24/10/23

P. Keshava Chand
VU22CSEN0100987

DS-ASSIGNMENT-2

Q1. Implement queue using 2 stacks

Ans:-

Algorithm

① Create 2 stacks : 'stack1' and 'stack2'.

② For enqueue operation, push an element into 'stack1'

③ For dequeue operation, first we check if 'stack2' is empty or not. If 'stack2' is empty, pop an element from stack1 and push it into 'stack2'. Now pop the element from 'stack2'. If 'stack2' is not empty, then directly pop an element from 'stack2'

④ The enqueue and dequeue operations make both stacks 'stack1' and 'stack2' together behave as a queue

⑤ Similar to queue, the collection of stacks also behaves in First In, First Out fashion. (FIFO).

Code

```
#include <stdio.h>
#include <stdlib.h>
#DEFINE  MAX 20
int stack1[MAX], stack2[MAX];
int top1 = -1, top2 = -1;
int  n=0;
void push (int data) {
    if (top1 == MAX) {
        printf("Stack Overflow");
    }
    else {
        top1++;
        stack1[top1] = data;
}
}
```

```
void push2(int data) {
    if(top2==MAX-1) {
        printf("Stack Overflow");
    }
    else {
        top2++;
        stack2[top2]=data;
int pop1() {
    if(top1==-1) {
        printf("Stack Underflow");
    else {
        return stack1[top1];
        top1--
    }
}
int pop2() {
    if(top2==-1) {
        printf("Stack Underflow");
    }
    else {
        return stack2[top2];
        top2--;
void enqueue(int x) {
    push1(x)
    n++;
```

```
Void dequeue() {
    int a,b,i;
    if (top1==-1 && top2==-1) {
        printf("Queue is Empty");
    }
    else {
      for(i=0;i<n;i++) {
        a=pop1();
        push2(a);
        b=pop2();
      }
      printf("Element in queue is %d",b);
      n--;
      for (i=0;i<n;i++) {
        a=pop2();
        push1(a);
```

```
void display() {
    int i;
    for(i=0;i<n;i++) {
        printf("%d", stack1[i]);
    }
```

```
int main() {
    enqueue(3);
    enqueue(2);
    enqueue(7);
    dequeue();
    dequeue();
    display();
}
```

Q2. Implement stacks using 2 queues.

Algorithm

Push Operation:-

1. Consider 2 queues : 'Queue1' and 'Queue2' and let the element to be inserted be n.
2. Directly enqueue the element n into queue1

Pop Operation:-

1. Consider 2 queues : 'Queue1' and 'Queue2' and we want to remove an element.
2. If queue1 is not empty, then we first find out the size of queue1 and for every element in queue1, we dequeue it and enqueue them into queue2. (except the topmost one which has to be popped).
3. Now we extract the element n (info/data/value) and dequeue it
4. Finally, dequeue all elements from queue2 and enqueue them into queue1.

CODE

```
#include <stdio.h>
#include <stdlib.h>
# DEFINE MAX 20
int queue1[MAX], queue2[MAX]
int front1=-1; front2=-1, rear1=-1, rear2=-1;
```

```
int isEmpty() {
    if( front==-1 || front>rear) {
        printf("Queue Underflow");
        exit;
    }
}
Void enqueue1(int data) {
    if(rear1==MAX-1) {
        printf("Queue Overflow");
        return;
    }
    else {
        if (front1==-1) {
            front1++;
        }
        rear1++;
        queue1[rear1]=data;
void enqueue2(int data) {
    if(rear2==MAX-1) {
        printf("Queue Overflow");
        return;
    }else {
        if(front2==-1) {
            front2++;
        rear2++;
        queue2[rear2]=data;
int dequeue1() {
    if(isEmpty()) {
        printf("Queue Underflow");
        return;
```

```
    int data;
    data = queue1[front];
    if (front1 == rear1 == -1) {
        front1 = rear1 = -1;
    else {
        front1++;
        return data;

int main() {
    push(1);
    push(5);
    push(15);
    printf("element in queue is %d", pop());

return 0;
```

Q3. Reverse stack using queue.

Algorithm

~~(1) Create 2 empty queues queue1 and queue2~~

~~(2) Push all elements~~

(1) Create an empty queue : queue

(2) ~~Push~~ Pop all elements of stack into the queue. (by poping)

(3) Now dequeue all elements one by one and push into stack.

