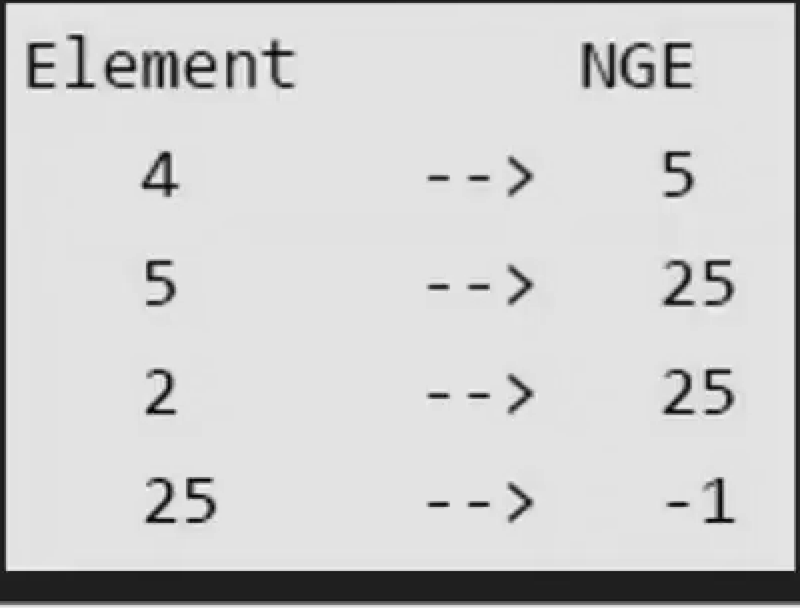
Next Greater Element

- Given an array, print the Next Greater Element (NGE) for every element.
- The Next greater Element for an element x is the first greater element on the right side of x in array.
- Elements for which no greater element exist, consider next greater element as -1.

Next Greater Element

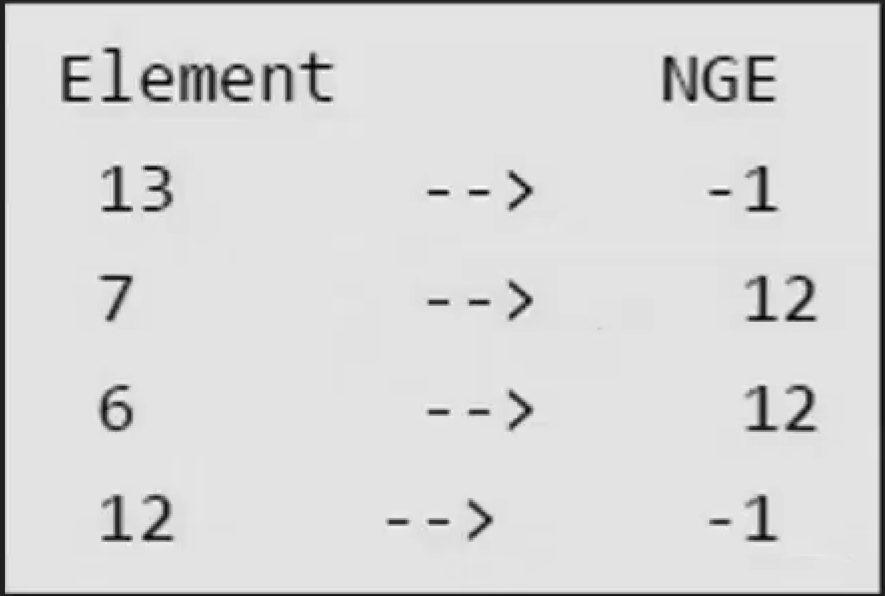
- For any array, rightmost element always has next greater element as -1.
- For an array which is sorted in decreasing order, all elements have next greater element as -1.
- For the input array [4, 5, 2, 25}, the next greater elements for each element are as follows.



```
Element        NGE
   4      -->   5
   5      -->   25
   2      -->   25
   25     -->   -1
```

Next Greater Element

- For the input array [13, 7, 6, 12}, the next greater elements for each element are as follows.

