CS111，C Programming Lab / Pointer

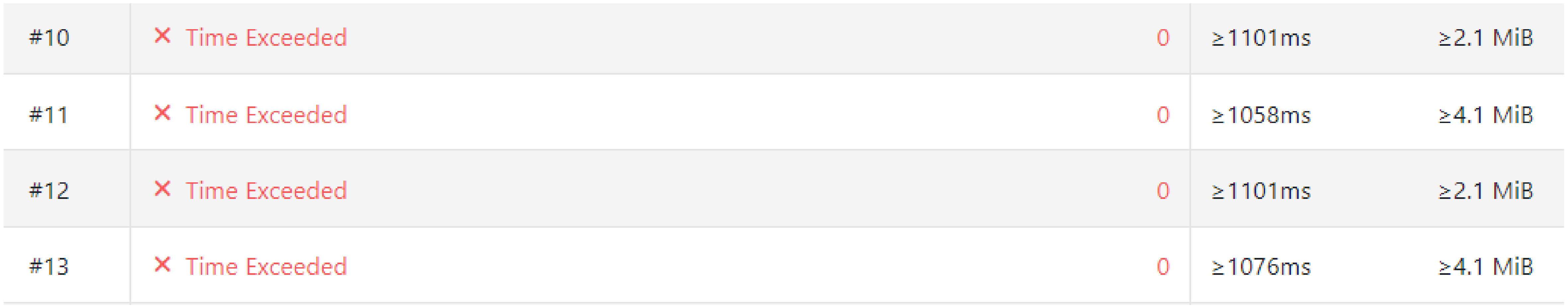
黄嘉炜

huangjw3@mail.sustech.edu.cn

Outline

- Review
- Pointer: dynamic allocation/deallocation
- Pointer: safe issue, const, ...
- Assignment

