ASSIGNMENT - 1

CH. Venkata Navaneeth.
19931121
CSE

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

Abstract data type (ADT) :-

An abstract data type (ADT) is a theoritical model that defines a set of operations and the semantics (behaviour) of those operations on a data structure, without specifing how the data structure should be implemented. It provides a high level description of what operations can be performed on the data and what constraints apply to those operations.

Characteristics of ADT :-

Operations :- Defines a set of operations that can be performed on the data structure.

Semantics :- Specifies the behaviour of each operation.

Encapsulation :- Hides the implementation details, focussing on the interface provided to 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 the stack

→ pop :- Removes and returns the element from the top of stack.

peek :- Returns the element from the top of the stack without removing it.

is empty :- Checks if the stack is empty.

isfull :- checks if the stack is full.

Concrete data structures :-

The implementations using arrays and linked list are specific ways of implementing the stack ADT in c. ADT focusses on the operations and their behaviour while concrete data structures focus on how those operations are realized using specific programming constructs.

Advantages of ADT :-

By separating the ADT from its implementation, you achieve modularity, encapsulation and flexibility in designing and using data structures in program. This separation allows for easier maintenance, code reuse, and abstraction of the complex operations.

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) {
    printf("memory allocation failed!\n");
    return 1;
}
new node -> data = 2;
new node -> next = top;
top = new node;
new node = (node*) malloc (sizeof (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) {
    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);
```

Implementation in c using array's

```
#include <stdio.h>
#define max_size 100
typedef struct {
    int items[MAX_size];
    int top;
} stackarray;
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: %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("popped element: %d\n", stack.items
                                [stack.top--]);
} else {
```

```
top = top -> next ;
free (temp) ;
} else {
  printf (" stack underflow :\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 sorted 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 decreasing 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 only prior sorting of the data.

steps for Linear Search :-

1. Starts from the first element.
2. Check if the current element is equal to the target element
3. If the current element is not target, move to next.

4. Continue this process until either the target element is found or you reach the end.
5. If the target is found, return its position. If the end of the list is reached.

Procedure :-

Given the list :-

2014 2015, 2014 2033, 2014 2011, 2014 2017, 2014 2010, 2014 2056, 2014 2003

1. Start at the first element of the list.
2. Compare "2014 2010" with "2014 2015"
3. Compare "2014 2010" with "2014 2010" they are equal.
4. The element "2014 2010" is found at the fifth element

C code for linear search :-

```
#include <stdio.h>
int main() {
    int reg numbers [] = {2014 2015, 2014 2033, 2014 2011, 2014 2017
                          2014 2056, 2014 2003}
int target = 2014 2010;
int n = size of (reg numbers) / size of (reg numbers)
 int found = 0;
  int i;
for (i = 0; i < n; i++) {
    if (reg numbers [i] == target) {
    print ("Registration number %d found at index
    found = 1;
      break;
    }}
```

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. Iterate through each element matches the target.
4. If the loop completes with out finding the target.
5. The program will print the index of the found registration number or intialize that the registration is not present.

Write pseudo code for stack operations.

1. Intialize stack operations():
   Intialize necessary variable or structures to respond the stack.

2. push (elements):
   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.

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

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

6) if full:
return true, if top is equal to max size -1
Otherwise return false.

Pseudo code :-

→ Initialize the necessary variables of data structures to represent a stack.

→ Adds an element to the top of the stack. Check if the stack is full.

→ Removes and returns the element from the top of the stack. checks if the stack.

→ Checks if the stack is empty by inspecting the top pointer or equipment variable.

→ Checks if the stack is full by comparing the top pointer or equivalent variable.