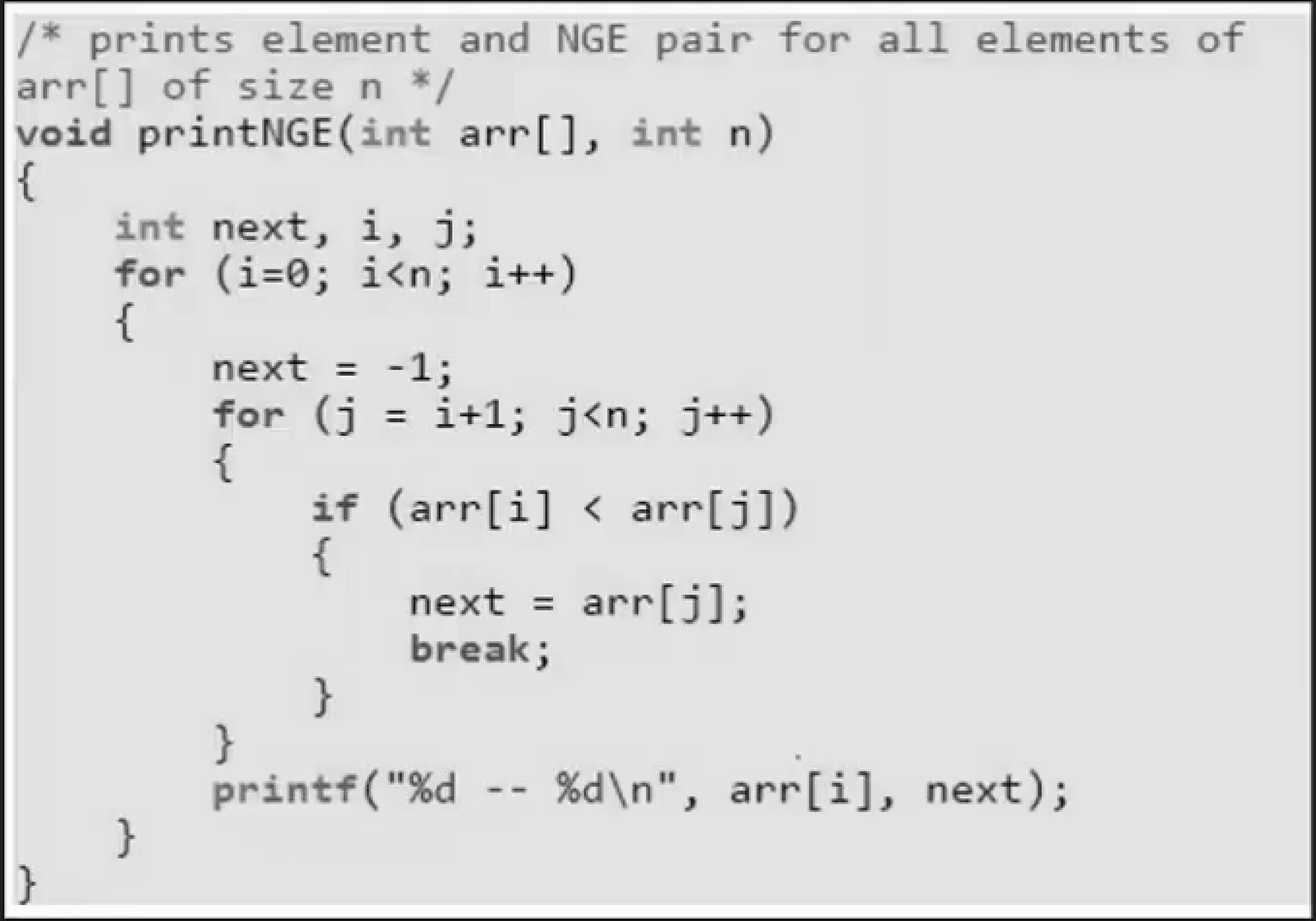


```
Element            NGE
 13          -->    -1
 7           -->     12
 6           -->     12
 12        -->      -1
```

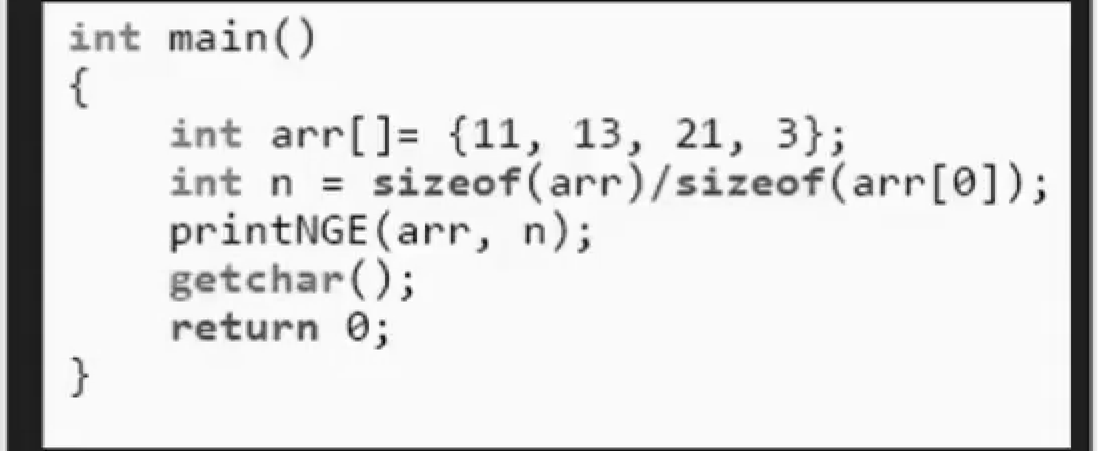
Method 1 (Simple)

- Use two loops:
  - The outer loop picks all the elements one by one.
  - The inner loop looks for the first greater element for the element picked by outer loop.
  - If a greater element is found then that element is printed as next, otherwise -1 is printed.

Method 1 (Simple)



```
/* prints element and NGE pair for all elements of
arr[] of size n */
void printNGE(int arr[], int n)
{
    int next, i, j;
    for (i=0; i<n; i++)
    {
        next = -1;
        for (j = i+1; j<n; j++)
        {
            if (arr[i] < arr[j])
            {
                next = arr[j];
                break;
            }
        }
        printf("%d -- %d\n", arr[i], next);
    }
}
```



```
int main()
{
    int arr[]= {11, 13, 21, 3};
    int n = sizeof(arr)/sizeof(arr[0]);
    printNGE(arr, n);
    getchar();
    return 0;
}
```

Method 2 (Using Stack)

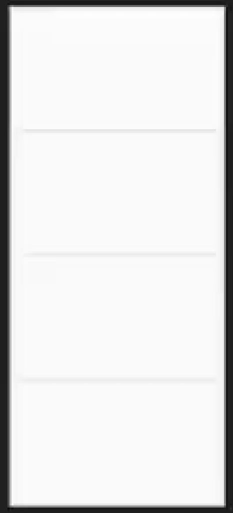
1) Push the first element to stack.

