KESHAV KRISHNA
2401331530115

Page :
Date :

Assignment - 03

1) Recursion is a better technique for solving the Tower Hanoi problem because the problem itself is defined recursively. Each step involves solving a smaller instance of the same problem, which makes the recursive approach simple, natural and easy to understand. This, iterative method is more complex and harder to implement, while recursion directly represents the logical structure of the solution.
Hence, recursion is preferred over iteration for the Tower of Hanoi problem.

2) Datatype using Stack

Code -

```
# include <iostream>
# include <stack>
using namespace std;
int main () {
    stack <char> S;
    String input = "ABCDEF";
    for (char ch : Input)
        S.Push (ch);
    count << "Reversed Order:";
    while (!S.empty ());
        S.pop ();
    }
    return 0;
}
```

Page :
Date :

Output → Reversed order = FEDCBA

Que-3 Application of Stack

- Expression evaluation and conversion (eg. infix to postfix).
- Function call management (using call stack in recursion).

Application of Queue

- Job scheduling or Process management in operation system
- Data transfer in networking (eg. buffering message queues)

Representation of Stack using Linked List

A Stack can be represented using a singly linked list where -

- Each node contains data to the head of the next node.
- The top of the stack points to the head of the linked list.

– Push operation - Insert a new node at the begining (head)

– Pop operation - Delete the node from the begining (head)

Que-4 Given expression → =+11421*3+1567

1142 = 7

+15 = 6

*36 = 18

1186 = 3

+73 = 10

-107 = 3

Ans = 3

Page :
Date :

5) To delete an element from the middle of a stack when only push and pop operations are allowed we use an auxiliary stack first, we pop elements from the original stack and push them into the auxiliary stack until we reach the middle element. Then we pop all elements from the auxiliary stack and push them back into the original stack to restore the order.

Algorithm -
Find middle position = size/2
Pop & push first half elements into auxiliary stack
Pop the middle element
Push back all elements from auxiliary stack to original stack.
Time complexity = O(n)
Space complexity = O(n)

6)

Given Infix expression -
A ^ B + C (D + E) / F - G

Step ① Insert the missing operator for clarity -
A ^ B * C * (D + E) / F - G

Step 2 Apply stack method (operator precedence : ^ > *, / > +, -)

Page :
Date :

Symbol	Stack	Output
A		A
^	^	A
B	^	AB
+	+	AB^
C	+	AB^C
*	+*	AB^C
C	+*C	AB^C
D	+* C	AB^CD
+	+* C	AB^CD
E	+*C	AB^CDE
)	+*	AB^CDE+
/	+/	AB^CDE+*
F	+/	AB^CDE++F
-	+/	AB^CDE+*F/+
G		AB^CDE+*F/+G
		AB^CDE+*F/+G-

7 Head Recursion

In head recursion the function calls itself before performing any operation. All statement are executed after the recursive call returns

Tail Recursion

In tail recursion the function calls itself after performing all operations. No work is left after the recursive call

Page :
Date :

Program for fibonacci series :

```
#include <iostream>
using namespace std;
int fibonacci (int n) {
    if (n<=1) {
        return n;
    return fibonacci (n-1) + fibonacci (n-2);
}
int main () {
    int n;
    cout << "Enter number of terms")
    cin >> n;
    Count << "Fibonacci series!"
        for (int i=0; i<n; i++)
            cout << fibonacci <<" ");
    return 0;
}
```

Output - 0 1 1 2 3 5 8

8) Circular Queue

A circular queue is a type of queue in which the last position is connected back to the first position to form a circle. It helps in efficient use of memory because unlike normal queue, the empty spaces left after deletions can be reused. In a normal queue, once the rear reaches the end, no new elements can be inserted even if there is space at the beginning.

But in a circular queue the rear can move back to the front and use the

Page :
Date :

free space again. Hence, it utilizes memory better a normal queue.

Algorithm for Circular queue

Insertion -

① If (front == 0 && rear == size-1) or (front = rear+1)
Queue is full

② Else if (front == -1)
front = 0
rear = 0

③ Else if (rear == size-1)
rear = 0

④ Else
rear = rear + 1

⑤ Insert the element at queue(rear)

Deletion

① If (front == -1)
Queue is empty

② Store queue[front] in item

③ If (front == rear)
front = -1
rear = -1

④ Else if (front == size-1)
front = 0

⑤ Else
front = front + 1

⑥ Display delete item

Page :
Date :

9) Rewrite : A+B*(D+E)+F-G

Parenthesize to show order

(((A+B*(D+E))+F-G)

Convert sub-expressions to prefix:

- D+E → +DE
- B*(D+E) → *B+DE
- A+(B*(D+E)) → +A*B+DE
- (A+B*(D+E)+F → ++A*B+DEF

Final - (++A*B+DEF)G → ++A*B+DEFG

4)

Given expression : /-*XY*WZ-MN

-MN → (M-N)

*WZ → (W*Z)

*XY → (X*Y)

-*XY*WZ → (X*Y)-(W*Z)

/-+XY*WZ-MN → ((X*Y)-(W*Z))/(M-N)

12)

A number is said to be palindrome if it remains the same when its digits are reversed for eg. 121, 1331, 4554 are palindromes

Using a stack we can check this by pushing digits and then popping them in reverse order

Algorithm (using Stack)

Start

Read the number n

Page :
Date :

Copy n into a temporary variable num
Initialize an empty stack
While num > 0
Push num % 10 onto the stack
num = num / 10
Set rev = 0 and pos = 1
while stack is not empty
Pop top element and add it to rev = rev * 10 + popped element
If rev == n
Print "Number is Palindrome"
else
Print "Number is not Palindrome".
Stop

Que-13

Merge Sort is based on the divide and conquer technique. In this technique, a problem is divided into smaller sub-problems each sub-problem is solved individually and then result are combined to get the final solution.

Algorithm:
If low < high then,
mid = (low + high) / 2
Merge Sort (arr, low, mid).
Merge sort (arr, mid, high)
Merge sort (arr, low, mid, high)
end if

Time complexity → O(nlogn)
Space complexity → O(n)

Page :
Date :

14)

Array → 11, 2, 45, 33, 15, 21, 10, 5

Algorithm

Merge Sort (arr, low, high)
If low < (high then
mid = (low + high) / 2
Merge . Sort (arr, low, mid)
Merge . Sort (arr, mid + 1, high)
Merge (arr, low, mid, high)
end if
Merge (arr, low, mid, high)
Copy left half and right half into two temporary array
Compare elements of both halves
Copy smaller element into main array
Repeat until you get sorted array

Step by Step sorting

Divide : [11, 2, 45, 33, 15, 21, 10, 5]
→ [11, 2, 45, 33] & [15, 21, 10, 5]

further divide

[11, 2], [45, 33], [15, 21], [10, 5]

Sort each →

[11, 2] → [2, 11]
[45, 33] → [33, 45]
[15, 21] → [15, 21]
[10, 5] → [5, 10]

Merge

[2, 11, 33, 45] & [5, 10, 15, 21]
[2, 5, 10, 11, 15, 21, 33, 45]