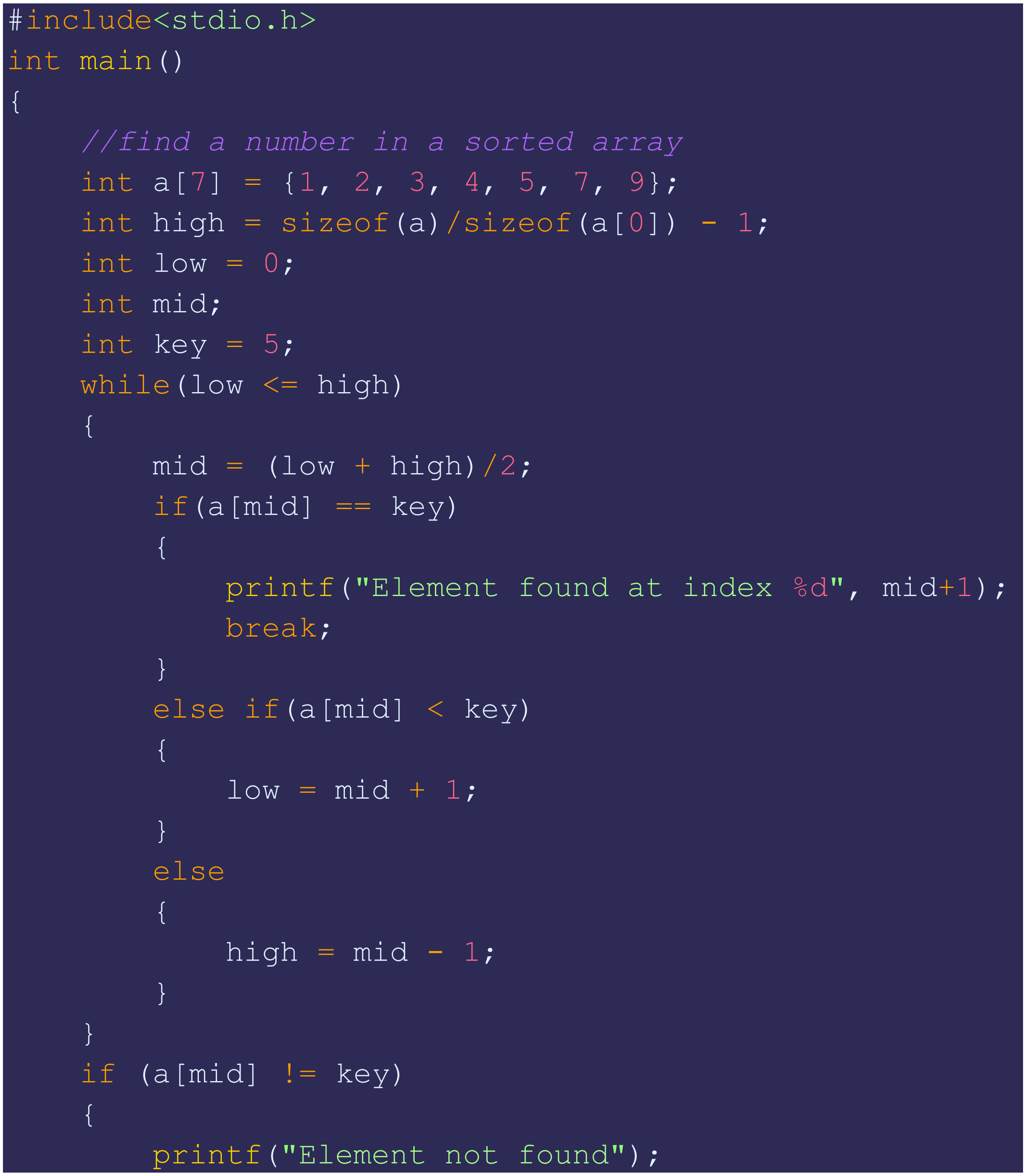
Binary Search

Find an Element in a Sorted Array
You are given a sorted array of numbers. Use binary search to find the position of a specific number. If the number is not in the array, return -1.



```c
#include<stdio.h>
int main()
{
    //find a number in a sorted array
    int a[7] = {1, 2, 3, 4, 5, 7, 9};
    int high = sizeof(a)/sizeof(a[0]) - 1;
    int low = 0;
    int mid;
    int key = 5;
    while(low <= high)
    {
        mid = (low + high)/2;
        if(a[mid] == key)
        {
            printf("Element found at index %d", mid+1);
            break;
        }
        else if(a[mid] < key)
        {
            low = mid + 1;
        }
        else
        {
            high = mid - 1;
        }
    }
    if (a[mid] != key)
    {
        printf("Element not found");
```

```
            return -1;
        }
        return 0;
}
```

