Assignment-2

Name:- R. Thrisha

Reg NO:- 192373045

DEPT:- CSE(DS)

CODE:- CSA0389

Course:- Data structures

Faculty Name:- Dr. Ashok kumar

Assignment NO:- 01

Assignment:- pseudo code and their explaination.

Submission Date:- 31/7/24

of Abstract data type (ADT) and
concrete data structures
tack and implement it using
in C. Include operations like
pty is full and peek

pe (ADT)

ype (ADT) is a theoritical model
perations and the semantics
on a data structure without
ata structures should be
vides an high level description
can be performed on the data
apply to those operations

ADTS:-

nes a set of operations that
on the data structure

es the behaviour of each operatio
es the implementation details,
rface provided to the user

mental data structure that
, first out(LIFO) principle. It
ing operations:

ment to the top of the stack
d returns the element from
ck

e element from the top of t
oving it

if the stack is empty

* is full :- checks if the array is full

Concrete data structures :-

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

How ADT differs from concrete data structu

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

Advantages of ADT :-

By separating the ADT from its implementati achieve modularity, encapsulation and flexibili in designing and using data structures in prog This separation allows for easier maintenanc code reuse, and abstraction of the complex ope

Implementation in C using arrays :-

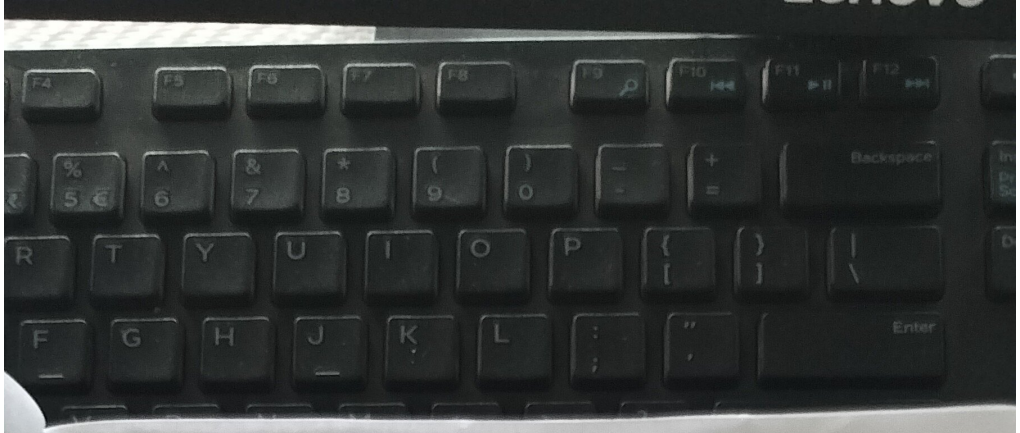
```
#include<stdio.h>
#define MAX_SIZE 100
typedef struct {
    int items[MAX_SIZE];
    int top;
} stack_array;
int main() {
stack_array stack;
```

```
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("poped element: %d\n", stack.items[stack.pop--])
}
else {
  printf("stack underflow : \n");
}
if (stack.top != -1) {
printf("top element after pops : %d\n", stack.item.(sta
    top));
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(sizeof(Node));
    if (newnode == NULL) {
        printf("memory allocation failed\n");
        return 1;
    }
    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;
if (top != NULL) {
printf("Top element: %d\n", top->data);
}
else {
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);
   top = top->next;
   free(temp);
}
else {
printf("stack underflow!\n");
 if (top != NULL){
printf("Top element after pops: %d\n", top->data);
}
else {
printf("stack is empty!\n");
   }
while (top != NULL){
 Node* temp = top;
 top = top->next
  free(temp)
```

3
return 0;
}
3

2) The university announced the selected candidates registers Numbers for placement training. The student xxx reg no - 20/4/2010 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 2014/2015, 20142033, 20142011, 20142017, 20142010, 20142056, 20142003.

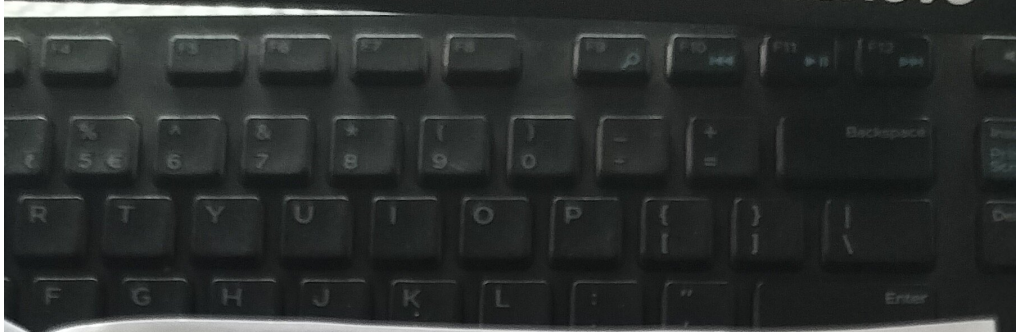
A) linear search:-

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 is a simple searching technique that doesn't require any prior sorting of the data.

steps for linear 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 lis
5) If the target is found, return its positio

Lenovo



not been found, indicate the element is not present

Procedure :-

Given the list :-

2014018, 2014033, 2014011, 2014017, 2014010, 2014056, 2014009

1) Start at the first element of the list

2) Compare '2014010' with '2014015' (first element) '2014033' (second element) '2014011' (third element) '2014017' (fourth element) these are not equal

3) Compare '2014010' with '2014010 (fifth element), they are equal

4) The element '2014010' is found at the fifth position index in the list.