Review: Remove char / Time Exceeded

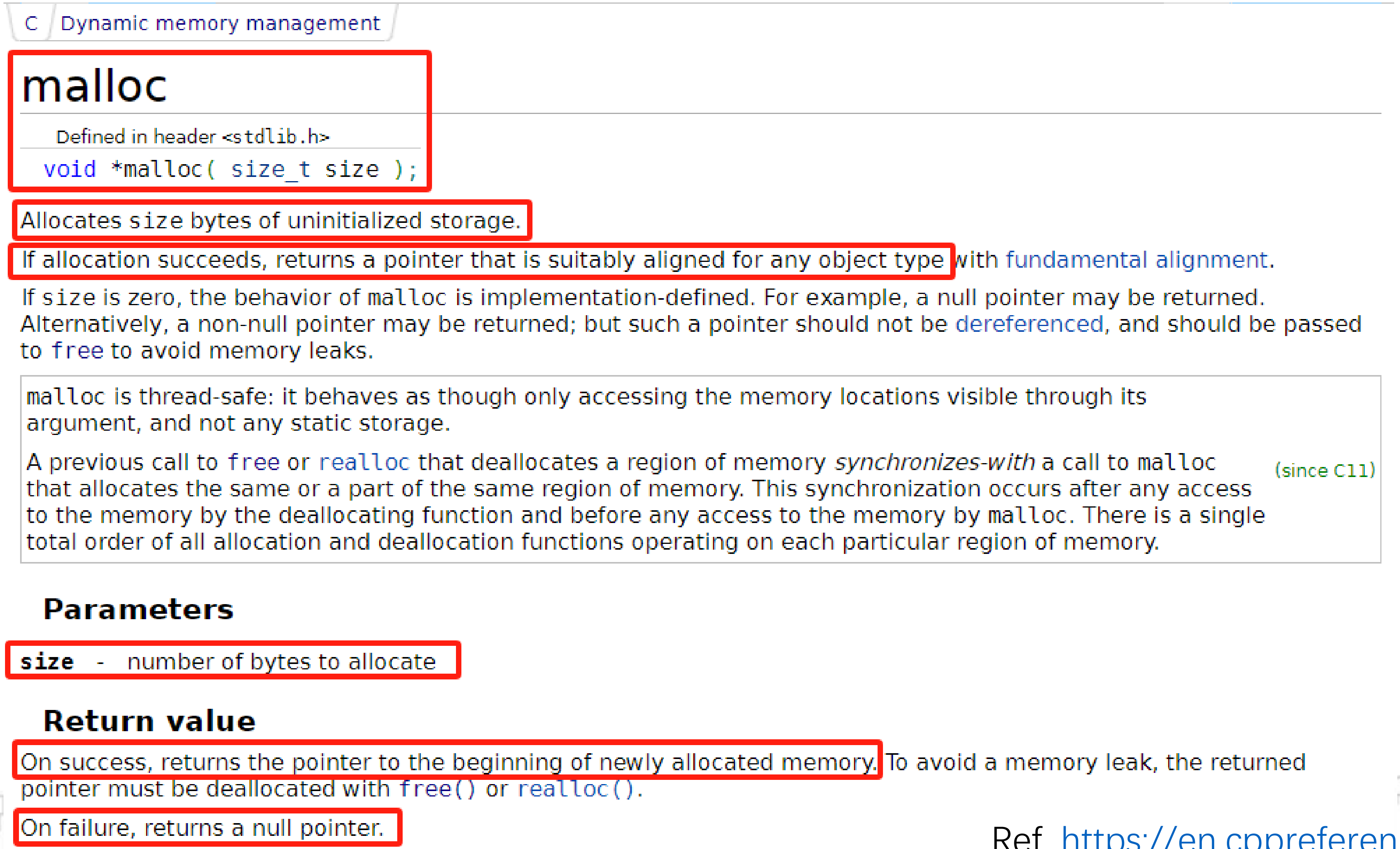


#10	× Time Exceeded	0	≥1101ms	≥2.1 MiB
#11	× Time Exceeded	0	≥1058ms	≥4.1 MiB
#12	× Time Exceeded	0	≥1101ms	≥2.1 MiB
#13	× Time Exceeded	0	≥1076ms	≥4.1 MiB

Outline

- Review
- **Pointer: dynamic allocation/deallocation**
- Pointer: safe issue, const, …
- Assignment

Pointer: dynamic allocation/deallocation



C / Dynamic memory management

malloc

Defined in header `<stdlib.h>`

```
void *malloc( size_t size );
```

Allocates `size` bytes of uninitialized storage.

If allocation succeeds, returns a pointer that is suitably aligned for any object type with fundamental alignment.

If `size` is zero, the behavior of `malloc` is implementation-defined. For example, a null pointer may be returned. Alternatively, a non-null pointer may be returned; but such a pointer should not be dereferenced, and should be passed to `free` to avoid memory leaks.

`malloc` is thread-safe: it behaves as though only accessing the memory locations visible through its argument, and not any static storage.

A previous call to `free` or `realloc` that deallocates a region of memory *synchronizes-with* a call to `malloc` that allocates the same or a part of the same region of memory. This synchronization occurs after any access to the memory by the deallocating function and before any access to the memory by `malloc`. There is a single total order of all allocation and deallocation functions operating on each particular region of memory. (since C11)

