자료구조론 CC343_2207

Reading assignment 7

경기대학교 컴퓨터공학부

201511837 이상민

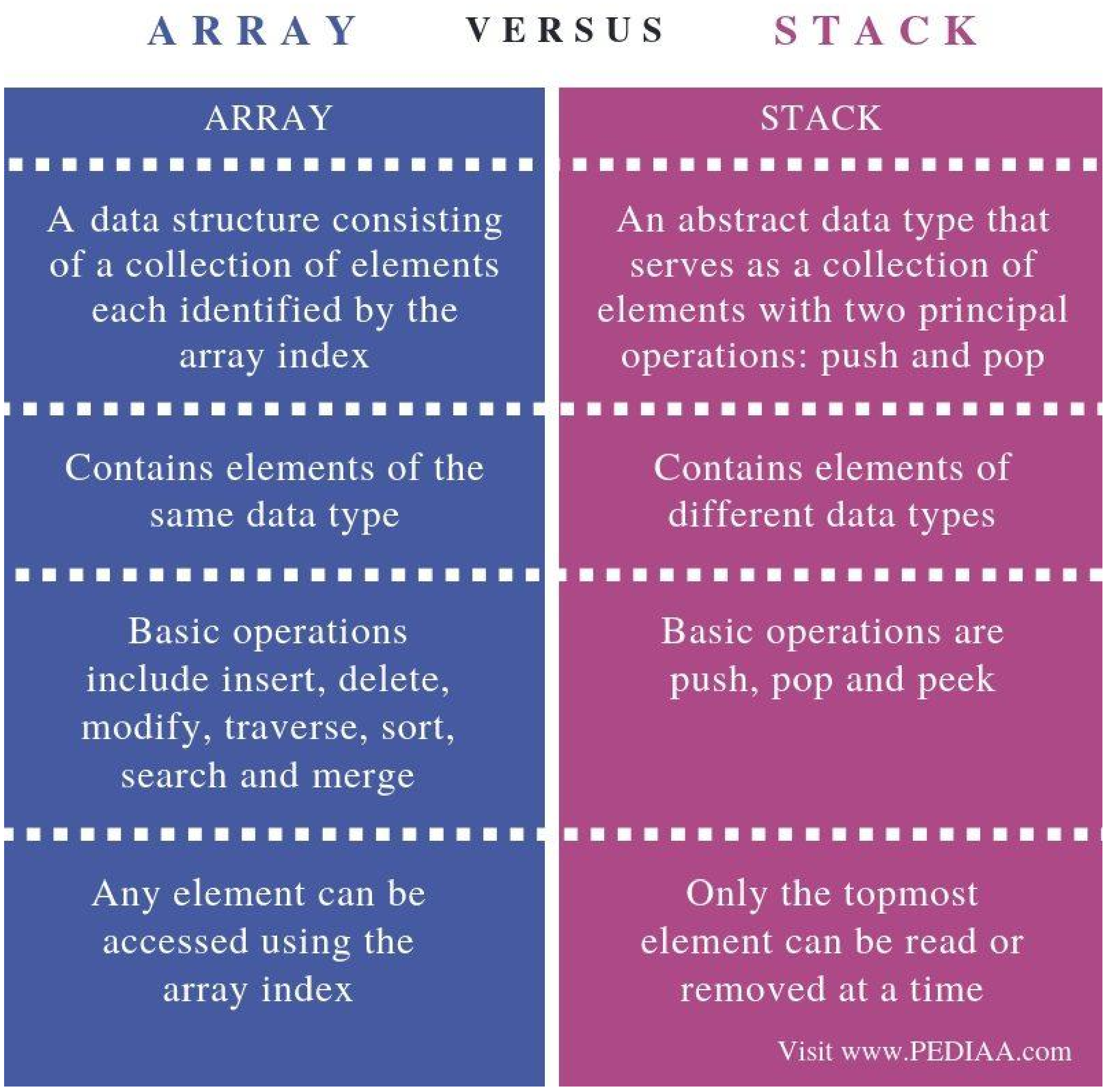
Review Questions

1. What do you understand by stack overflow and underflow?

Stack is a collection of elements in which elements follows Last In First Out(LIFO) mechanism. In stack, to perform push(insert) or pop(delete) operation on elements we need the special variable called tos(top of stack). Traditionally, top of stack is named as tos note, variable name could be anything.