C code for linear search :-

```
#include <stdio.h>

int main() {

int reg Numbers[] = {2014013, 2014033, 2014011, 2014017,
                     2014010, 2014056, 2014003}

int target = 2014010;

int n = size of (reg Numbers)/sizeof(reg Numbers(0));

  int found = 0;

  int i;

for(i=0; i<n; i++) {

  if (reg Numbers[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;
}
```

Explaination of the code:-

1) The 'regNumbers' 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 the 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 is not found.
7) The program will print the index of the found registration Number or indicate that the registration is not present.

Output: Registration Number 2014010 found at index 4

Write pseudocode for stack operations.

1) Initializestack():

Initialize necessary variable or structures 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 appropriate 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 appropriate error value).

else:

return element at the top of the stack without removing it.

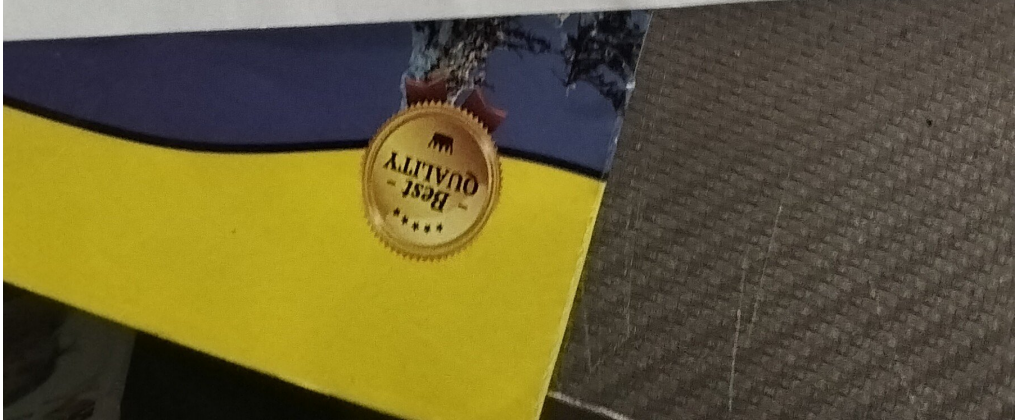
5) is empty():

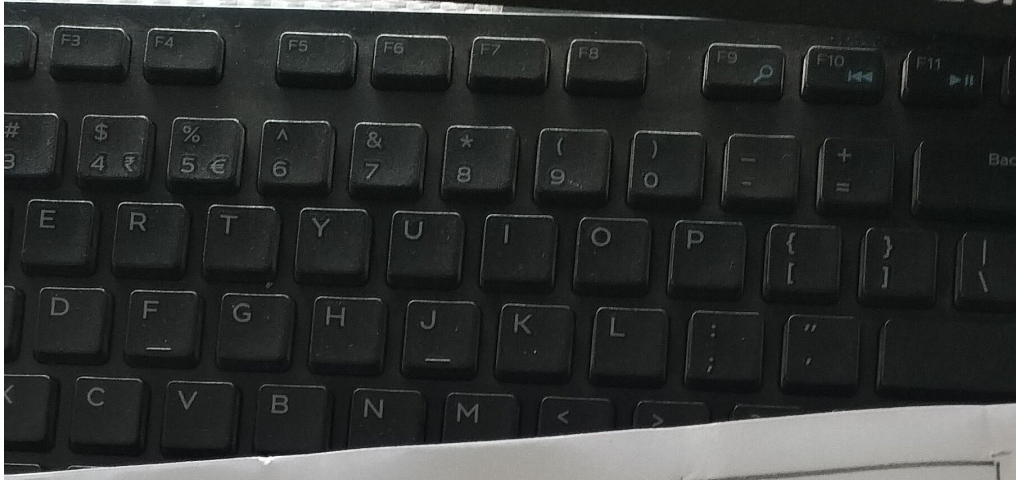
return true: if top is -1 (stack is empty)

otherwise return false

6) is full:

return true, if top is equal to Maxsize-1 (stack is full)





otherwise, return false

Explanation of the pseudocode:-

* Initializes the necessary variables as data structures to represent a stack
* Add an element to the top of the stack. checks if the stack is full before pushing
* Removes and returns the element 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
* checks if the stack is empty by inspecting the top pointer as equivalent variable
* checks if the stack is full by comparing the top pointer as equivalent variable to the maximum size of the stack.