Name: Kvamsi Krishna yadav.

Reg no: 192324165

Course code: CSA0389

Course name: Data structure for Stack Overflow

Assignment = 01

Date of submission: 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 operation like Push, Pop, Peak, is empty, is full and Peek.

Abstract datatypes:

An abstract data types (ADT) is a the original model that defines a set of operations. and the semantics of these operation 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 ADTs:

- operations: Define 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 operation

Push: Adds or pushes elements to top of the stack.

Pop: Remove and returns the elements from top of the stack.

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

is empty: checks if stack is empty.

is full: checks if stack is full.

concrete Data structure:

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

How ADT differs concrete data structure:

ADT focuses on the operations and their behaviour, while concrete data structures focus on how those operations are achieved using specific programming constructs like arrays and linked lists.

Advantages of ADT:

By seperating the ADT from its implementation you achieve modularity and flexibility in data structures in program. This separation allows for easier maintaince, code and optimization of the complex operations.

/* Implementation using Array:

```
#include <stdio.h>
#define Max-size 100
typedef struct {
    int items [Max-size];
    int top;
} stackArray;
int main() {
        stack Array stack;
        stack top = -1;
        stack.item [++ stack.top] = 10;
        stack.item [++ stack.top] = 20;
        stack.item [++ stack.top] = 30;
if (stack.top == -1) {
  Printf ("TOP element: %d/n", stack.item[stack top]);
} else {
  Printf ("stack is empty"); }
if (stack.top != -1) {
  Printf ("Popped element "%d/n", stack.items
                [stack.top-]};
} else {
    Print ("stack underflow! /n") }
  if (stack top! = -1) {
  Printf ("Poped element : %d/n") stack.
          item [stack.top-]);
  } else {
    Printf ("stack underflow: /n");
    }.
```

```
    If (stack.top != -1) {
    Printf ("top element after POP : %d\n", stack
                [stack.top]);
    } else {
        Printf ("stack is empty!\n");
    }
    return 0;
    }.
```

implementation using linked list :

```
#include <stdio.h>
#include <stdlib.h>
    type def struct node {
        int data;
        struct node * next;
    } node;
    int main() {
        node * top = null;
    node * new node = (node)* malloc (sizeof(nod
    If (new node == null) {
        Printf ("memory allocation\n");
        return;
    }
    new node -> data = 10;
    new node -> next = top;
        top == new node;
    new node = (node *) malloc (size of (node));
```

```
if(newnode == null){
Printf("memory allocation failed \n");
    return 1; }
    new node -> data = 10;
    new node -> next = top;
      top = new node;
new node = (node*) malloc (size of (node));
    if (new node == = null){
    Printf("memoryallocation failed \n")
      return 1;
      }
    new node -> data = 30;
    new node -> next = top;
      new == top;
      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);
```

```
top = top -> next;
    free (temp);
  } else {
  Printf (" stack underflow /n");
  if (top != null) {
Printf (" TOP element after POP= %d/n" ,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 cardiate 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 suitable. Produuse. list includes. 2014, 2015, 2014, 2023, 2014 2011, 2014 2017, 20142010, 2014 2056, 20142003.

Sol 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 of seached. It'a simple searching technique that doesn't require any pair sorting of the data.

Steps for linear search:

1) start from the first element.
2) checks if the cosrent element is equal to the target element.
3) if the cosrent element is not the target, move to next element in list.
4.) continue this process until either the target element is found or you reach the end of 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 the element is not present.

Procedure:

given the list

20142018, 20142033, 20142011, 20142010,

20142056, 20142002.

i) start at the first element of list

ii) compare '20142010' with ('20142018' '20142033'

'20142011' '20142017', these not equal.

iii) compare '20142010' with '20142010' they are equal.

iv) the element '20142010' is found at fifth position in list.

code for linear search:

```
# include <stdio.h>
    int main (){
    int reg no[]= {20142010 20142033,
      20142011, 20142017, 20142010,
       20142056 , 20142003};
    int target = 20142010;
    int n=size of (reg no)/ size of (reg no[0])
     int found = 0;
        int i;
    for (i=0; i<n; i++){
    if ( reg no [i] = target ){
Print f ( "registration no.d found at index,
                target ,i);
     found = 1;
       break;
    }
}.
```

```
if (!found) {
    printf("registration no %d not found
                in list in", target);
}
return 0;
}
```

Explanation of code:

i) The 'reg no' array contains the list of registration no.

ii) target is the registration no we are searching for

iii) n is the total no. of element in array.

iv) Iterate through each element in array.

v) If the current element matches the target, print that the registration number is not found.

Output:

Registration number 20142010 found at index 4.

③ write pseudocode for stack operation.

1) Initialize stack():

initializes 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 stack
    increment top pointer.
```

PopC):

If stack is empty
print ("stack underflow")
return null (or appropriate error value)
else:
remove and return elements from the top of stack decrement end pointer

④ Peek C):

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

⑤ is empty C):

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

⑥ is full C):

return true if top is equal to maxsize -1 [stack is full] otherwise, return false.