Parameters

size - number of bytes to allocate

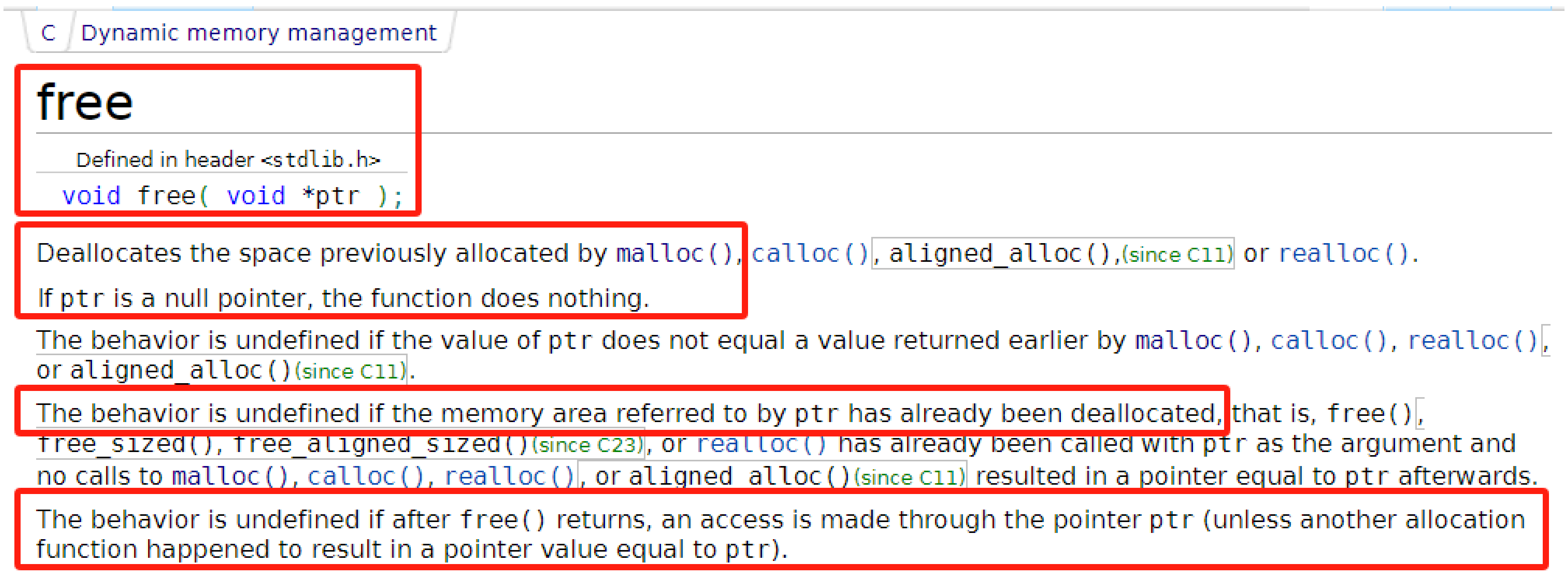
Return value

On success, returns the pointer to the beginning of newly allocated memory. To avoid a memory leak, the returned pointer must be deallocated with `free()` or `realloc()`.

On failure, returns a null pointer.

Ref, https://en.cppreference.com/w/c/memory/malloc

Pointer: dynamic allocation/deallocation



C / Dynamic memory management

free

Defined in header `<stdlib.h>`

```
void free( void *ptr );
```

Deallocates the space previously allocated by `malloc()`, `calloc()`, `aligned_alloc()`,(since C11) or `realloc()`.

If `ptr` is a null pointer, the function does nothing.

The behavior is undefined if the value of `ptr` does not equal a value returned earlier by `malloc()`, `calloc()`, `realloc()`, or `aligned_alloc()`(since C11).

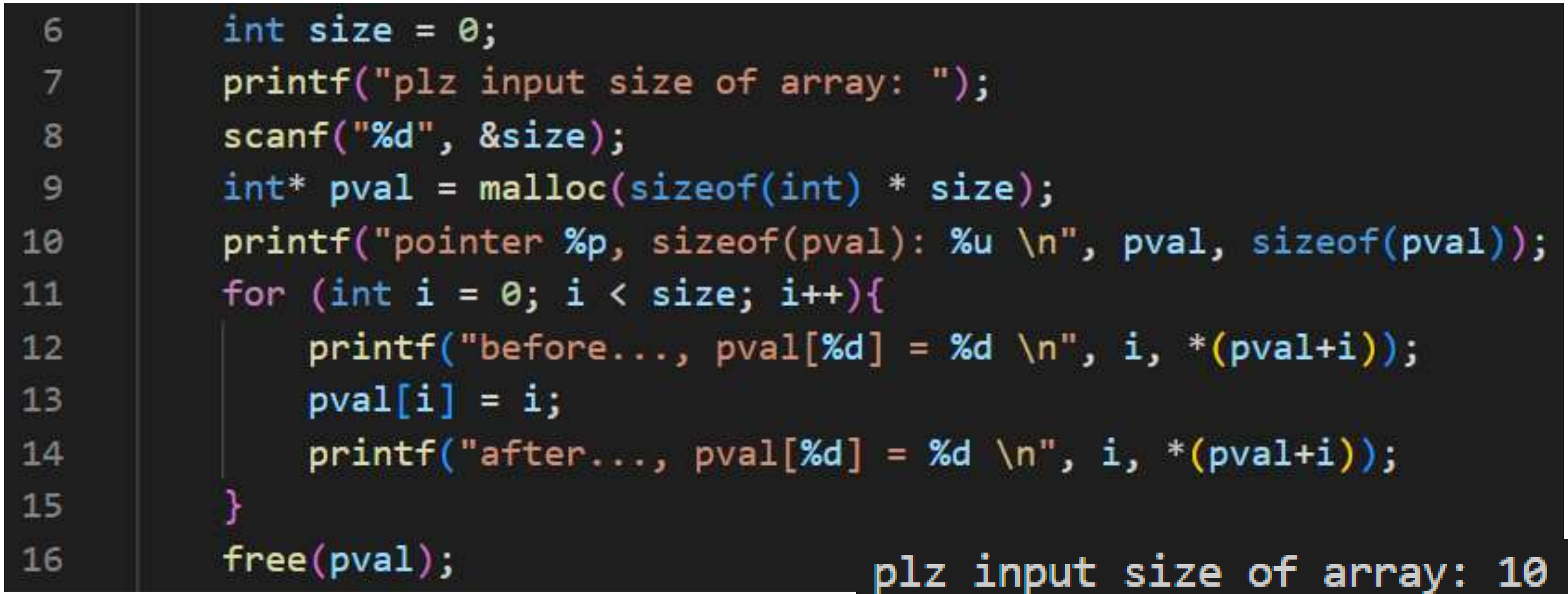
The behavior is undefined if the memory area referred to by `ptr` has already been deallocated, that is, `free()`, `free_sized()`, `free_aligned_sized()`(since C23), or `realloc()` has already been called with `ptr` as the argument and no calls to `malloc()`, `calloc()`, `realloc()`, or `aligned_alloc()`(since C11) resulted in a pointer equal to `ptr` afterwards.

