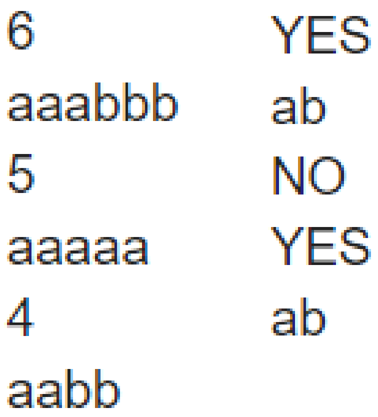
CSI2P(I) 2019 Spring

Midterm 1 Exercise

12130 – Oh! I find de way!

Description

- Good string: no letter appears strictly more than **n/2** times
- **minimum length of the good substring**
- **at most one answer** in each testcase
- **The input ends with EOF**



```
6         YES
aaabbb    ab
5         NO
aaaaa     YES
4         ab
aabb
```

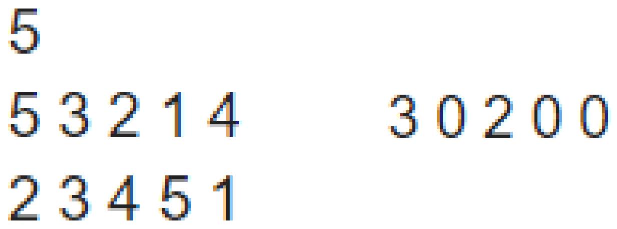
Simplify it!

If good substring exists, its length must be 2 and unique

12132 – too many watermelons

Description

- **eat all watermelon above the a_i watermelon indexed**



```
5
5 3 2 1 4        3 0 2 0 0
2 3 4 5 1
```

Hint

- Use an extra array to store whether the index of watermelon is eaten or not
- If the first watermelon was eaten according to the extra array in this round, it means that it will eat 0 watermelon.

Round 1

2

Input	Now	Valid
5	5	0
3	3	0
$\boxed{2}$	2	$\boxed{0}$
1	1	0
4	4	0

Output

Round 1

2

Input	Now	Valid
5	0	1
3	0	1
$\boxed{2}$	0	1
1	1	0
4	4	0

Output 3

Round 2

3

Input	Now	Valid
5	0	1
3	0	1
2	0	1
1	1	0
4	4	0

Output 3 0

Round 3

4

Input	Now	Valid
5	0	1
3	0	1
2	0	1
1	1	0
4	4	0

Output 3 0

Round 3

4

Input	Now	Valid
5	0	1
3	0	1
2	0	1
1	0	1
4	0	1

Output 3 0 2

Round 4

5

Input	Now	Valid
$\boxed{5}$	0	$\boxed{1}$
3	0	1
2	0	1
1	0	1
4	0	1

Output 3 0 2 0

Round 5

1

Input	Now	Valid
5	0	1
3	0	1
2	0	1
$\boxed{1}$	0	$\boxed{1}$
4	0	1

Output 3 0 2 0 0

12133 – Yes papa

Description

Input: Two string(ex: 1.ap , 2.pa)