2) Pick rest of the elements one by one and follow following steps in loop.
    1) Mark the current element as next.
    2) If stack is not empty, then pop an element from stack and compare it with next.
    3) If next is greater than the popped element, then next is the next greater element for the popped element.
    4) Keep popping from the stack while the popped element is smaller than next. next becomes the next greater element for all such popped elements
    5) If next is smaller than the popped element, then push the popped element back.

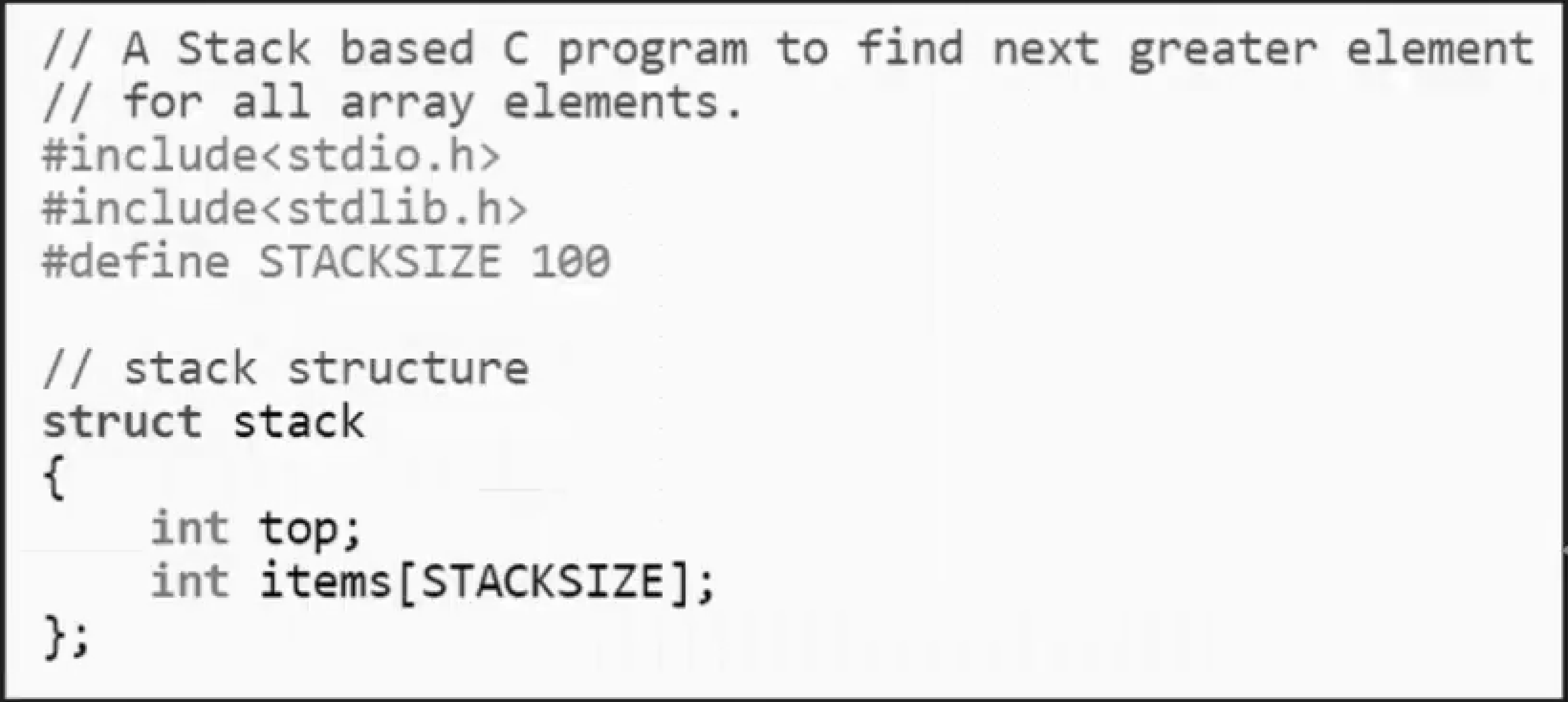
3) After the loop in step 2 is over, pop all the elements from stack and print -1 as next element for them.