Initially, tos should point at -1(i.e,tos=-1;).By incrementing the pointer we can push elements into the stack and by decrementing the pointer, elements would be poped. Suppose i want to create an stack of n elements. Now if by pushing n elements into the stack, that is my tos is pointing at (n-1)th index, and now again if i try to push any element at nth index, i cannot becoz the stack is now full,i.e stack has been overflow.Also, when my tos = -1,i.e all the elements have been poped from the stack, means the stack is now underflow.

2. Differentiate between an array and a stack.



ARRAY VERSUS STACK

ARRAY	STACK
A data structure consisting of a collection of elements each identified by the array index	An abstract data type that serves as a collection of elements with two principal operations: push and pop
Contains elements of the same data type	Contains elements of different data types
Basic operations include insert, delete, modify, traverse, sort, search and merge	Basic operations are push, pop and peek
Any element can be accessed using the array index	Only the topmost element can be read or removed at a time

Visit www.PEDIAA.com

3. How does a stack implemented using a linked list differ from a stack implemented using an array?

Stack: Linked list: As a singly-linked list with a head pointer. Array: As a dynamic array.

Queue: Linked list: As a singly-linked list with a head and tail pointer. Array: As a circular buffer backed by an array.

4. Differentiate between peek() and pop() functions.

top is a state variable for your stack, which is stored in a regular array. The variable top references the top of the stack by storing an array index.

The first operation, pop, uses the decrement operator to change the variable top, and hence the state of the stack: --top is equivalent to top = top - 1. The value is still in the array, but since the variable top now references a different index, this value is effectively removed: the top of the stack is now a different element. Now if you call a push, this popped value will be overwritten.

The second operation doesn't change the value of the variable top, it only uses it to return the value at the top of the stack. The variable top still references the same value as the stack's top, and so the stack is unchanged.

5. Why are parentheses not required in postfix/prefix expressions?

Parenthesis is not required because the order of the operators in the postfix /prefix expressions determines the actual order of operations in evaluating the expression

6. Explain how stacks are used in a non-recursive program?

The same way they are used in a program with recursive functions.

The stack is used to keep track of what function called which other function.

```
int square(int x) {
  return x * x;
}

int main() {
  int a = square(64);
  int b = square(2);
  printf("a = %d, b = %d₩n", a, b);
  return 0;
}
```

In the example, without a stack, when line 2 is executed, how does it know whether to return to line 6 or line 7? The stack is used to store where to return to.

7. What do you understand by a multiple stack? How is it useful?

When a stack is created using single array, we can not able to store large amount of data, thus this problem is rectified using more than one stack in the same array of sufficient array. This technique is called as Multiple Stack.

8. Explain the terms infix expression, prefix expression, and postfix expression. Convert the following infix expressions to their postfix equivalents:

(a) A-B+C
answer : AB-C+

(b) A*B+C/D

answer : AB*CD/+

(c) (A – B) + C * D / E – C
answer : AB-CDEC-/*+

(d) (A * B) + (C / D) – (D + E)

answer : AB*CD/DE+-+

(e) ((A – B) + D / ((E + F) * G))

Answer : AB-DEF+G*/+

(f) (A – 2 * (B + C) / D * E) + F

Answer : ABC+DE*/*2–F+

(g) 14 / 7 * 3 – 4 + 9 / 2

Answer : 2 / 9 + 4 – 3 * 7 / 41

9. Convert the following infix expressions to their postfix equivalents:

(a) A-B+C
answer : AB-C+

(b) A*B+C/D
answer : AB*CD/+

(c) (A – B) + C * D / E – C
answer : AB-CDEC-/*+

(d) (A * B) + (C / D) – (D + E)

answer : AB*CD/DE+-+

(e) ((A – B) + D / ((E + F) * G))

Answer : AB-DEF+G*/+

(f) (A – 2 * (B + C) / D * E) + F

Answer : ABC+DE*/*2–F+

(g) 14 / 7 * 3 – 4 + 9 / 2

Answer : 2 / 9 + 4 – 3 * 7 / 14

10. Find the infix equivalents of the following postfix equivalents:

(a) AB+C*D-

Answer : (A+B)*C-D

(b) ABC*+D-

Answer : A+B*C-D

11. Give the infix expression of the following prefix expressions.

(a) *-+ABCD
answer : (((A*B)-C)+D)

(b) +-A*BCD

Answer : A+B*C-D

12. Convert the expression given below into its corresponding postfix expression and then evaluate it. Also write a program to evaluate a postfix expression.