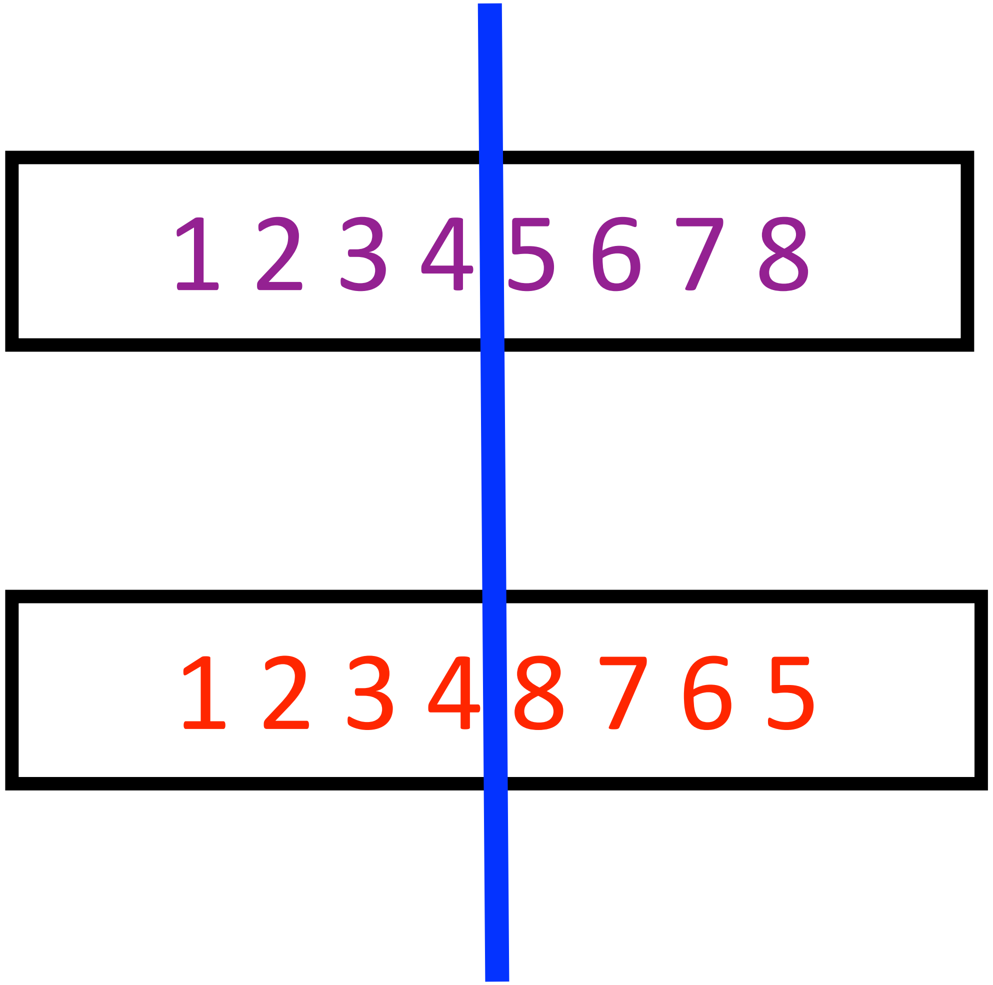
Output: if the two string is same

Rule:

- **The first string is equal to the second string**.
- If two string are not equal , you can divide them into two same size part **(a1 , a2 , b1 ,b2), and (a1 == b1) && (a2 == b2)) or ((a1 == b2) && (a2 == b1)**