Method 2 (Using Stack)

4	5	2	25

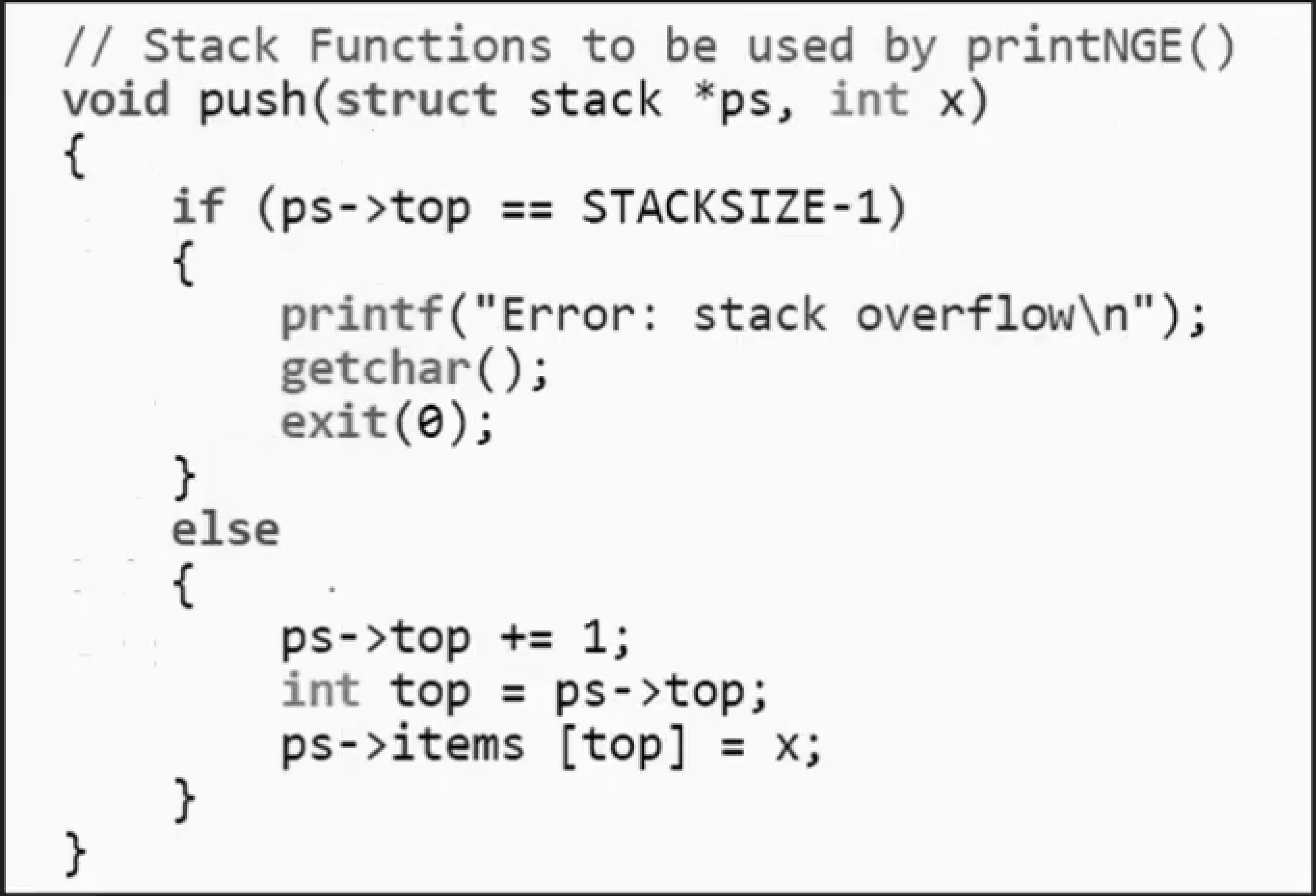


Method 2 (Using Stack)



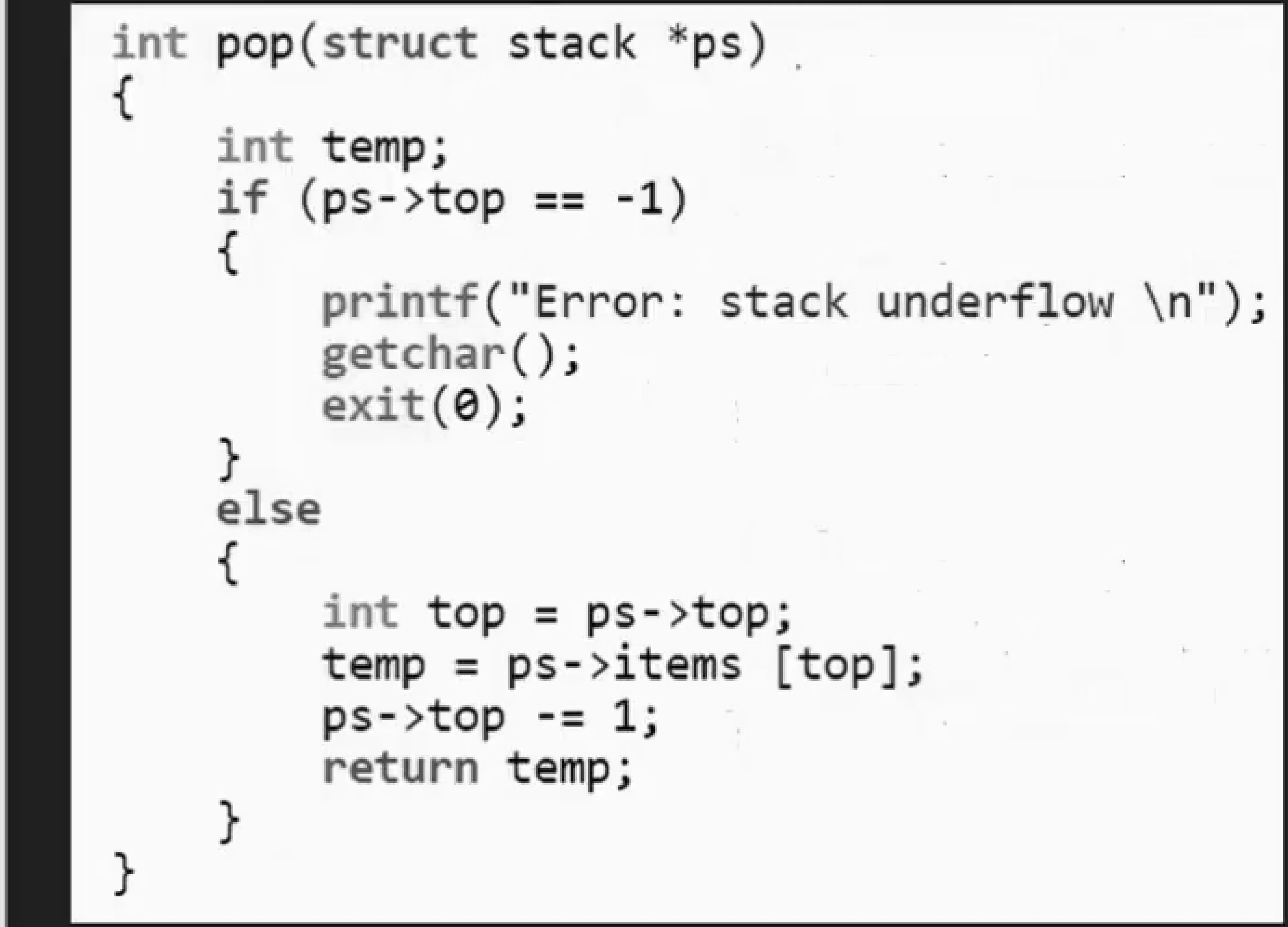
```c
// A Stack based C program to find next greater element
// for all array elements.
#include<stdio.h>
#include<stdlib.h>
#define STACKSIZE 100

// stack structure
struct stack
{
    int top;
    int items[STACKSIZE];
};
```

