Q18. Count pairs with given sum.

i/p → {1, 5, 7, −1} , k = 6
O/p → 2

Brute force
Consider all the possible pairs by running 2 nested loops and if we find a pair having sum equal to the given sum, then simply increase the count of the count Pairs. At the end simply return the countPairs.

Time complexity = $O(n^2)$
Space complexity = $O(1)$

Optimal solution + Dry run

1) Create a map in which the frequency of elements will be stored.

1 : 1
5 : 1
7 : 1
−1 : 1

2) Traverse the array now and find whether k − arr[i] is present in the array & this we can get to know about from the map.

✓ index = 0
$m[6-1] = m[5] = 1 > 0$ and hence count ++.
Here count = 1.

✓ index = 1
m [6 − 5] = m [1] = 1 > 0 and hence again count ++.
Here count = 2.

✓ index = 2
m [6 − 7] = m [−1] = 1 > 0 and hence again count ++.
Here count = 3.

✓ index = 3
m [6 − (−1)] = m [7] = 1 > 0 & hence again count ++.
Here count = 4.

But here we have considered the pair twice.
(1, 5) and (5, 1) ⇒ 2 pairs but it should be considered only once.
Hence count / 2 will be the answer.

Note → Here we have to make sure that if k − arr [i] == arr [i] , then decrement count as we don't have to consider (arr [i], arr [i]) pair.

Code

```
int getPairs (int arr [], int n, int k) {
        // Create map to store frequency
        unordered_map <int, int> m;
        // Store the frequency of elements in map
        for (int i = 0 ; i < n ; i++) {
                m [arr [i]] ++ ;
        }
```

```
    // Initially countPairs = 0
    int countPairs = 0;
    for (int i = 0; i<n; i++) { //Element present
         if (m [k-arr [i]] != 0) {
              countPairs + = m [k-arr [i]];
         }
         // arr [i], arr [i] not to be considered
         if (k-arr [i] = = arr [i]) {
              countPairs --; // Decrement
                                   count pairs
         }
    }
    int ans = countPairs /2; // 2 times pair
    return ans;                have been considered
}
```

Time complexity = O(n)
Space complexity = O(n)