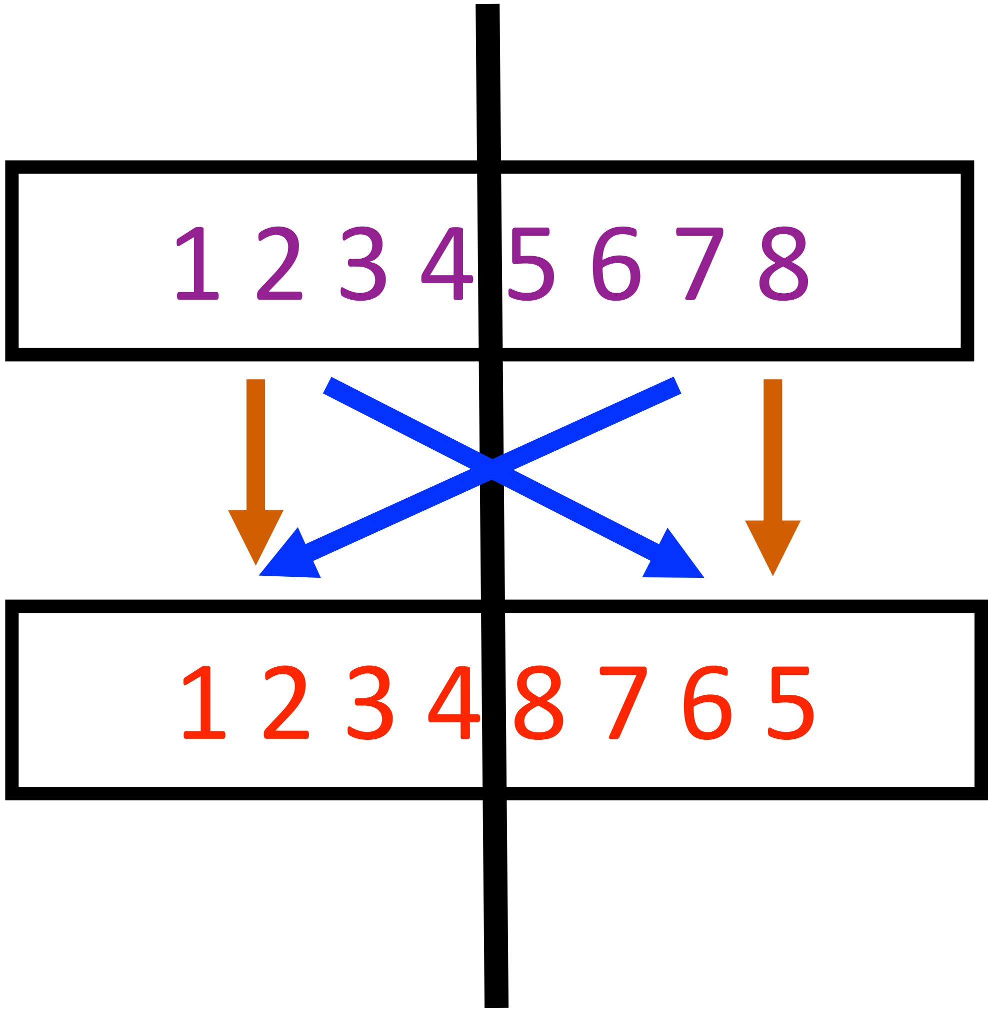
1 2 3 4 5 6 7 8

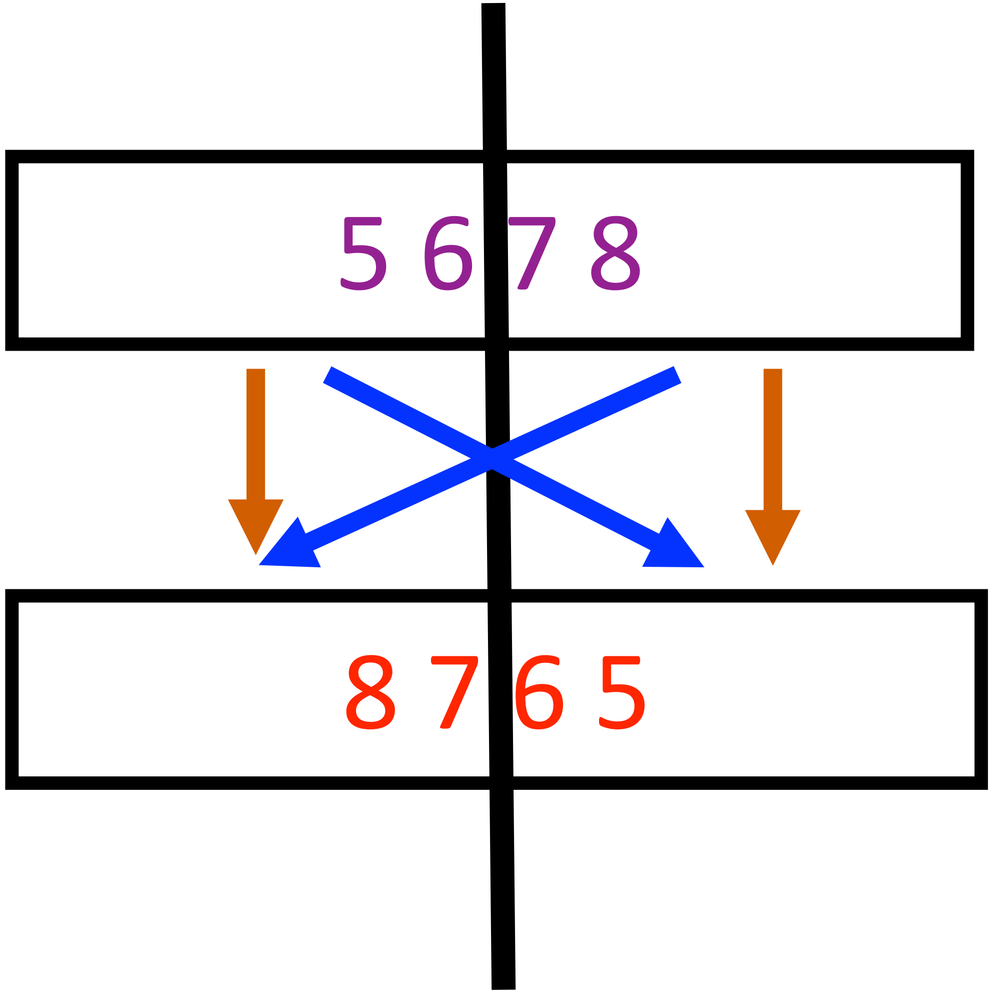
1 2 3 4 8 7 6 5



1 2 3 4 5 6 7 8

1 2 3 4 8 7 6 5