Method 2 (Using Stack)

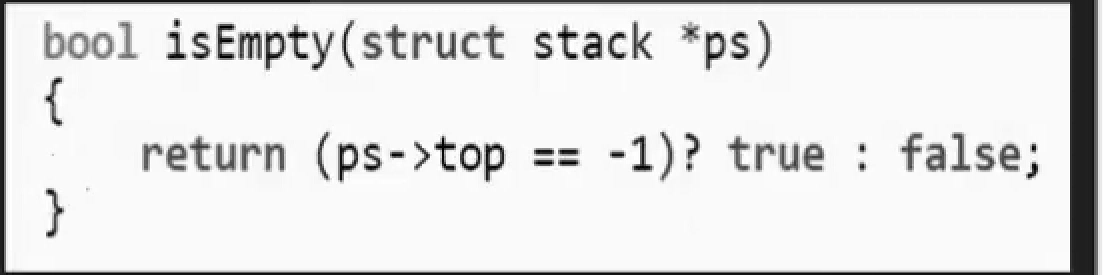


```
// Stack Functions to be used by printNGE()
void push(struct stack *ps, int x)
{
    if (ps->top == STACKSIZE-1)
    {
        printf("Error: stack overflow\n");
        getchar();
        exit(0);
    }
    else
    {
        ps->top += 1;
        int top = ps->top;
        ps->items [top] = x;
    }
}
```

Method 2 (Using Stack)

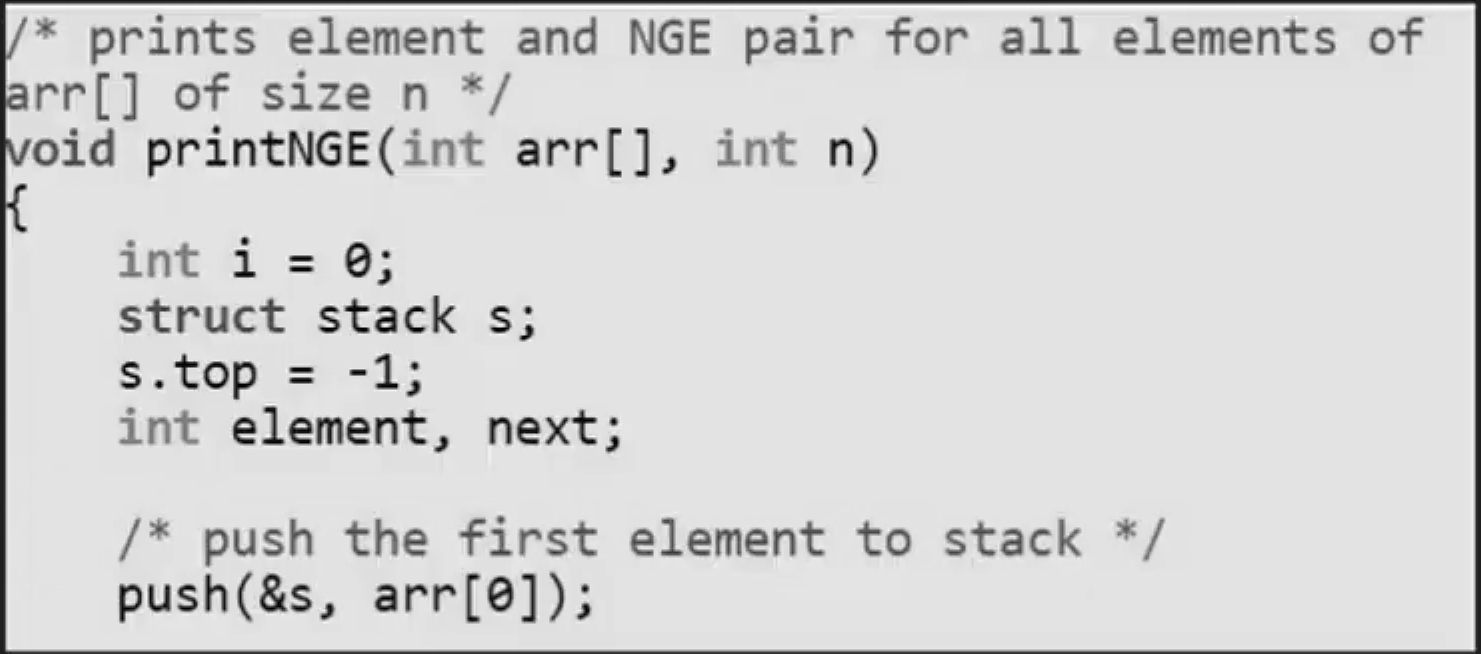


```
int pop(struct stack *ps)
{
    int temp;
    if (ps->top == -1)
    {
        printf("Error: stack underflow \n");
        getchar();
        exit(0);
    }
    else
    {
        int top = ps->top;
        temp = ps->items [top];
        ps->top -= 1;
        return temp;
    }
}
```