Find the First Position of a Number

Given a sorted array where a number can appear multiple times, find the first position of a specific number. If the number is not present, return -1.

```
#include<stdio.h>
int main()
{
    //find the first position of a number
    int a[7] = {1, 3, 3, 3, 5,7, 9};
    int high = sizeof(a)/sizeof(a[0]) - 1;
    int low = 0;
    int mid = (low + high)/2;
    int key = 6;
    while(low <= high)
    {
        if(a[mid] == key)
        {
            if(a[mid-1] == key)
            {
                high = mid - 1;
            }
            else
            {
                printf("Element found at index %d",
mid+1);
                break;
            }
        }
        else if(a[mid] < key)
        {
            low = mid + 1;
        }
        else
        {
            high = mid - 1;
        }
        mid = (low + high)/2;
    }
    if (low > high)
    {
        printf("Element not found");
        return -1;
    }
    return 0;
}
```

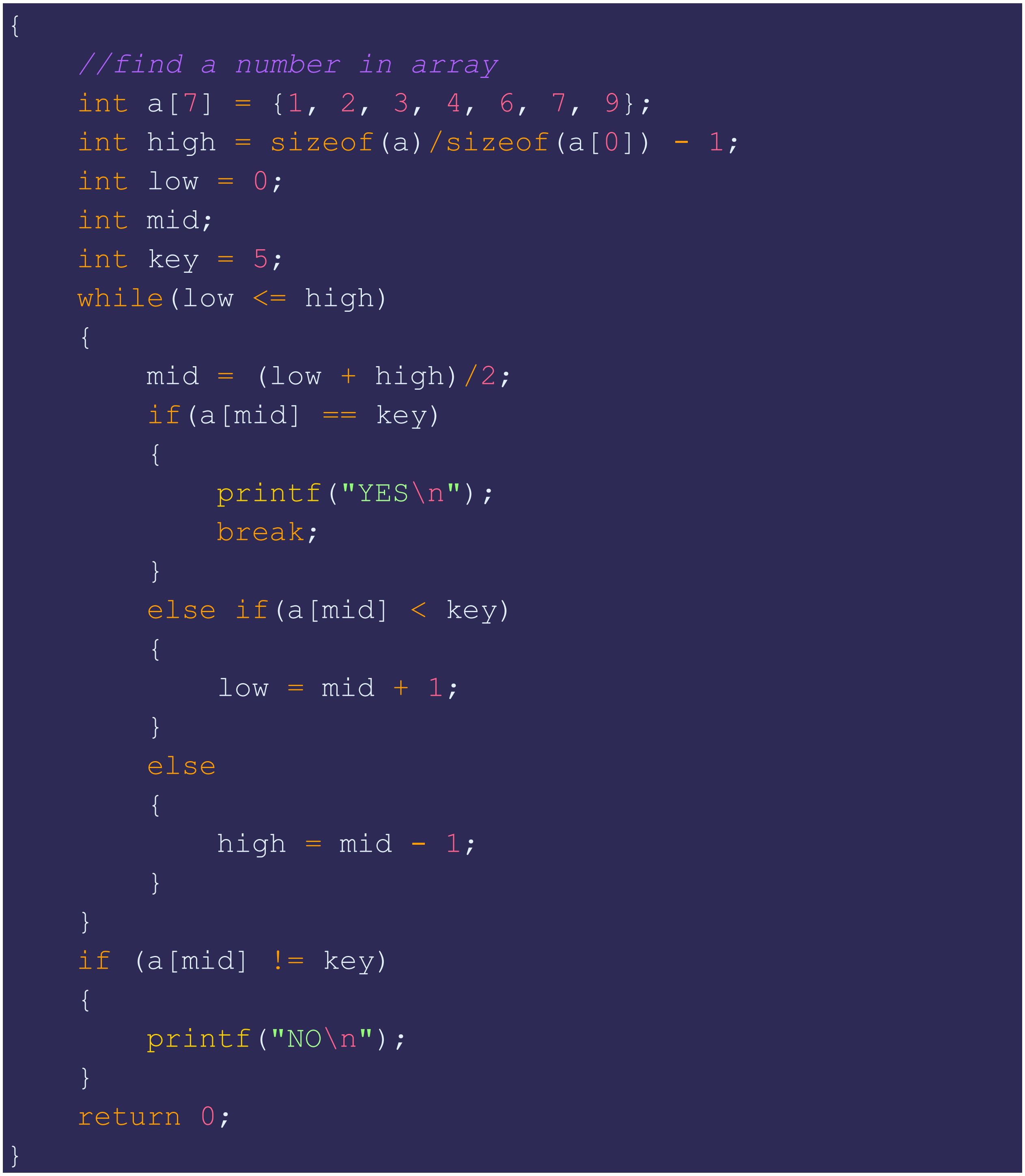
Find the Last Position of a Number

Find the last position of a specific number in a sorted array. Return -1 if the number is not found.

```
#include<stdio.h>
int main()
{
    //find the last position of a number
    int a[7] = {1, 2, 3, 3, 3, 7, 9};
    int high = sizeof(a)/sizeof(a[0]) - 1;
    int low = 0;
    int key = 3;
    int mid;
    while(low <= high)
    {
        mid = (low + high)/2;
        if(a[mid] == key)
        {
            if(mid < high && a[mid+1] == key)
            {
                low = mid + 1;
            }
            else
            {
                printf("Element found at index %d",
mid+1);
                break;
            }
        }
        else if(a[mid] < key)
        {
            low = mid + 1;
        }
        else
        {
            high = mid - 1;
        }
    }
    if (low > high)
    {
        printf("Element not found");
        return -1;
    }
    return 0;
}
```

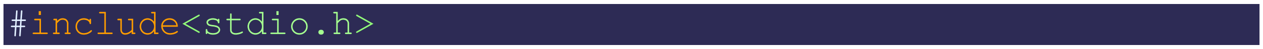
Check if a Number Exists
Given a sorted array, check if a number exists in the array. Return "YES" if the number is present, otherwise return "NO."

```
#include<stdio.h>
int main()
```