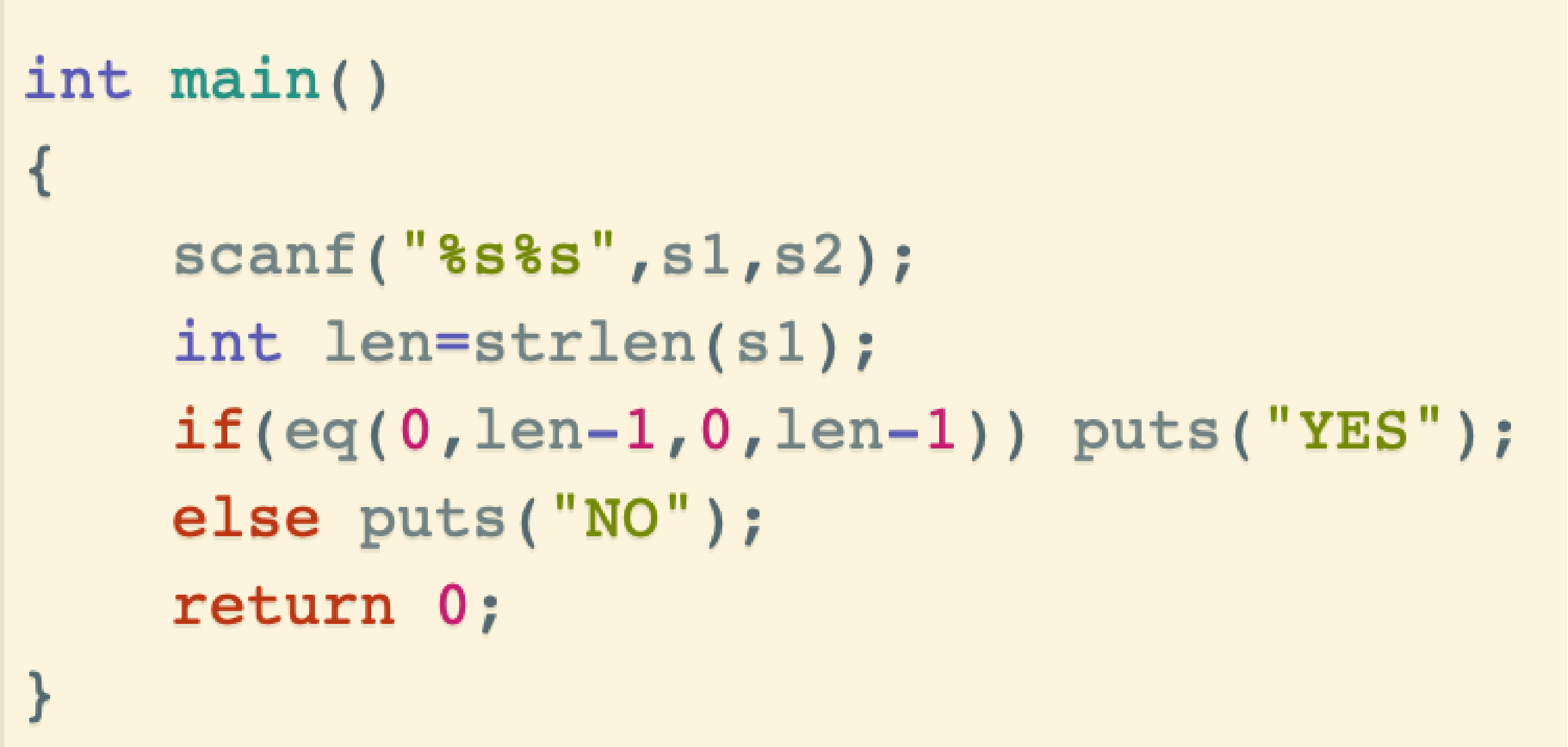

1 2 3 4 5 6 7 8
1 2 3 4 8 7 6 5



5678

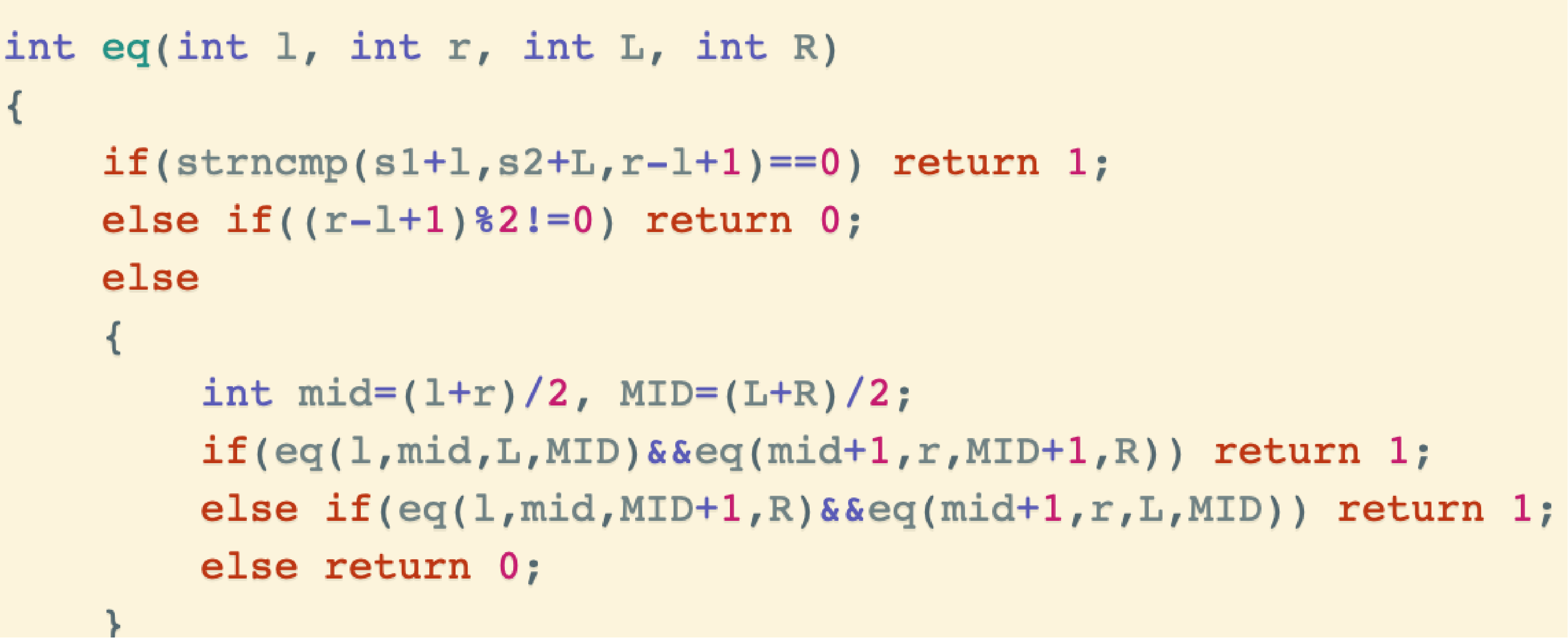
8765

- Concept: **Use recursion to check substring**



```
int main()
{
    scanf("%s%s",s1,s2);
    int len=strlen(s1);
    if(eq(0,len-1,0,len-1)) puts("YES");
    else puts("NO");
    return 0;
}
```