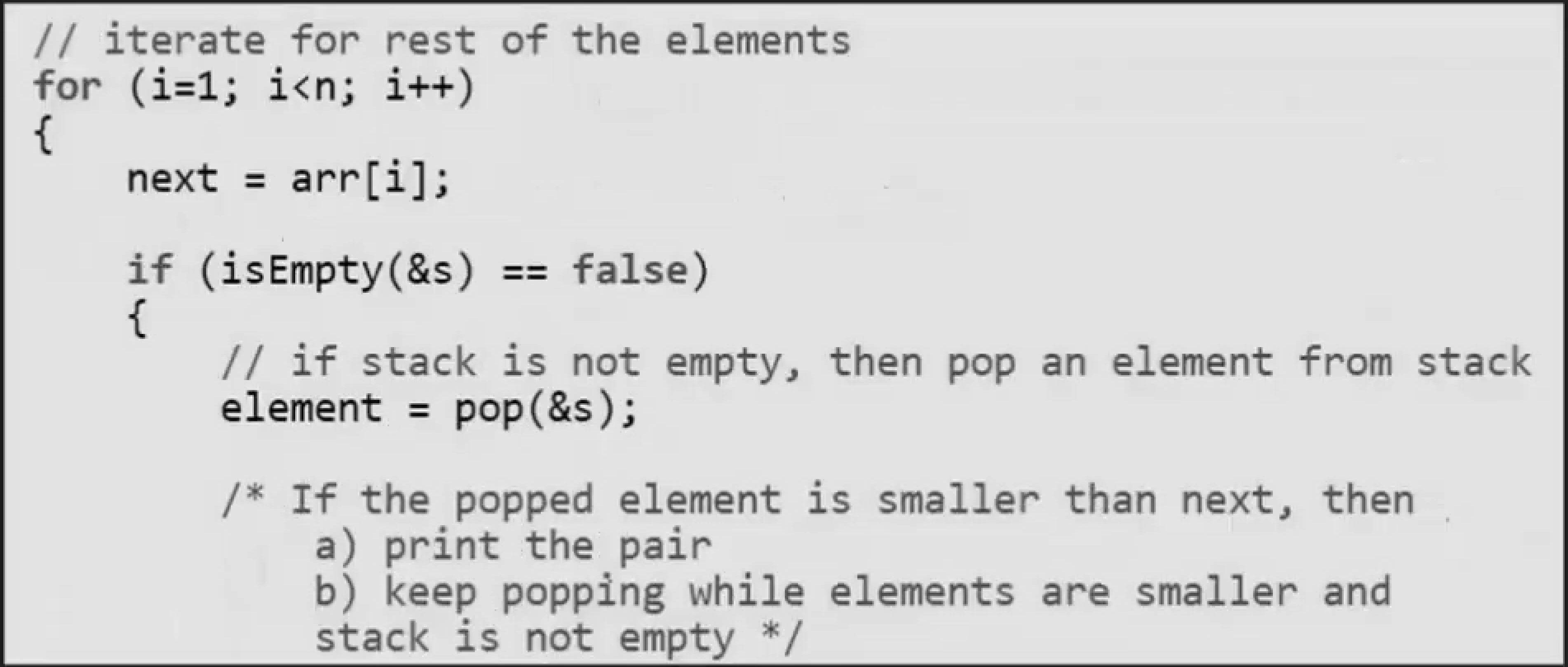
```
bool isEmpty(struct stack *ps)
{
    return (ps->top == -1)? true : false;
}
```

Method 2 (Using Stack)



