PRACTICAL 13

AIM: Write a Program to view a tree from right view.

SOURCE CODE:

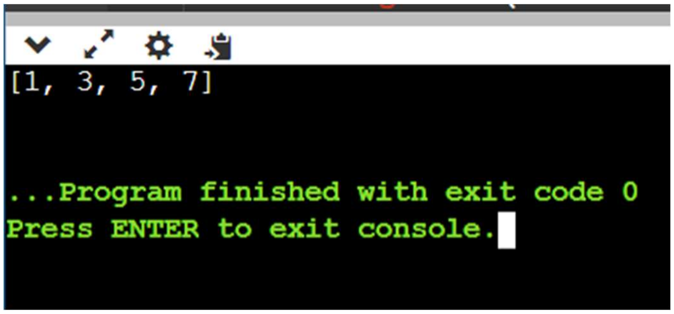
```
class BinaryTree:
    def __init__(self, val = 0, right = None, left = None):
        self.val = val
        self.left = left
        self.right = right

class Solution:
    def rightView(self, root, temp, res):
        if root is None:
            return
        if len(res) == temp:
            res.append(root.val)
        self.rightView(root.right, temp+1,res)
        self.rightView(root.left, temp+1,res)
        return res


obj1 = BinaryTree(1)
obj1.left = BinaryTree(2)
obj1.right = BinaryTree(3)
obj1.right.left = BinaryTree(4)
obj1.right.right = BinaryTree(5)
obj1.right.right.left = BinaryTree(7)
obj1.left.left = BinaryTree(9)

obj2 = Solution()
res1 = obj2.rightView(obj1, 0, [])
print(res1)
```

OUTPUT:



Result: The above code is successfully executed in Lab.

PRACTICAL 14

AIM: Write a Program for Building a Function ISVALID to VALIDATE.

SOURCE CODE:

```
class BinaryTree:
    def __init__(self, val = 0, right = None, left = None):
        self.val = val
        self.left = left
        self.right = right

class Solution:
    def validBST(self, root, lower = 0, upper = 1000):
        if root is None:
            return True
        if not lower < root.val < upper:
            return False
        return self.validBST(root.left, lower, root.val) and self.validBST(root.right, upper, root.val)

obj1 = BinaryTree(1)
obj1.left = BinaryTree(2)
obj1.right = BinaryTree(3)
obj1.left.left = BinaryTree(4)
obj1.left.right = BinaryTree(5)

obj1.right.left = BinaryTree(6)
obj1.right.right = BinaryTree(7)

obj2 = Solution()
res = obj2.validBST(obj1, 0, 1000)
print(res)
```

OUTPUT:

```
False


...Program finished with exit code 0
Press ENTER to exit console.
```

Result: The above code is successfully executed in Lab.

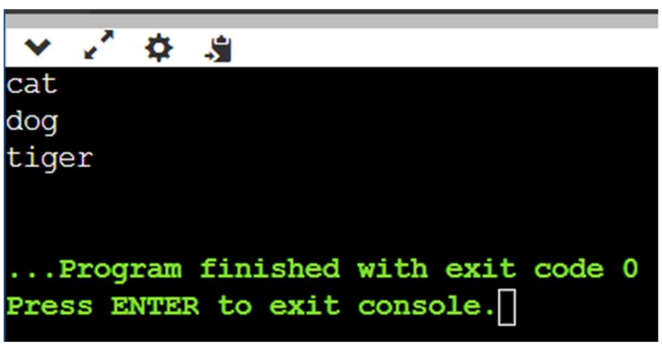
PRACTICAL 15

AIM: Write a Program for a basic hash function in a programming language of your choice. Demonstrate its usage to store and retrieve key-value pairs.

SOURCE CODE:

```
my_map = {}
my_map[5] = 'cat'
my_map[6] = 'dog'
my_map[7] = 'tiger'
print(my_map.get(5))
print(my_map.get(6))
print(my_map.get(7))
```

OUTPUT:



Result: The above code is successfully executed in Lab.

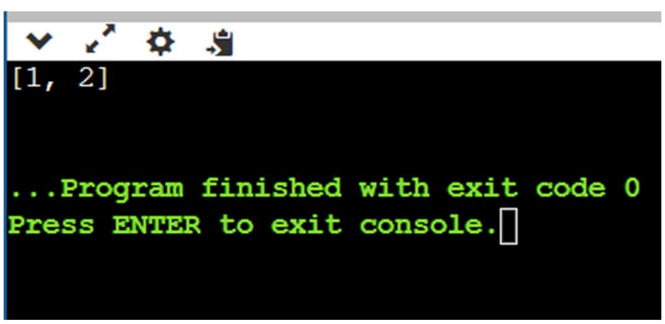
PRACTICAL 16

AIM: Write a Program to implement two sums using Hashmap.