- Concept: **Use recursion to check substring**



```
int eq(int l, int r, int L, int R)
{
    if(strncmp(s1+l,s2+L,r-l+1)==0) return 1;
    else if((r-l+1)%2!=0) return 0;
    else
    {
        int mid=(l+r)/2, MID=(L+R)/2;
        if(eq(l,mid,L,MID)&&eq(mid+1,r,MID+1,R)) return 1;
        else if(eq(l,mid,MID+1,R)&&eq(mid+1,r,L,MID)) return 1;
        else return 0;
    }
```

12137 – Johnny Johnny

Description

- Input:
    - n: how many numbers you have to read
    - k: target number you have to choose numbers to equal to it
    - a_1 ~ a_n
- Output:
    - Number of available ways to pick numbers, which summation is equal to k

Solution of Recursion

- Argument of recursion

  Assume that we find possible solution from index 1 to index n(1 -> 2 -> … -> n)

  sum：summation right now (when entering this state of recursion)

  start：processing index right now (when entering this state of recursion)

- Termination condition

  sum == k

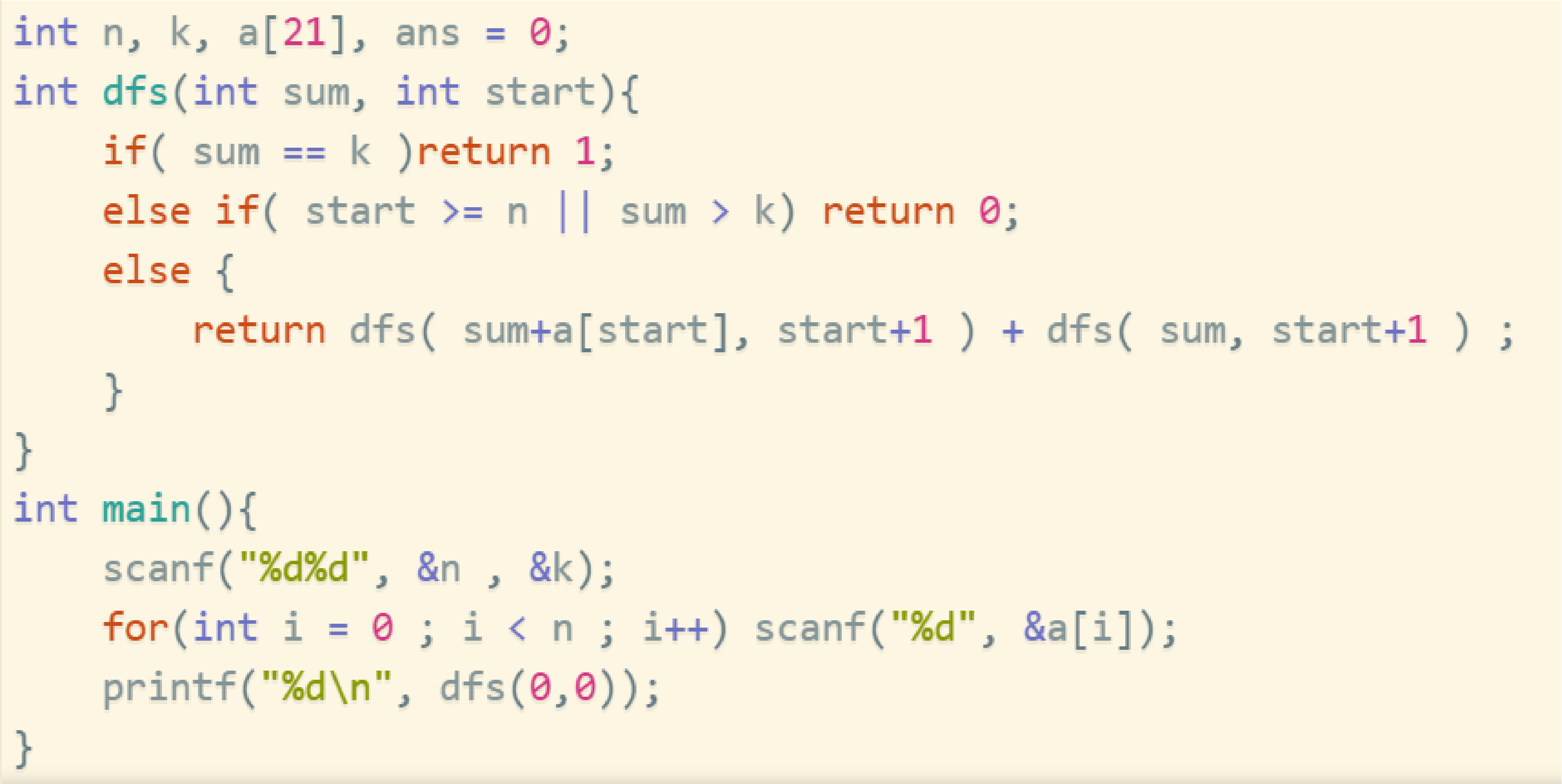
  start >= n || sum > k

- Recursion function

  Condition 1：Add a_i and move to next item a_{i+1}, simultaneously add a_i value to sum

  Condition 2：Not to add a_i and move to next item a_{i+1}

