Assignment-1

NAME : Sahithi.K

REGNO : 192311240

SUB.CODE : CSA0389

SUB.NAME : Data structure for stack overflow.

FACULTY.NAME : Dr. Ashok Kumar

NO.OF.PAGES : 12

DATE : 29/07/2024

ASSIGNMENT.NO : 01.

Describe the concept of Abstract data type (ADT) and how they differ from concrete data structures. Design an ADT for a stack and implements it using arrays and linked list in c. Include operations like push, pop, peek, is empty, is full and peek.

Sol: **Abstract Data Type (ADT)**

An abstract Data Type (ADT) is a theoretical model that defines a set of operations and the semantics (behavior) 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 constraint apply to those operations.

- operations : Defines a set of operations that can be performed on the data structure.
- Semantics: specifies the behavior of each operation.
- Encapsulation: Hides the implementation details, focusing on the ~~Pr~~ implemntation details, focusing on the interfance provided to the user.

A Stack is a fundamental data Structure that follow the last in, first out (LIFO) principle. It support the following operations:

- push: Adds an element to the top of the stack.
- pop: Removes and returns the element from the top the Stack.
- peek: Returns the element from the top of the Stock 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 lites are specite ways of implementing the Stack ADT inc.

How ADT differ from concrete Data Structures:

ADT focuses on the operations and their behawior, while, concrete, data structures focus on how those operations are realized with specfic programming constructs (arrays are linked listes).

Implementation in C using Arrays:

```
# Include <stdio.h>
# define MAX-SIZE 100
 type def structs
    int intems{max-size]
    int top;
 } Stack Array;
 Int  main(){
      Stack  Array strack;
      Stack .top=-1;
      Stack .items [++ stacks . top] =10;
      Stack . items [++stacks . top] = 20;
      Stack . items [++ Stack .top]= 30;
 if (Stack . top! = -1) {
   print{"TOP element: %d\n", Stack . items[stack.top];
 } else {
      print["Stack is empty! \n");
 }
  if (Stack . top! = -1) {
     Print("popped element: %d \n", stack item[
                                   stack - top- -]);
 } else{
      printf("stack underflow!\n");
  printf ("stack underflow;\n");
  }
  if (stack . top ! = 1){
     print ("Top element after pops: %d\n",
       stack . item [stack . top);
  } else{
     print ("stack is empty!\n");
  }
  return 0;
  }
```

Implementation in C using linked list:

```
 # include < stdio.h>
 # include < stdio.h>
 typedef struct node{
      int data;
      struct Node * next;
 } Node;
int main(){
      Node * top = Null;
   node* NewNode = (node)*malloc(size of(node));
if (new Node == Null){
        print ("memory allocation failed!\n");
   return1;
}
```

```
top = new node;
new node = (node* ) malloc (size of (node);
if (new node == Null) {
    print("memory allocation failed \n");
  return;
}
new node -> data = 20;
new node -> next = top;
top = new node;
new node = (node* ) malloc (size of (node));
if (new node == Null) {
        printf ("memory allocation failed :\n");
return 1;
}
new node -> data = 30,
new node -> next = top;
top = new node;
if (top != Null) {
    print ("Top element: %d \n", top->data);
} else{
    print ("stack is empty :\n");

    node * temp = top;
    printf ("popped element: %d \n", temp->data);
top = top → next;
    free(temp);
} else{
    print ("Stack underflow!\n");
if (top! = Null){
print("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;
}
```

The university announced. the selected candidates, 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 sor-ted in any order. Identify the searching technique that can be applied and explain the searching steps with the suitable procedure.

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 liner search:

1) Start from the first element.

2) Check if the current element is equal to the target element.

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.

Procedure:

Given the lists

2014 2016, 2014 2033, 2014 2017, 2014 2010, 2014 2050,

2014 200 3.'

1) Start at the first element of the list.

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

3) Compare '20142010' with '20142010' (fifth element. They are equal.

4) The element '2014210' is found at the fifth position (index.

4) in the list.

C code for linear search:

```
#include < stdio.h>
 int main() {
    int regnumbers [] = {20142015, 20142033,
        20142011, 20142017, 20142056, 20142003};
   int target = 20142010;
   int n= size of (reg numbers/ size of(reg(0));
   int faurd =0;
   int i;
 for (i=0;i<n ; i++){
    if (regnumbers[i] == target){.
intf (" Registration number found at index %d.\n",
            target, i);
 found =1;
   break;
 }
 }
 if(!found){
   print (" Registration numbers %d found in list.\n",
     targets);
 }
 return 0;
 }
```

Explanation of the code:

1) The 'reg numbers' array contains the list of registration numbers.

2) 'target' is the registration number we are searching for.

3) 'n' is the total number of elements in array.

4) Interate through each element of the array.

5) Set the 'found' flag to '1'.

6) If the loop completes without finding the target, print that the registration numbers is not found.

Output: Registration number 20142010 found at index 4.

3) write pseudocode for stack operations.

A) 1) Initialize stack():

```
Initialize necessary variable or structures
to represent the stack.
```

2) push (elements):

```
    if stack is full:
        print "stack over flow".
else:
    add element to the top of the stack.
    increment top pointer.
```

3) pop():

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

4) peek():

```
    if stack is empty:
        print "stack is empty".
        return null (or appropriat error value)
else: return element at the top of
the stack (without removing it).
```

```
er wise, return false

, is full:
return true, if top is equal to max size
   otherwise, return false.
```

Explanation of the pseudocode:

- Initializes the necessary variables of data structurs to represent a stack.
- Adds an element to the top of the stack. Checks if the stack is full before pushing.
- Removes and returns the elements from the top of the stack. Checks if the stack is empty before popping.
- Returns the element at the top of the stack without removing it. Checks if the stack is empty before peeking.