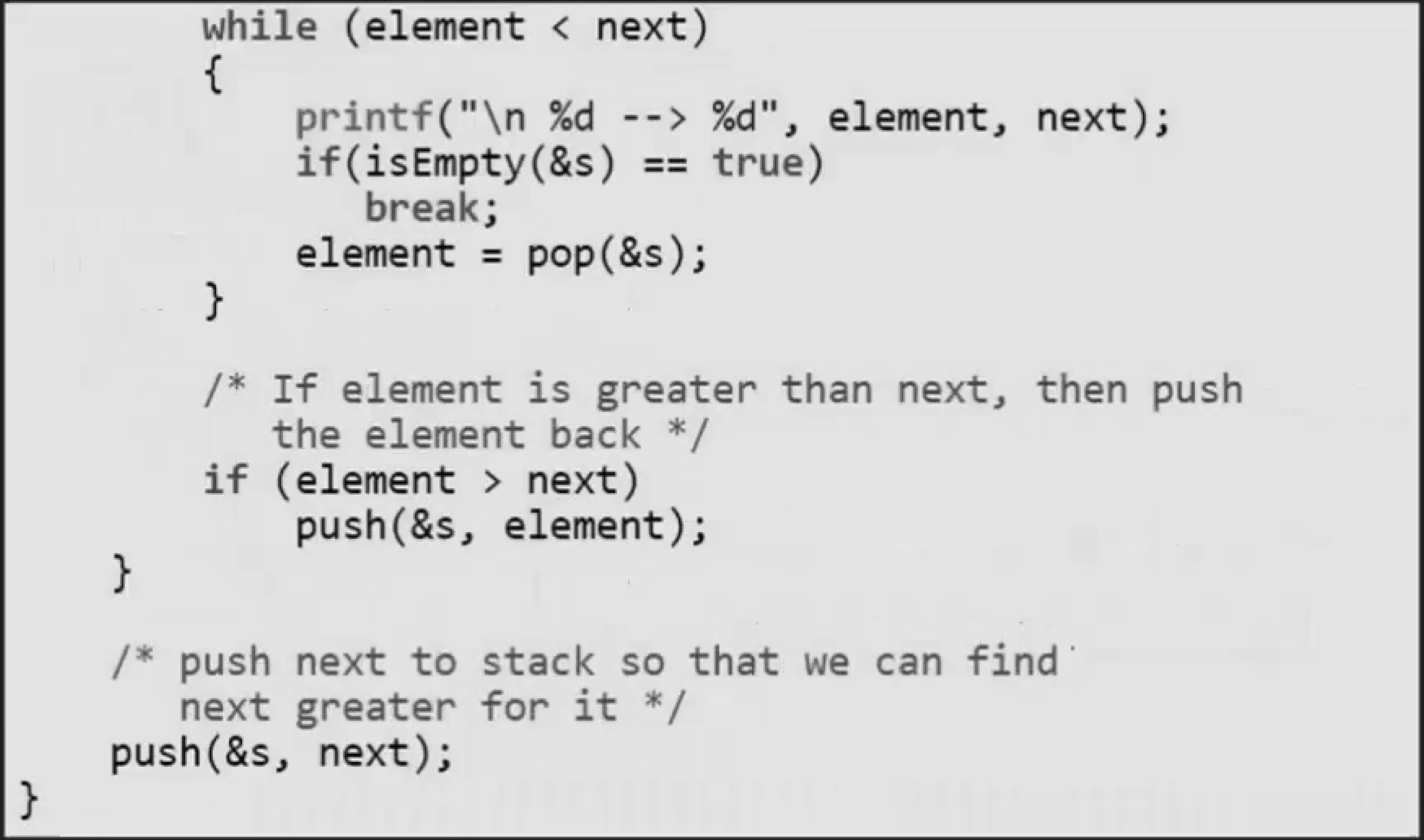
```
/* prints element and NGE pair for all elements of
arr[] of size n */
void printNGE(int arr[], int n)
{
    int i = 0;
    struct stack s;
    s.top = -1;
    int element, next;

    /* push the first element to stack */
    push(&s, arr[0]);
```

Method 2 (Using Stack)



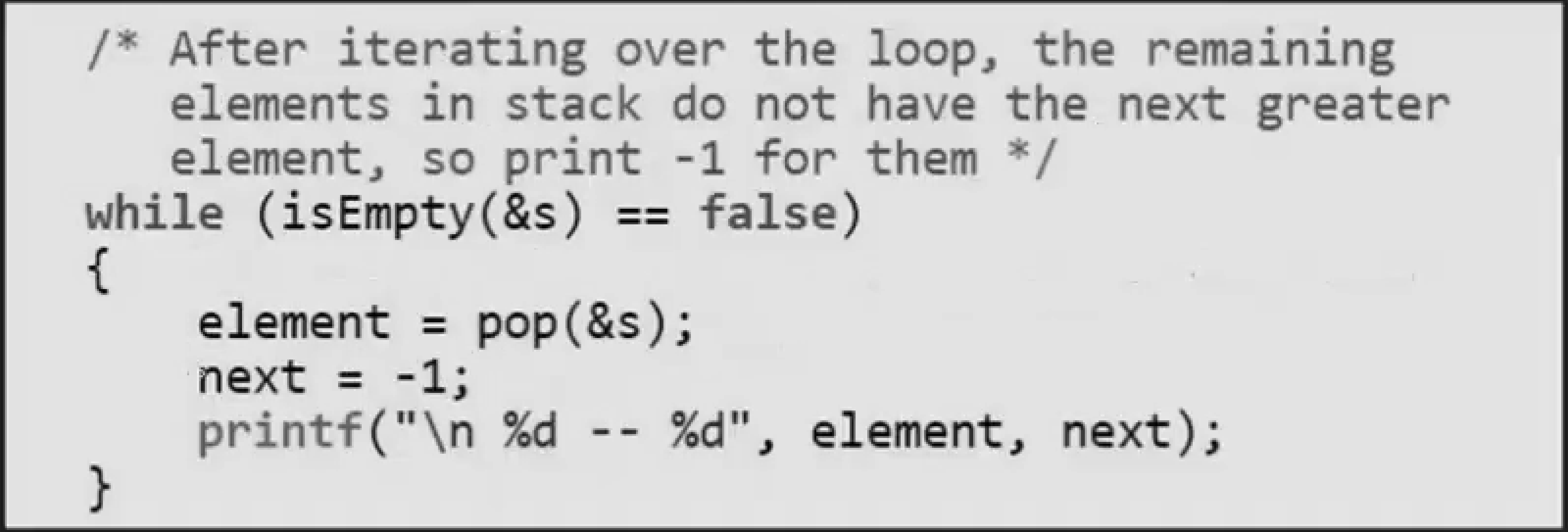
```
// iterate for rest of the elements
for (i=1; i<n; i++)
{
    next = arr[i];

    if (isEmpty(&s) == false)
    {
        // if stack is not empty, then pop an element from stack
        element = pop(&s);

        /* If the popped element is smaller than next, then
            a) print the pair
            b) keep popping while elements are smaller and
            stack is not empty */
```