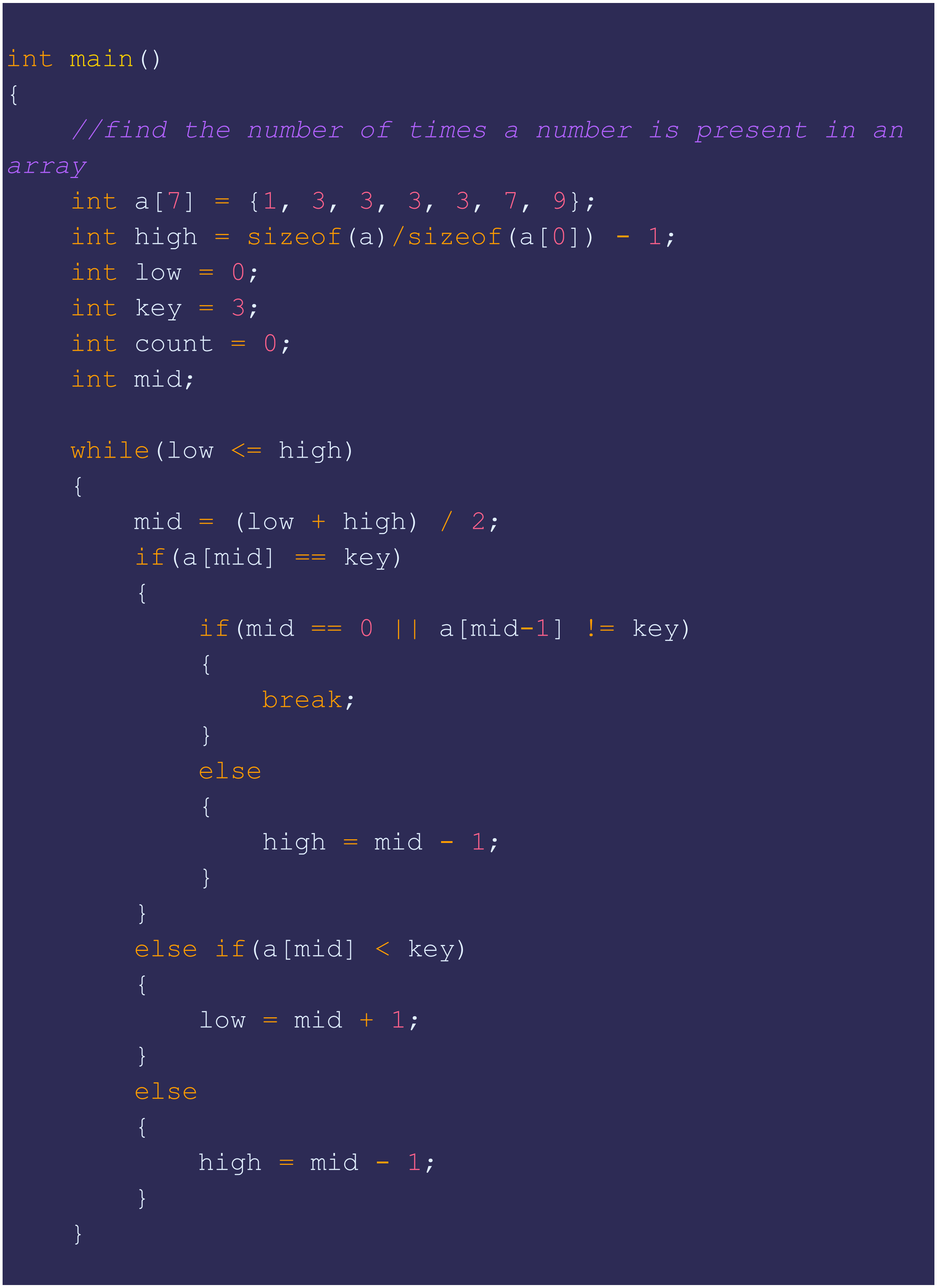


```
{
    //find a number in array
    int a[7] = {1, 2, 3, 4, 6, 7, 9};
    int high = sizeof(a)/sizeof(a[0]) - 1;
    int low = 0;
    int mid;
    int key = 5;
    while(low <= high)
    {
        mid = (low + high)/2;
        if(a[mid] == key)
        {
            printf("YES\n");
            break;
        }
        else if(a[mid] < key)
        {
            low = mid + 1;
        }
        else
        {
            high = mid - 1;
        }
    }
    if (a[mid] != key)
    {
        printf("NO\n");
    }
    return 0;
}
```

Count How Many Times a Number Appears
In a sorted array, count how many times a specific number appears. Use binary search to solve it efficiently.



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



```
int main()
{
    //find the number of times a number is present in an
array
    int a[7] = {1, 3, 3, 3, 3, 7, 9};
    int high = sizeof(a)/sizeof(a[0]) - 1;
    int low = 0;
    int key = 3;
    int count = 0;
    int mid;

    while(low <= high)
    {
        mid = (low + high) / 2;
        if(a[mid] == key)
        {
            if(mid == 0 || a[mid-1] != key)
            {
                break;
            }
            else
            {
                high = mid - 1;
            }
        }
        else if(a[mid] < key)
        {
            low = mid + 1;
        }
        else
        {
            high = mid - 1;
        }
    }
    if(low > high)
    {
        printf("Element %d is not present in the
array\n", key);
        return 0;
    }

    int first = mid;
    count = 1;
    for(int i = first + 1; i < sizeof(a)/sizeof(a[0]) &&
a[i] == key; i++)
    {
        count++;
    }

    printf("Element %d is present %d times\n", key,
count);

    return 0;
}
```