10 + ((7 – 5) + 10)/2

Postfix : 10 7 5 10 + 2 / + = 16

13. Write a function that accepts two stacks. Copy the contents of first stack in the second stack. Note that the order of elements must be preserved. (Hint: use a temporary stack)

```
struct Stack {

        int data;

        struct Stack *next;

};

struct Stack *CreateStack () {

        return NULL;

}

int isEmptyStack(struct Stack *top) {

        return (top == NULL);

}
```

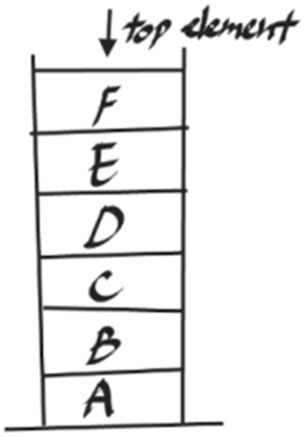
```
};

void Push(struct Stack **top, int data) {
            struct Stack *newNode = (struct Stack*) malloc(sizeof(struct Stack));
            if(!newNode)
                  return;
            newNode->data = data;
            newNode->next = *top;
            *top = newNode;
}
int Pop(struct Stack **top) {
            struct Stack *temp;
            int data;
            if(isEmptyStack(*top)) {
                  printf("Empty Stack.₩n");
                  return INT_MIN;
            }
            temp = *top;
            data = (*top)->data;
            *top = (*top)->next;
            free(temp);
            return data;
}
struct Queue {
            struct Stack *S1;
            struct Stack *S2;
};
```

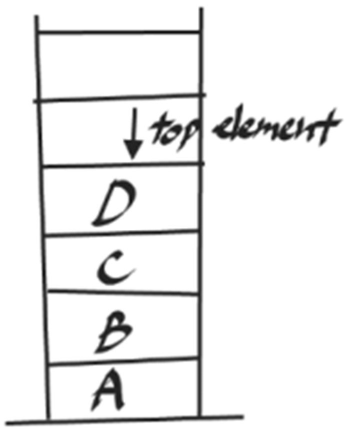
```
struct Queue *CreateQueue() {
        struct   Queue   *newNode = (struct Queue *) malloc(sizeof(struct   Queue ));
        return newNode;
}
void EnQueue(struct Queue *Q, int data) {
        Push(&Q->S1, data);
}
int DeQueue(struct Queue *Q) {
        if(!isEmptyStack(Q->S2)) {
            return Pop(&Q->S2);
        }
        else {
            while(!isEmptyStack(Q->S1)) {
                Push(&Q->S2, Pop(&Q->S1));
            }
            return Pop(&Q->S2);
        }
}
```