Method 2 (Using Stack)



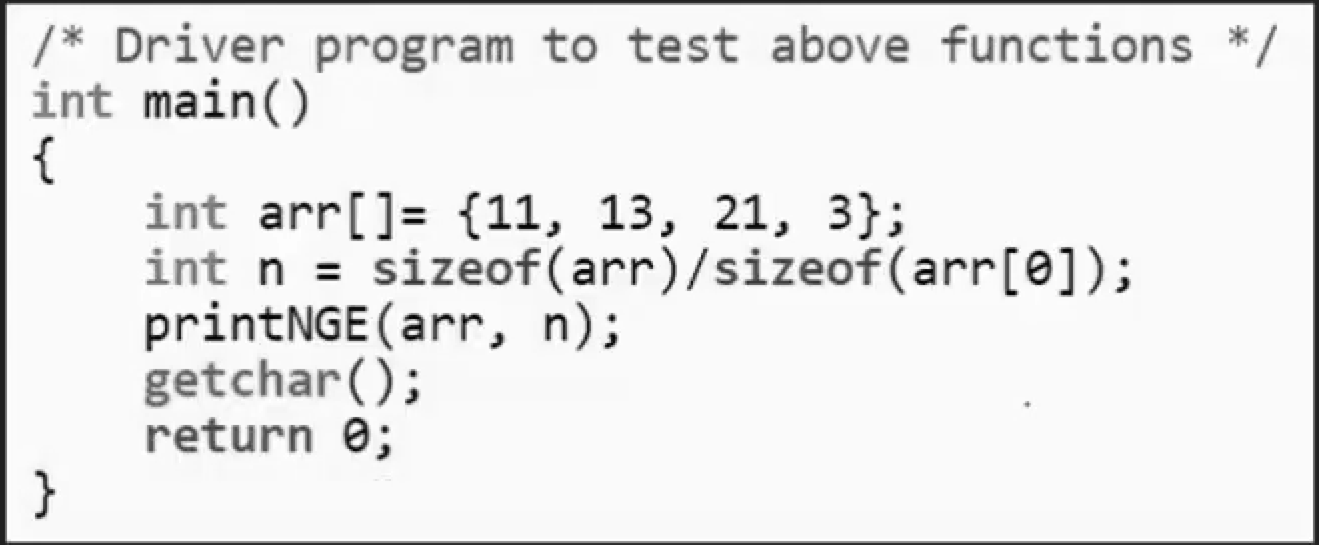
```
        while (element < next)
        {
            printf("\n %d --> %d", element, next);
            if(isEmpty(&s) == true)
               break;
            element = pop(&s);
        }

        /* If element is greater than next, then push
           the element back */
        if (element > next)
            push(&s, element);
    }

    /* push next to stack so that we can find
       next greater for it */
    push(&s, next);
}
```

Method 2 (Using Stack)



```
/* After iterating over the loop, the remaining
   elements in stack do not have the next greater
   element, so print -1 for them */
while (isEmpty(&s) == false)
{
    element = pop(&s);
    next = -1;
    printf("\n %d -- %d", element, next);
}
```

Method 2 (Using Stack)



```
/* Driver program to test above functions */
int main()
{
    int arr[]= {11, 13, 21, 3};
    int n = sizeof(arr)/sizeof(arr[0]);
    printNGE(arr, n);
    getchar();
    return 0;
}
```

Method 2 (Using Stack)

Time Complexity: O(n). The worst case occurs when all elements are sorted in decreasing order. If elements are sorted in decreasing order, then every element is processed at most 4 times.

- Initially pushed to the stack.
- Popped from the stack when next element is being processed.
- Pushed back to the stack because next element is smaller.
- Popped from the stack in step 3 of algo.

Summary

Method	Time Complexity
METHOD 1 (Linear Search)	O(n*n)
Method 2 (Using Stack)	O(n)

Thank you for watching!
Please leave us your comments.