Find the Smallest Number Greater than X

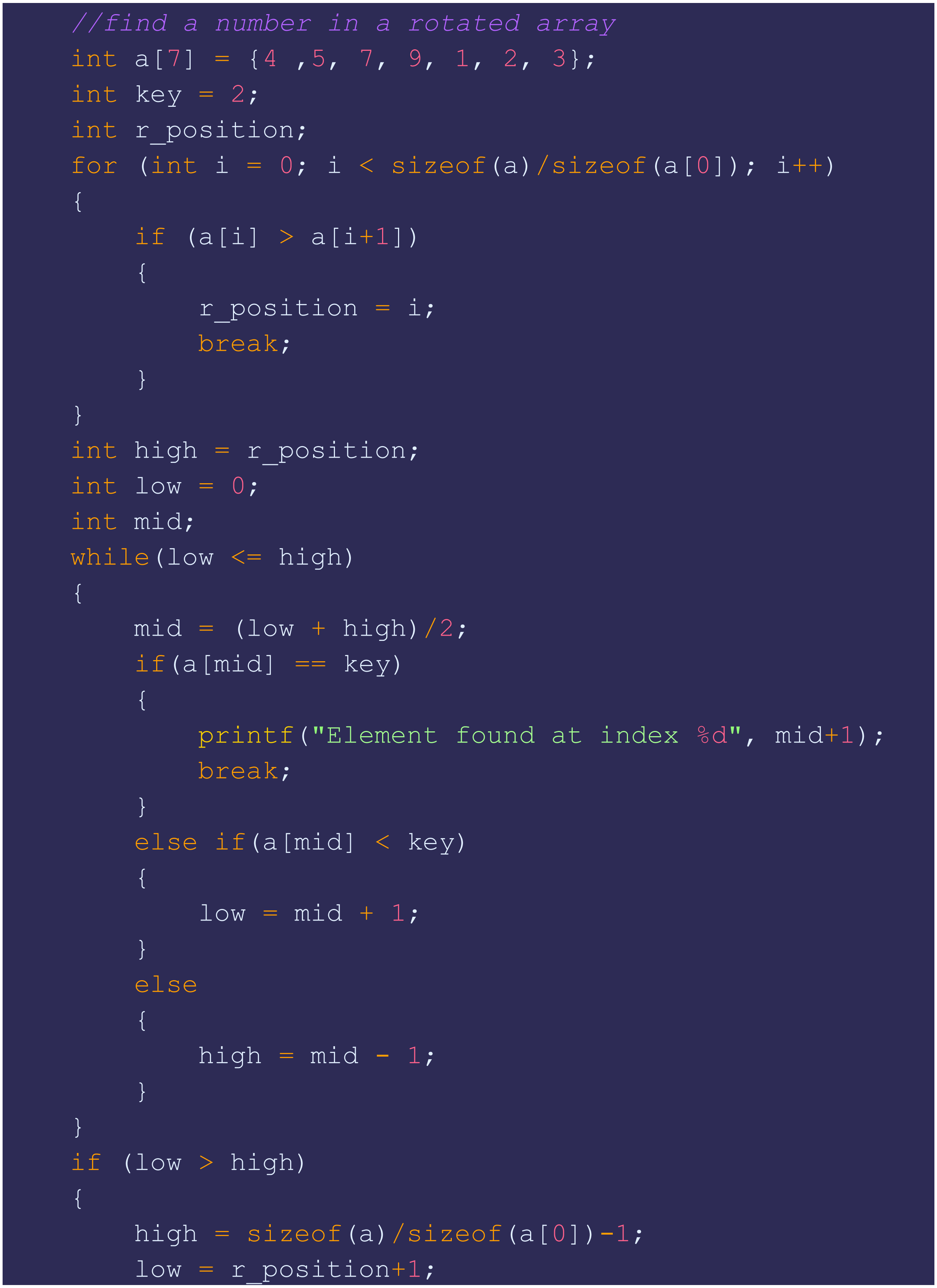
Given a sorted array, find the smallest number that is larger than a given number X. If no such number exists, return -1.

```
#include<stdio.h>
int main()
{
    //find the smallest number greater than a number
    int a[7] = {1, 2, 3, 5, 5, 7, 9};
    int high = sizeof(a)/sizeof(a[0]) - 1;
    int low = 0;
    int mid;
    int key = 5;
    while(low <= high)
    {
        mid = (low + high)/2;
        if(a[mid] == key)
        {
            int x = 1;
            while (a[mid+x] == key)
            {
                x++;
            }
            printf("The smallest num greater than %d is 
%d", key, a[mid+x]);
            break;
        }
        else if(a[mid] < key)
        {
            low = mid + 1;
        }
        else
        {
            high = mid - 1;
        }
    }
    if (a[mid] != key)
    {
        printf("Element not found");
        return -1;
    }
    return 0;
}
```

