NAME : THONDURU SUSHMA

REG NO : 192325135

DEPT : AIML

CODE : CSA0389

COURSE : DATA STRUCTURES

FACULTY NAME : DR. ASHOK KUMAR

ASSIGNMENT NUMBER : 01

ASSIGNMENT : PSEUDO CODE AND THEIR EXPLANATION.

DATE : 31/07/2024

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 lists 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 (behaviour) 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 constraints apply to those operations.

CHARACTERISTICS OF ADT'S:

- 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, focusing on the interface provided 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 and returns the element from the top of the stack.

• peek : Returns the element from the top
out removing it.
• is Empty : checks if the stack is empty.
• is Full : checks if stack is full.

Concrete Data structures:

The implementations using arrays and linked lists are spec
-ific ways of implementing the stack ADT in C.

How ADT differ from concrete data structures:

ADT focuses on the operations and their behaviours, while concr
-ete data structures focus on how those operations are realized
using specific programming constructs (arrays are linked lists).

Advantages of ADT:

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

Implementation in C using arrays:

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

```
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 = new node;
New node = (node* malloc (size of (Node));
if (new node == NULL) {
printf("Memory allocation failed \n");
    return 1;
}
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) {
printf("top element: %d \n", top -> data);
```

```
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{
  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;
```

```
}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 under flow!\n");
if(top!=NULL){
printf("TOP element after pop:%d\n", data);
}
while(top!=NULL){
    top=top->next;
     free(temp);
}
return 0;
}
```

2) The universe 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 techniques that can be applied and explained searching steps with suitable procedure list includes 20142015, 20142033, 20142017, 20142010, 20142056.

Sol: Linear search:-

Linear search works by checking each element in the list one by one until we desired element is found at the end of the list is reached. It's a simple searching technique that doesnot require any pair sorting of the data.

Steps for linear search:-

1. Start from the first element.
2. check if the current element is equal to 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.
5. If the target is found, return its position. If the end of the list is reached and the element has not been found, indicate that element is not present.

pseudo code:-

Given the list:

'20142015, 20142033, 20142017, 20142010, 20142056, 20142003

1) Start at first element of the list.
2) compare the elements of the list.
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.

code for linear search:-

```
#include<stdio.h>
int main() {
int reg numbers[] = {20142015, 20142033, 20142011, 20142017, 20142010,
                     20142056, 20142003};
int target = 20142010;
int n = size of(reg numbers)/size of (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;
}
```

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. Iterate through each element of the array
5. If the current element matches the target, print its and set found.
6. The program will print the index of the found registration number or indicate that it is not found.

output:

Registration number 20142010 found at index.

3. Write Pseudo code for stack operation:-

Sol: 1. Initialize stack():

Initialize memory 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 errorvalue)
else:
    remove and return element from the top of stack
    decrement end pointer
```

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

5. isempty():
return true if top is -1 (stack is empty)
otherwise return false.

6. isfull():
return true, if top is equal to max size - (stack is full)
otherwise, return false.

Explanation of the Pseudo code:

1. Initializes the necessary variables of data structures to represent a stack.
2. Adds an element at the top of the stack. checks if the stack is full before pushing.
3. Removes and returns the element at the top of the stack.
4. Returns the elements at the top of the stack without removing it checks if the stack empty before peeking
5. checks if the stack is empty by inspection the top pointer or equivalent check.
6. checks if the stack is full by comparing the top pointer or equivalent variable to the maximum size of stack