SOURCE CODE:

```
ar = [2, 4, 6, 5]
target = 10
my_map = {}
for index, num in enumerate(ar):
    dif = target - num
    if dif in my_map:
        print([ my_map[dif], index])
        break
    my_map[num] = index
else:
    print("Not Found")
```

OUTPUT:



Result: The above code is successfully executed in Lab.

PRACTICAL 17

AIM: Write a Program to find distinct substrings in a string.

SOURCE CODE:

```
s = "abcd"
ds = {s[i:j] for i in range(len(s)) for j in range(i+1, len(s)+1)}
print(len(ds))
for i in ds:
    print(ds)
    break
```

OUTPUT:

```
10
{'c', 'bc', 'd', 'abcd', 'bcd', 'a', 'ab', 'abc', 'b', 'cd'}


...Program finished with exit code 0
Press ENTER to exit console.
```

Result: The above code is successfully executed in Lab.

PRACTICAL 18

AIM: Write a Program for an infix expression, and convert it to postfix notation. Use a queue to implement the Shunting Yard Algorithm for expression conversion.

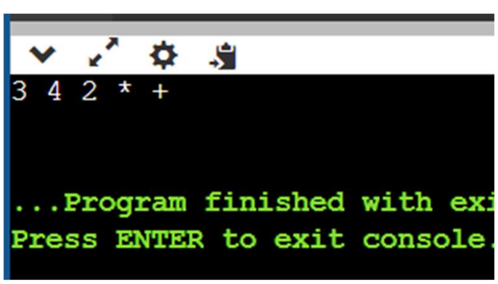
SOURCE CODE:

```
exp = "3 + 4 * 2"
opq = []
opr = []
pre ={'+':1, '-':1, '*':2, '/':2}

for i in exp.split():
    if i.isdigit():
        opq.append(i)
    else:
        while opr and opr[-1] in pre and pre[i] <= pre[opr[-1]]:
            opq.append(opr.pop())
        opr.append(i)

while opr:
    opq.append(opr.pop())
result = ' '.join(opq)
print(result)
```

OUTPUT:



Result: The above code is successfully executed in Lab.

PRACTICAL 19

AIM: Write a Program to find the Merge point of two linked lists(sorted).

SOURCE CODE:

```
class List:
    def __init__(self, data):
        self.data = data
        self.next = None

class Solution:
    #mP =merge point
    def mP(self, headA, headB):
        if not headA or not headB:
            return None
        a = headA
        b = headB
        while a != b:
            a = headB if not a else a.next
            b = headA if not b else b.next
            if a is None and b is None:
                return None
        return a

obj1 = List(1)
obj1.next = List(2)
obj1.next.next = List(3)
obj1.next.next.next = List(4)
obj1.next.next.next.next = List(9)

obj2 = List(6)
obj2.next = List(5)
obj2.next.next = obj1.next.next.next

obj3 = Solution()
res = obj3.mP(obj1, obj2)
print("merge point of this linked list is:", res.data)
```

OUTPUT:

```
merge point of this linked list is: 4


...Program finished with exit code 0
Press ENTER to exit console.
```

Result: The above code is successfully executed in Lab.

PRACTICAL 20

AIM: Write a Program to Build BST.

SOURCE CODE:

```
class BinaryTree:
    def __init__(self, val=0, right=None, left=None):
        self.val = val
        self.left = left
        self.right = right

class Solution:
    def insert(self, cur, val):
        if cur is None:
            return BinaryTree(val)
        elif val < cur.val:
            cur.left = self.insert(cur.left, val)
        else:
            cur.right = self.insert(cur.right, val)
        return cur

    def inOrder(self, cur):
        if cur:
            cur.left = self.inOrder(cur.left)
            print(cur.val)
            cur.right = self.inOrder(cur.right)

obj1 = None
obj2 = Solution()
obj1 = obj2.insert(obj1,8)
obj1 = obj2.insert(obj1,10)
obj1 = obj2.insert(obj1,7)
obj1 = obj2.insert(obj1,14)
obj1 = obj2.insert(obj1,11)
obj1 = obj2.insert(obj1,5)
obj1 = obj2.insert(obj1,15)

result = obj2.inOrder(obj1)
print(result)
```

OUTPUT:

```
5
7
8
10
11
14
15
None
```

Result: The above code is successfully executed in Lab.

PRACTICAL 21

AIM: Write a Program for finding the Product of the three largest Distinct Elements. Use a Priority Queue to efficiently find and remove the largest elements.

SOURCE CODE:

```
n = [5, 7, 3, 8, 3, 1, 0, 2]

if len(n) < 3:
    print("Increase the number of elements in the list.")
else:
    f = s = t = float('-inf')
    for i in n:
        if i > f:
            t = s
            s = f
            f = i
        elif i > s and i != f:
            t = s
            s = i
        elif i > t and i != f and i != s:
            t = i
    print(f, s, t)
    product = f * s * t
    print(product)
```

OUTPUT:

```
8 7 5
280


...Program finished with exit code 0
Press ENTER to exit console.
```

Result: The above code is successfully executed in Lab.