Search in a Rotated Array

A sorted array is rotated at an unknown point. For example, [4, 5, 6, 7, 1, 2, 3]. Search for a specific number in this array using binary search.

```
#include<stdio.h>
int main()
{
```



```
    //find a number in a rotated array
    int a[7] = {4 ,5, 7, 9, 1, 2, 3};
    int key = 2;
    int r_position;
    for (int i = 0; i < sizeof(a)/sizeof(a[0]); i++)
    {
        if (a[i] > a[i+1])
        {
            r_position = i;
            break;
        }
    }
    int high = r_position;
    int low = 0;
    int mid;
    while(low <= high)
    {
        mid = (low + high)/2;
        if(a[mid] == key)
        {
            printf("Element found at index %d", mid+1);
            break;
        }
        else if(a[mid] < key)
        {
            low = mid + 1;
        }
        else
        {
            high = mid - 1;
        }
    }
    if (low > high)
    {
        high = sizeof(a)/sizeof(a[0])-1;
        low = r_position+1;
        while(low <= high)
        {
            mid = (low + high)/2;
            if(a[mid] == key)
            {
                printf("Element found at index %d",
mid+1);
                break;
            }
            else if(a[mid] < key)
            {
                low = mid + 1;
            }
            else
            {
                high = mid - 1;
            }
        }
    }
    if (low > high)
    {
        printf("Element not found");
    }
    return 0;
}
```

Linear search

1. **Find an Element in an Array**
   Search for a specific number in an unsorted array. Return its position if found or -1 if it doesn't exist.

```
#include<stdio.h>
int main()
{
    int a[5] = {9, 2, 8, 3, 5};
    int key = 3, flag = 0;
    for (int i = 0; i < sizeof(a)/sizeof(a[0]); i++)
    {
        if (a[i] == key)
        {
            printf("%d found at index %d\n",key, i+1);
            flag = 1;
            break;
        }
    }
    if (flag == 0)
    {
        printf("Element not found\n");
        return -1;     }

    return 0;
}
```

2. **Count Occurrences of a Number**
   Count how many times a given number appears in an array. Return the count or 0 if the number is not present.

```
#include<stdio.h>
int main()
{
    int a[5] = {9, 2, 3, 3, 5};
    int key = 3, flag = 0, count = 0;
    for (int i = 0; i < sizeof(a)/sizeof(a[0]); i++)
    {
        if (a[i] == key)
        {
            flag = 1;
            count++;
        }
    }
    if (flag == 0)
    {
        printf("Element not found\n");
        return -1;
    }else{
        printf("%d found %d times\n", key, count);
    }

    return 0;
}
```

3. **Check if a Number Exists**
   Check if a specific number is present in an array. Return "YES" if it exists and "NO" otherwise.

```
#include<stdio.h>
int main()
{
    int a[5] = {9, 2, 8, 3, 5};
    int key = 3, flag = 0;
    for (int i = 0; i < sizeof(a)/sizeof(a[0]); i++)
    {
        if (a[i] == key)
        {
            printf("YES\n");
            flag = 1;
            break;
        }
    }
    if (flag == 0)
    {
        printf("NO\n");
        return -1;     }

    return 0;
}
```

5. **Find the First and Last Position of a Number**
   Find the first and last position of a given number in an array. If the number is not present, return -1 for both.

```c
#include<stdio.h>
int main()
{
    int a[5] = {9, 3, 8, 3, 3};
    int key = 3, flag = 0;
    int first_pos = 0,last_pos =  0;
    for (int i = 0; i < sizeof(a)/sizeof(a[0]); i++)
    {
        if (a[i] == key)
        {
            flag = 1;
            if (first_pos == 0)
            {
                first_pos = i+1;
            }
            if (first_pos != 0)
            {
                last_pos = i+1;
            }

        }
    }
    if (flag == 0)
    {
        printf("Element not found\n");
        return -1;
    }else
    {
        printf("First occurence of %d is at position 
%d\n",key,first_pos);
```

```
        printf("Last occurence of %d is at position %d\n",key,last_pos);
    }

    return 0;
}
```

