Name: O. Jagan mohan reddy

Reg No : 192324193

Course code : CSA 0389

Course name : Data Structure for Stack Over flow

Assignment : 01

Submission : 29/07/24

O. Jagan mohan reddy
192324193

Assignment

1. Describe of The concept of Abstract data Type(ABT) and how They differ from data Structures and its characteristics.

Sol:

Abstract data Type(ADT):

An abstract data type (ADT) is a Theatrical model That set of operations and Semantics behaviour of Those operations on a data structure without specifing how to The data structure should be implemented. It provides a high level description of what operations.

characterstics 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). It suppourts The following operations.

②

Implementation in c using arrays:

```
# include < stdio.h>
 # define  MAX_ SIZE  100

Type def  Struct {
      int items [max_size];
       int Top;
      }
      stack array;
int main(){
   Stack-Array   stack;
    stack. top= -1;
    stack. items [++stack. top]= 20;
    if ( stack. top = -1){
       printf (" Top element : %d\n", stack.items") }
  else :
   {
      printf (" stack is empty ! \n");
   }
     if ( stack. top: =-1) {
        print (" popped element : %. d \n");

     }
     if (stack. top! =-1) {
       printf (" popped element : %. d\n",);
       } else {
```

(3)

```
printf("Stack underflow;\n");
}
if (Stack.Top1 = -1) {
    printf(" Stack is empty \n");
}
    return 0;
}
```

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 (newnode = Null) {
   printf(" memory allocation failed: \n");
     return 1;
  }
 newnode -> data = 10;
 newnode -> next = top;
```

④

```
Top = newnode ;
if (newnode = Null) {
    printf (" memory allocation failed;\n");
    return 1;
}
newnode -> data = 20;
newnode -> next = top;
Top = newnode;
newnode = (node*) malloc( sizeof (Node));
if (newnode = Null) {
    printf (" memory allocation failed:\n")
if (newnode = = Null) {
  printf (" memory) {
  return 1;
 if ( top ! = Null) {
    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 is empty :\n");
```

```
int Target = 20142010;
int n = size of (reg numbers / size of (reg numbers (0) ;
int found =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, :) ;
found = 1 ;
break ;
}
}
if ( found ) {
printf (' Registration number not found) ;
}
return 0;
}
```

Explanation of the code:

1) The 'regnumbers' array contains the list of regestration numbers

2) " Target " is the regestration number we are cusing for.

4) Iterate Through each element of The array

5) if The current element matches The "Target" print its index and sets The flag To "1".

6) if The loop completes without find The Target That The regestration number is not found.

7) The program will print The index of The found regestration.

<u>Output:</u> " Regestration numbers 20142010 found at index"

③ <u>Write pseudocode for stack operations</u>

1. <u>Instalise Stack ();</u>

Instalise neccessary Variable or Structure to represent The stack

2. <u>push</u>

if stack is full;

print " stack over flow";

else:

add element to The Top Of The stack

increment Top pointer

3. pop ( ):

if stack is empty :

print (" stack underflow")

return null (or appriciate 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 appriciate error value

else:

return element at The Top of The stack

5. is empty ( ) :

return True : if Top is -1 (stack is empty)

Otherwise, return false

6. is full :

return True, if Top is equal Top is equal To max. size (Stack is full)

Otherwise, return false.

Explanation of pseudocode:

=) Instalises The nessaccary Variables of data structures

=) Adds an element to The Top of The stack.

=) Removes and returns The element from The Top of The Stack.

=) if The Stack is empty before popping

=) Returns The element at The top of The Stack without meaningit.

=) Checks if The Stack is full by Comparing The Top pointer or equivalent Variable The maximum size of The Stack.

linear Search:

linear Search wants by checking each element in The list one by one until The desired element is found in The doesn't require any prior sorting of data

Steps for linear search:

1) Start from The first element
2) Check if The current is equal to The Target element
3) if The Current element is not The Target element is found you reach The end of The list
5) if The Target 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.

procedure:

Given The list:

"20142016, 20142033, 20142011, 20142017, 20142010, 20142056, 20142002"

=) Stack at The first and fifth position index

=) in The list!

C Code for linear Search:

```
# include <stdio.h>
 int main ( ) {
     int regnumbers { } = {given numbers in Q')
```

```
top = top -> next;
free(Temp);
} else {
  printf(" Top element after pops : %d\n", Top->data);

} else {
while (top != Null) {
  Node * Temp = top;
  Top = Top -> next;
  free(temp);
}
  return 0;
}
```