Solution of Code



```c
int n, k, a[21], ans = 0;
int dfs(int sum, int start){
    if( sum == k )return 1;
    else if( start >= n || sum > k) return 0;
    else {
        return dfs( sum+a[start], start+1 ) + dfs( sum, start+1 ) ;
    }
}
int main(){
    scanf("%d%d", &n , &k);
    for(int i = 0 ; i < n ; i++) scanf("%d", &a[i]);
    printf("%d\n", dfs(0,0));
}
```

12140 – HaSaKi!!

Main Idea: Find the amount of substring @ [l ,r]

Substring: In string “Abababababaab”, a substring could be ‘a’, ’b’, ‘ab’, ‘aba’, ...

Common Error: Brute force, attempt to iterate all possibilities -> TLE !

RECALL

Aggregate answers in an array (appeared in our labs before)

For example, counting, 1 2 2 3 **3** 3 3 **3** 3 4, indicates 0 (Not increasing)

Specifically, ans[i+1] = **sth**[i] + ans[i] (**sth**: some useful functions or efficient transformation)

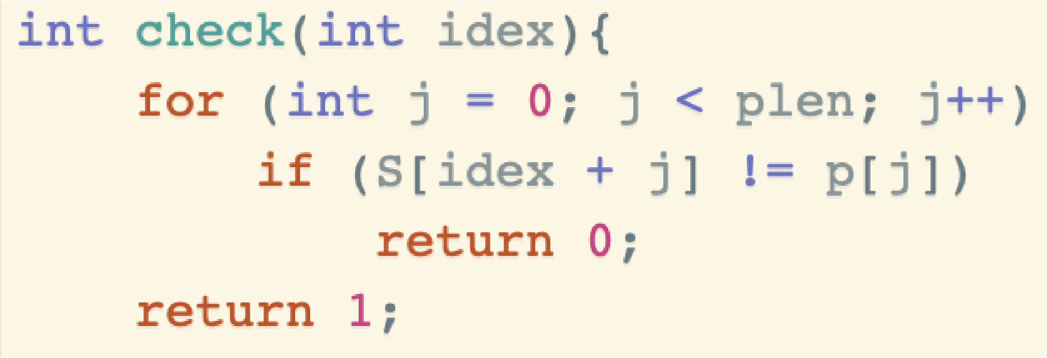
sth: something useful -> “check” function

We **check** if substring p exists in string **S**

A check is valid (return 1) when every ‘char’ in **p** are equal to those of **S**

The amount of ‘char’ depends on substring **p**, which is ‘plen’, a length of **p**