6. **Find All Even Numbers**
   Traverse the array and collect all even numbers in a new array. Return the list of even numbers.

```
#include<stdio.h>
int main()
{
    int a[5] = {9, 2, 8, 3, 5};
    int count = 0;
    for (int i = 0; i < sizeof(a)/sizeof(a[0]); i++)
    {
        if (a[i] % 2 == 0)
        {
            count++;
        }
    }
    int even[count];
    int j = 0;
    for (int i = 0; i < sizeof(a)/sizeof(a[0]); i++)
    {
        if (a[i] % 2 == 0)
        {
            even[j] = a[i];
            j++;
        }
    }
    for (int i = 0; i < count; i++)
    {
        printf("%d ", even[i]);
    }

    return 0;
}
```

7. **Find the Second Largest Number**
   Find the second largest number in the array using a single traversal.

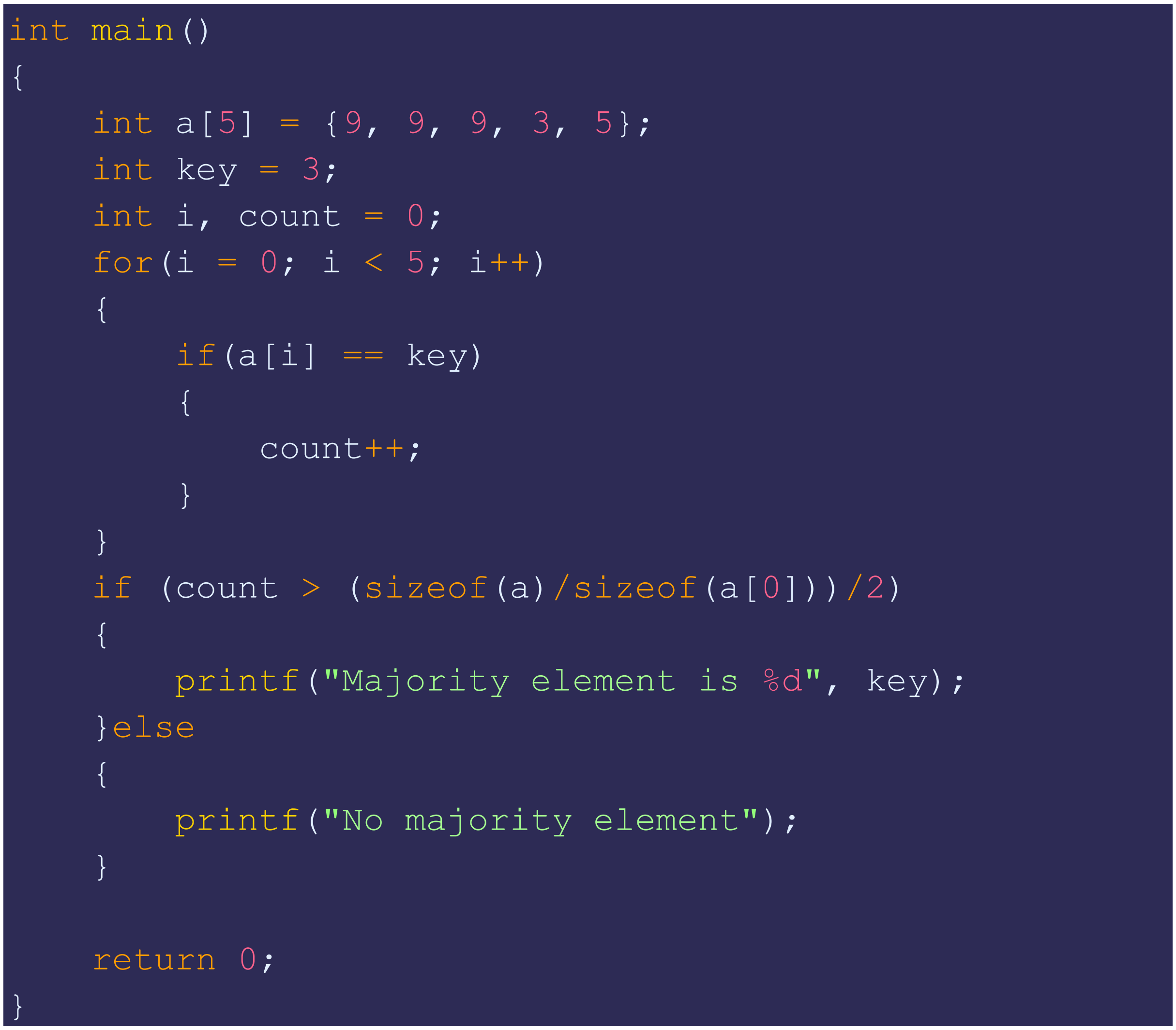
```
#include<stdio.h>
int main()
{
    int a[5] = {9, 2, 8, 3, 5};
    int max1=a[0], max2=-1;
    for (int i = 0; i < sizeof(a)/sizeof(a[0]); i++)
    {
        if (a[i] > max1)
        {
            max1 = a[i];
        }
        if (a[i] > max2 && a[i] < max1)
        {
            max2 = a[i];
        }

    }
    printf("The second largest element in the array is:
%d", max2);

    return 0;
}
```

