# ASSIGNMENT - 02

## CSA - 0389 -

## DATA STRUCTURE FOR STACK IMPLEMENTATION

Date - 31/7/24

- Aksharaa. B
192311278.

1) Describe the Abstract Data Type (ADT) and how they differ for concrete data structures.

ABSTRACT DATA TYPES:

It is a theoretical model that defines a set of operations and the behaviour of these 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 data.

Characteristic:

* Operations: Defines a set of operations that can be performed on data structure.
* Semantics: Specifies the behaviour of each operation

ADT for stack

A stack is a fundamental data structure that follows the LIFO principle. It supports the following

PUSH: Adds element to the top of stack.

POP: Removes and returns the elements from the top of the stack.

PEEK: Returns the element from the top of the stack, without removing it.

ISEMPTY: Checks if the stack is empty.

ISFULL: Checks if the stack is full.

Concrete data structure

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

How ADT differ from concrete data structure:

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

Advantages:

By seperating the ADT from implementing, you acheive modularity, encapsulation and flexibility in designing and using data structure in program. This seperation allows for easier maintance, code reuse and abstraction of the complex operations.

Implementation in C

```
#include <stdio.h>
int main {
        stack Array stack;
        stack.top == -1;
        stack.items[++stack.top] = 10;
    if (stack.top != -1) {
        printf ("Top Element %d \n", stack.items [stack.top]);}
    else {
        printf ("Stack is empty \n"); }
```

2) The university announced the selected candidates register number for placement baining. The student XXX, reg NO. 20142010 wishes to check is name is listed or not.

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 to the data.

Steps for Linear Search:

1) Start the first element
2) Check if the current element is equal to the target element
3) If the current element not the target, then move to the next element in the list.

4) Continue this process until the target element is found or you reach the end of the list.

5) If the target is found, returns its position. If the end of the list is reached and the element has not been found, indicate that element is not present.

Procedure :

Given the list,

1) Start at the first element of the List.

2) Compare '20142010' with '20142015', 20142033 (second element) these are not equal.

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

4) The element '20142010' is found at the fifth position (index 4) in the list.

→ C code for Linear search

```
#include <stdio.h>
int main() {
    int regno[] = {  };
    int target = 20142010;
    int n = sizeof(regno) / sizeof(regno[0]);
    int found = 0;
    int i;

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

```
if (regno[e] == 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;
}
```

Write Pseudo Code for stack operations.

PUSH ():

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

POP ():

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

PEEK():

if stack is empty ():
print "stack is empty"
return null

else:
return element at the tops of the stack (without removing it)

IS EMPTY:

Return true if top is -1 (stack is empty) otherwise, return false.

IS FULL:

return true, if top is equal to maxsize -1 (stack is full) otherwise, return false

⇒ Explanation

* Add an element to the top of stack, check if the stack is full before pushing.
* Remove and return the element from the top of the stack.
* Return the element at the top of the stack without removing it checks if the stack is empty.