CODE

```
#include <stdio.h>
#include <stdlib.h>

int stack[10];
int top = -1;
void push(int item) {
    if (top == 9) {
        printf("Stack Overflow");
        return;
    stack[++top] = item;}
int pop() {
    if (top == -1) {
        printf("Empty Stack");
        return -1;
    }
    return stack[top--];

}
void ~~reverse~~ reverse() {
    if (top == -1) {
        printf("Stack Empty");
        return;
    }
    int queue[10];
    int front = -1;
    int rear = -1;
```

```
while (top != -1) {
        int item = pop();
        if (rear == 9) {
            printf("Queue full");
            return;
        }
        queue[++rear] = item;
    }
    while (front != rear) {
        int item = queue[++front];
        push (item);
    }
}
int main() {
    push (1);
    push (2);
    push (4);
    push (8);
    push (16);
    printf("Original : 16 8 4 2 1")
    reverse ();
    printf("Reverse : %d, %d, %d, %d, %d", pop(),
            pop(), pop(), pop(), pop());
    return 0;
}
```

1 2 4 8 16

O/P

~~5, 4, 3, 2, 1~~

16 8 4 2 1

Q4. Reverse queue using stack

Ans:-

Algorithm

(1) Consider a queue having elements
(2) Create a stack
(3) ~~Start popping all elements one by one and~~
e
(3) Start dequeue-ing elements and push them one-by-one to the stack.
(4) Now once all elements have been removed from queue & pushed onto stack, start to ~~deque~~ pop all elements one by one and enqueue them.
(5) The elements will now be in reverse order.

CODE

```
#include<stdio.h>
#include<stdlib.h>
#DEFINE MAX 10

int queue[MAX];
int stack[MAX];
int front=-1, rear=-1, top=-1;
int isQueueEmpty() {
    return (front==-1 && rear==-1);
}
```

```
int isStackEmpty() {
    return (top==-1);
}
int isQueueFull() {
    return (rear==MAX-1);
}
~~void dequel~~
void enqueue(int item) {
    if (isQueueFull()) {
        printf("Queue Full");
        return;
    }if (front==-1) {
        front=0;
    queue[++rear]=item;
}
int dequeue() {
    if (isQueueEmpty()) {
        printf("Empty Queue");
    return;
    int item=queue[front];
    if (front==rear) {
        front=rear=-1;
    }else {
        front++;
    }return item;
}
```

Q5. Construct Binary Search tree for the following nodes:
71, 32, 12, 82, 45, 91, 38, 70, 40, 61

Ans:-

① Node: 71
Take 71 as the ~~current~~ ~~not~~ root node.

② Node: 32
32 is less than 71, so 32 can be taken as leftchild of 71.

③ Node: 12
12 is less than both 32 and 71. So it can be considered as the leftmost node, i.e. left -child of 32.

④ Node: 82
82 is larger than 71, so 82 is rightchild of 71.

⑤ Node: 45
45 is less than root node 71 So it will be put in left subtree of 71. Now the root of left subtree is 32 and 45 is greater than 32. So 45 is rightchild of 32.

⑥ Node: 91
91 is the highest number in all the nodes presend till now, so it is the rightmost node, Thus 91 is rightchild of 82.

⑦ Node: 38
38 is less than 71, more than 32, less than 45. So 38 can be given as left child of 45.

(8) Node : 70

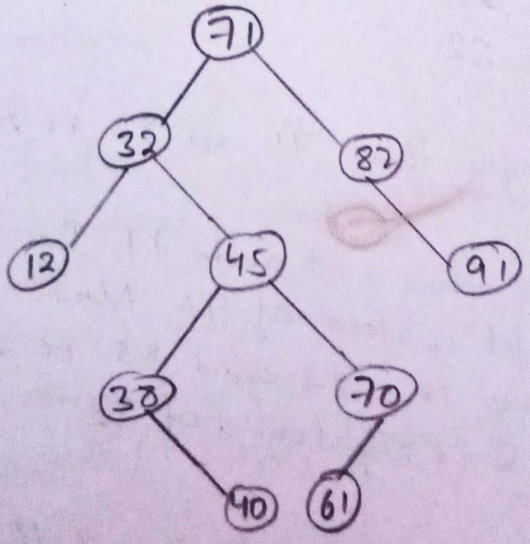
70 is less than 71, more than 32, more than 45
So 70 is right child of 45.