8. **Find the Majority Element**
   Find the element that appears more than half the times in the array. If no such element exists, return -1.

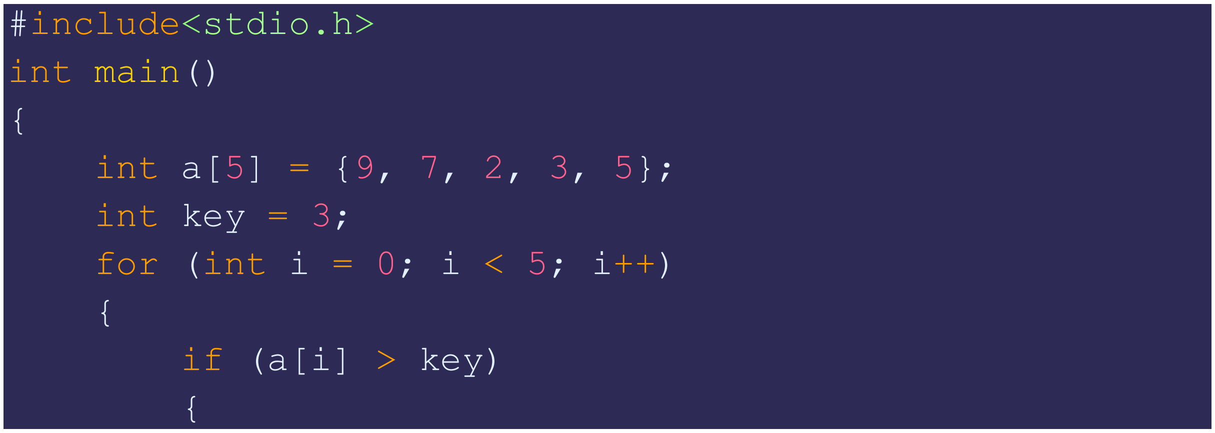
```
#include<stdio.h>
```



```c
int main()
{
    int a[5] = {9, 9, 9, 3, 5};
    int key = 3;
    int i, count = 0;
    for(i = 0; i < 5; i++)
    {
        if(a[i] == key)
        {
            count++;
        }
    }
    if (count > (sizeof(a)/sizeof(a[0]))/2)
    {
        printf("Majority element is %d", key);
    }else
    {
        printf("No majority element");
    }

    return 0;
}
```

9. **Find All Numbers Greater than a Given Number**
   Find all numbers in the array that are greater than a specified number. Return a list of such numbers.



```c
#include<stdio.h>
int main()
{
    int a[5] = {9, 7, 2, 3, 5};
    int key = 3;
    for (int i = 0; i < 5; i++)
    {
        if (a[i] > key)
        {
            printf("%d ", a[i]);
        }
    }
    printf("\n");
    return 0;
}
```

10. **Find the Closest Number to a Target**
    Find the number in the array that is closest to a given target number. If there are ties, return any one of them.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a[5] = {9, 7, 2, 3, 5};
    int target = 4;
    int closest = a[0];
    int diff = abs(a[0] - target);
    for (int i = 1; i < 5; i++)
    {
        if (abs(a[i] - target) < diff)
        {
            diff = abs(a[i] - target);
            closest = a[i];
        }
    }
    printf("Closest number to %d is %d\n", target,
closest);
    return 0;
}
```