14. Draw the stack structure in each case when the following operations are performed on an empty stack.

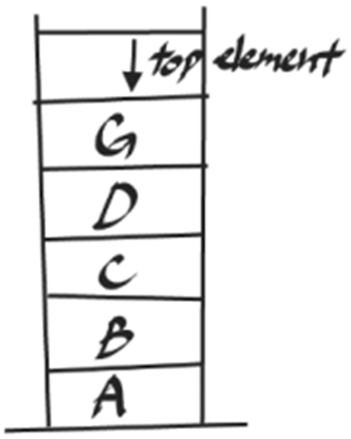
(a) Add A, B, C, D, E, F



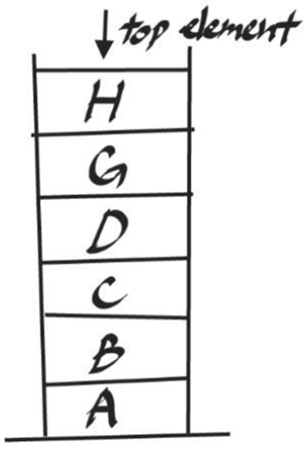
(b) Delete two letters



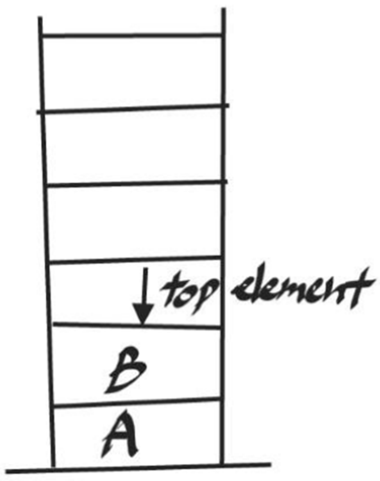
(c) Add G



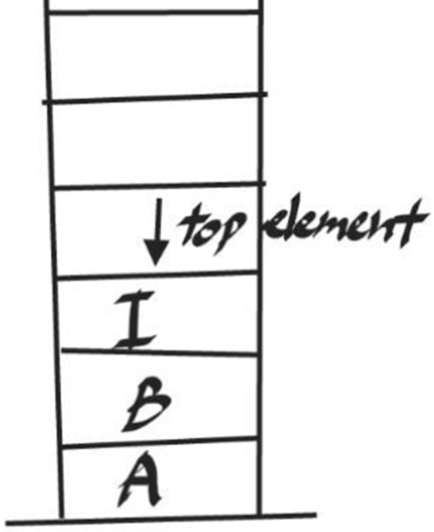
(d) Add H



(e) Delete four letters



(f) Add I



15. Differentiate between an iterative function and a recursive function. Which one will you prefer to use and in what circumstances?

BASIS FOR COMPARISON	RECURSION	ITERATION
Basic	The statement in a body of function calls the function itself.	Allows the set of instructions to be repeatedly executed.
Format	In recursive function, only termination condition (base case) is specified.	Iteration includes initialization, condition, execution of statement within loop and update (increments and decrements) the control variable.
Termination	A conditional statement is included in the body of the function to force the function to return without recursion call being executed.	The iteration statement is repeatedly executed until a certain condition is reached.
Condition	If the function does not converge to some condition called (base case), it leads to infinite recursion.	If the control condition in the iteration statement never become false, it leads to infinite iteration.
Infinite Repetition	Infinite recursion can crash the system.	Infinite loop uses CPU cycles repeatedly.
Applied	Recursion is always applied to functions.	Iteration is applied to iteration statements or "loops".

BASIS FOR COMPARISON	RECURSION	ITERATION
Stack	The stack is used to store the set of new local variables and parameters each time the function is called.	Does not uses stack.
Overhead	Recursion possesses the overhead of repeated function calls.	No overhead of repeated function call.
Speed	Slow in execution.	Fast in execution.
Size of Code	Recursion reduces the size of the code.	Iteration makes the code longer.

16. Explain the Tower of Hanoi problem.
The Tower of Hanoi, is a mathematical problem which consists of three rods and multiple disks. Initially, all the disks are placed on one rod, one over the other in ascending order of size similar to a cone-shaped tower.

The objective of this problem is to move the stack of disks from the initial rod to another rod, following these rules:

A disk cannot be placed on top of a smaller disk

No disk can be placed on top of the smaller disk.

The goal is to move all the disks from the leftmost rod to the rightmost rod. To move N disks from one rod to another, 2^N-1 steps are required. So, to move 3 disks from starting the rod to the ending rod, a total of 7 steps are required.

Multiple-choice Questions

1. Stack is a

(a) LIFO (b) FIFO (c) FILO (d) LILO

2. Which function places an element on the stack?

(a) Pop() **(b) Push()** (c) Peek() (d) isEmpty()

3. Disks piled up one above the other represent a

(a) Stack (b) Queue (c) Linked List (d) Array

4. Reverse Polish notation is the other name of

(a) Infix expression (b) Prefix expression **(c) Postfix expression** (d) Algebraic expression

True or False

1. Pop() is used to add an element on the top of the stack. **: False**

2. Postfix operation does not follow the rules of operator precedence. **: True**

3. Recursion follows a divide-and-conquer technique to solve problems. **: True**

4. Using a recursive function takes more memory and time to execute. **: True**

5. Recursion is more of a bottom-up approach to problem solving. **: False**

6. An indirect recursive function if it contains a call to another function which ultimately calls it. **: True**

7. The peek operation displays the topmost value and deletes it from the stack. **: False**

8. In a stack, the element that was inserted last is the first one to be taken out. **: True**

9. Underflow occurs when TOP = MAX-1. **: False**

10. The storage requirement of linked representation of the stack with n elements is O(n). **: True**

11. A push operation on linked stack can be performed in O(n) time. **: False**

12. Overflow can never occur in case of multiple stacks. **: False**

Fill in the blanks

1. ______ is used to convert an infix expression into a postfix expression.

answer : stack

2. ______ is used in a non-recursive implementation of a recursive algorithm.

answer : stack

3. The storage requirement of a linked stack with n elements is ______.

answer : O(n)

4. Underflow takes when ______.

answer : We try no delete a node from a stack that is empty

5. The order of evaluation of a postfix expression is from ______.

answer : Left to right

6. Whenever there is a pending operation to be performed, the function becomes ________ recursive.

answer : Non-tail

7. A function is said to be _______ recursive if it explicitly calls itself.

answer : Directly