(9) Node : 40

40 is less than 71, more than 32, less than 45, and more than 38. So 40 is right child of 38.

(10) Node : 61

61 is less than 71, more than 32, more than 45 less than 70. So 61 is left child of 70.

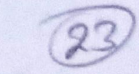


Binary Search Tree

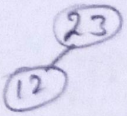
Q6. What are the preorder, inorder and postorder traversals for the following binary tree

23, 12, 11, 9, 6, 45, 32, 69, 56

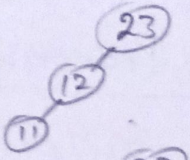
Ans:- Construction



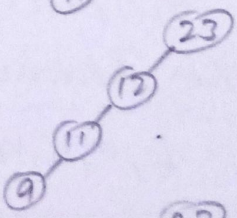
① Node : 23
Root Node



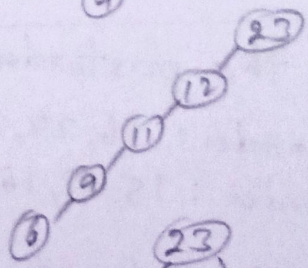
② Node : 12
12 is less than 23
12 is leftchild of 23



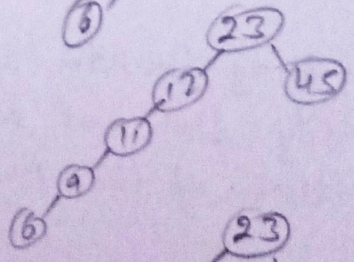
③ Node : 11
11 < 12
11 is left child of 12



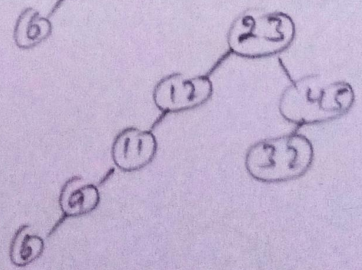
④ Node : 9
9 < 11
9 is left child of 11



⑤ Node : 6
6 < 9
6 is left child of 9

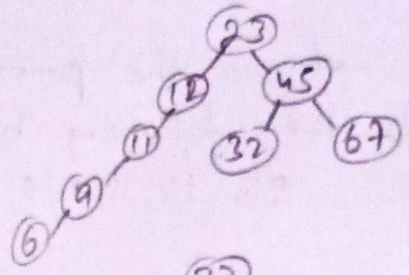


⑥ Node : 45
45 > 23
45 is right child of 23

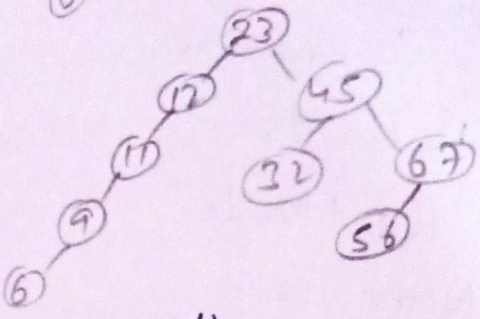


⑦ Node : 32
32 < 45
32 is leftchild of 45

⑧ Node: 67
67 > 45
67 is rightchild of 45



⑨ Node: 56
56 < 67
56 is leftchild of 67



Preorder Traversal (root-left-right)

23, 12, 11, 9, 6, 45, 32, 67, 56

Inorder Traversal (left-root-right)

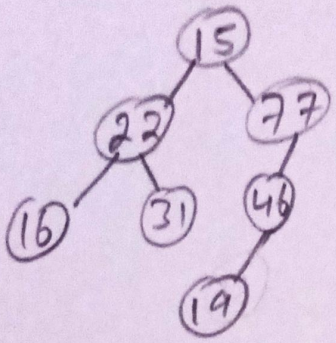
6, 9, 11, 12, 23, 32, 45, 56, 67

Postorder Traversal (left-right-root).

6, 9, 11, 12, 32, 56, 67, 45, 23

Q7. Find the postorda using:
Inorder: 16, 22, 31, 15, 46, 77, 19
Preorder: 15, 22, 16, 31, 77, 46, 19

Ans:- By using inorder and preorder, we can easily find out roots, left children & right children of the Binary Search Tree.



Using the tree, we can find out the postorder:

Postorder: 16, 31, 22, 46, 19, 77, 15