundamental data structure that follows the last in, first out (LIFO) principle. It supports the following operations.

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

→ Pop : Removes & returns the element.

→ Peak :- returns the element from 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 structure :-

The implementations using arrays and linked lists are specific ways of implementing the stack ADT in.

Advantages of ADT :-

→ Modularity, encapsulation, flexibility in design.

Implementation in C using arrays :-

```
#include <stdio.h>
# define MAX_SIZE 100
typedef struct {
    int items[MAX_SIZE];
    int top; }
stack array;
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:", stack.items[stack.top]);
} else {
    printf ("stack is empty!\n");
} if (stack.top!=-1) {
printf ("popped element:", stack.top=-1);
} if (stack.top!=-1) {
printf ("popped element", stack.
} else {
    printf("stack underfl
} if (stack.top!=-1) {
printf ("Top element
pops:", stack.items[st
    top=-1);
} else {
print( "stack is em
} 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;
}
  newnode -> data = 10;
  newnode -> next = top;
  top = new node
  newnode = (Node*) malloc (size of (Node));
if (newnode == NULL) {
    printf ("memory allocation failed");
    return 1;
}
new node -> data = 20;
new node -> next = top;
top = new node;
new node = (Node * ) malloc (size of node)
}
if (top != NULL) {
    Node * temp = top;
  printf ("Popped element: ", temp -> data);
```

2) Write pseudocode for stack operations

1) Initialize stack ( ):
Initialize necessary variable or structure to represent the stack.

2. Push element :-
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 error value).
else:
remove & return element from top of stack.
decrement end pointer.

4. PEAK ( ) :-
If stack is empty:
print ("stack is empty");
return null
else:
return element at the top of stack.

5. 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
otherwise, return false.

Explanation of pseudocode :-

→ Initialises the necessary variable or data structure to represent a stack.

→ Adds an element to the top of the stack.

→ removes and returns the elements from the top of the stack. checks if the stack is empty before popping.

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

3. The university announced the selected candites register number for placement training the student xxx, reg no. 20142010 wishes to check wheather his name is listed or not. The list is not sorted in any order. Identify the searching technique that can be applied and explain searching steps.

linear search :-

linear search works by checking each element in the list one by one until the desired element is found at the end of the list is reached.

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.

→ If the target is found, return its position. If the end of the list is reached and the element has not been found.

Procedure :-

Given list :

2014 2016, 20142033, 2014, 2017, 2014 2010, 2014 2056

→ Start at the first element.

→ Compare "20142010" with "20102015", "2014 2033", "20142011", "2014 2019" these are not equal.

→ Compare '20142010' with '2014 2010' They are equal.

→ The element '20142010' is found at the fifth position.

C code for linear search :-

```
#include <stdio.h>
int main() {
    int reg numbers[] = { 20142015, 20142033.....
                                    2014 2003 };
    int target = 2014 2010;
    int n = size of (reg numbers) / sizeof (reg number[0]);
    int found = 0;
    int i;
    for (i=0 ; i<=n ; i++) {
        if (reg numbers[i] = target) {
```

```
        printf("registration number", target);
        found = 1;
        break;
    }
}
if (!found){
    printf("registration number", target);
}
return 0;
}
```