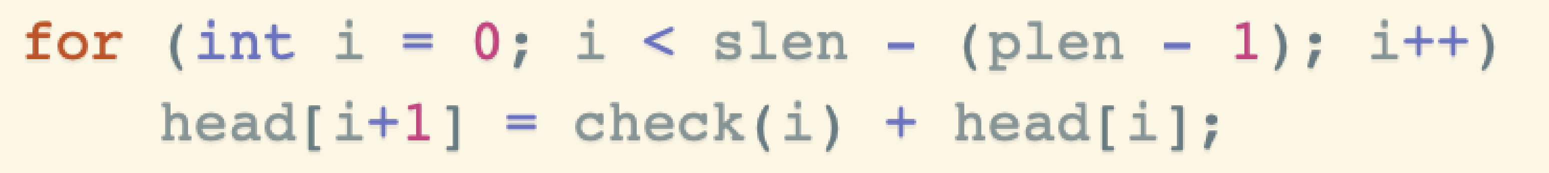
In others words, if any ‘char’ mismatches, the check fails



```
int check(int idex){
    for (int j = 0; j < plen; j++)
        if (S[idex + j] != p[j])
            return 0;
    return 1;
```

(S: input string; p: an objective substring)

Aggregate: (store answers in ‘head’ array)



```
for (int i = 0; i < slen - (plen - 1); i++)
    head[i+1] = check(i) + head[i];
```

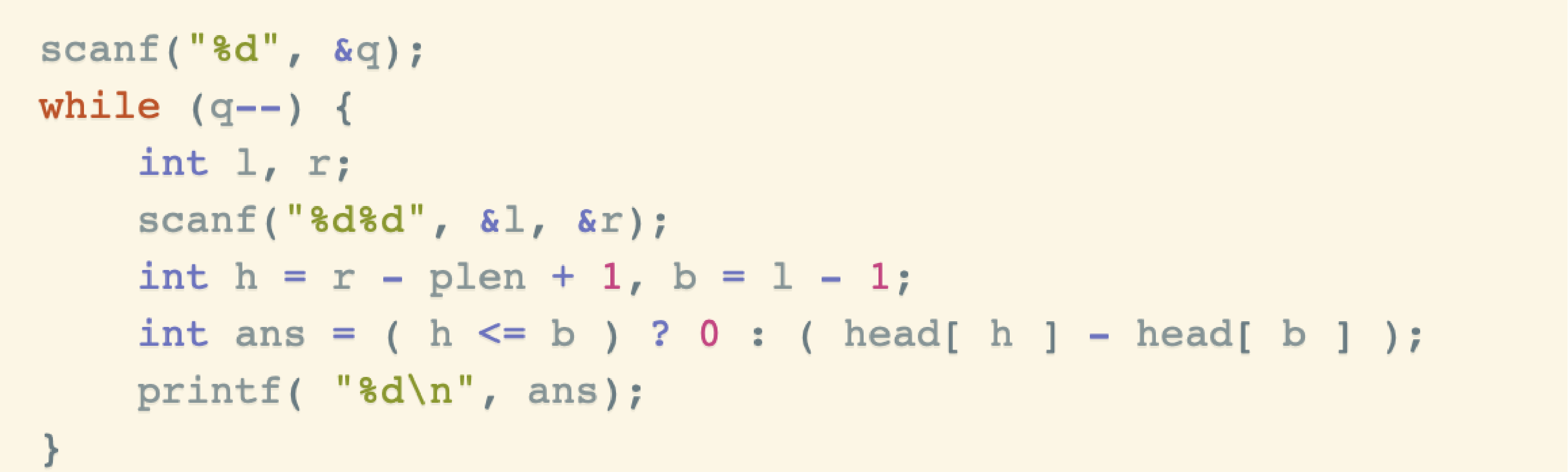
We consider all of the possible position that a substring **p** may exist

It starts from the beggining of **S**, ends in ‘slen’ - ‘plen’ + 1

For example, aaabb.

Checking substring **bbb** @ bb is trivial, since the length of **bbb** is > len(bb)

In interval [l, r], we have:



```
scanf("%d", &q);
while (q--) {
    int l, r;
    scanf("%d%d", &l, &r);
    int h = r - plen + 1, b = l - 1;
    int ans = ( h <= b ) ? 0 : ( head[ h ] - head[ b ] );
    printf( "%d\n", ans);
}
```

We calculate the answer through our ‘head’. Since we’ve already stored the answers before, it’s easy to get the correct answer by simply subtraction.

Note that, if the length of p is > the given interval [l, r], Mr. Yasuoo will gg. (The answer should be 0, meaningless)