The behavior is undefined if after `free()` returns, an access is made through the pointer `ptr` (unless another allocation function happened to result in a pointer value equal to `ptr`).

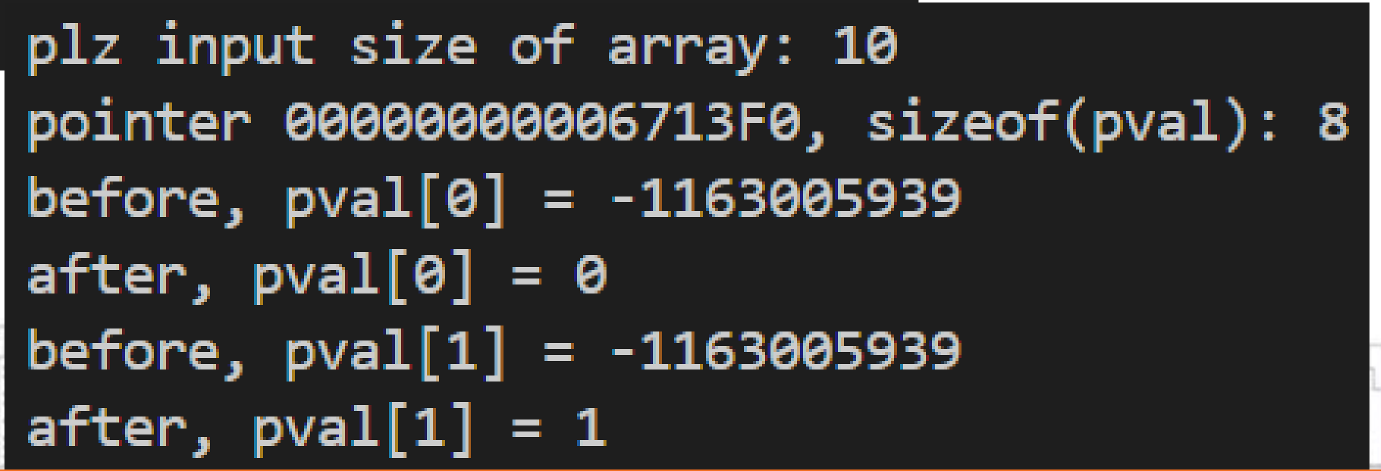
TODO

Ref, https://en.cppreference.com/w/c/memory/free

Pointer: dynamic allocation/deallocation



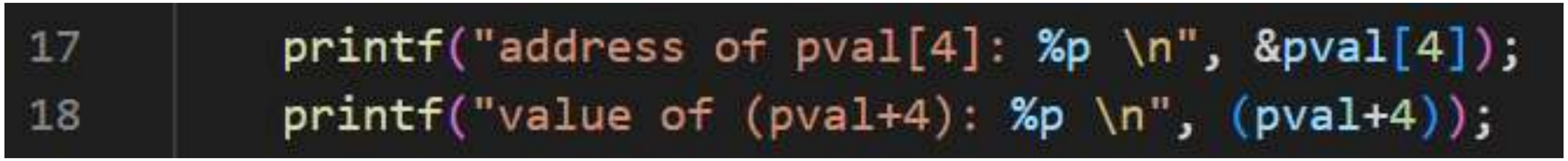
```c
    int size = 0;
    printf("plz input size of array: ");
    scanf("%d", &size);
    int* pval = malloc(sizeof(int) * size);
    printf("pointer %p, sizeof(pval): %u \n", pval, sizeof(pval));
    for (int i = 0; i < size; i++){
        printf("before..., pval[%d] = %d \n", i, *(pval+i));
        pval[i] = i;
        printf("after..., pval[%d] = %d \n", i, *(pval+i));
    }
    free(pval);
```



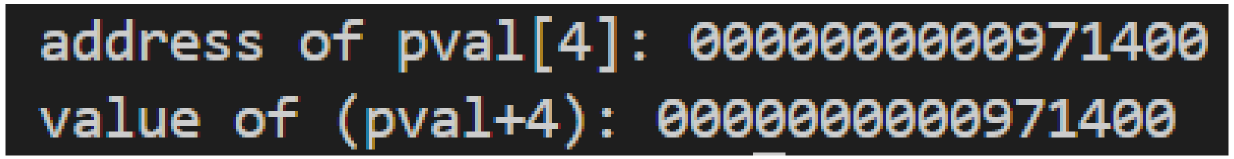
```
plz input size of array: 10
pointer 00000000006713F0, sizeof(pval): 8
before, pval[0] = -1163005939
after, pval[0] = 0
before, pval[1] = -1163005939
after, pval[1] = 1
```

Highlight
- 出发点：按需分配内存空间 (vs 数组)
- malloc 的内存空间未初始化
- malloc 后必须 free

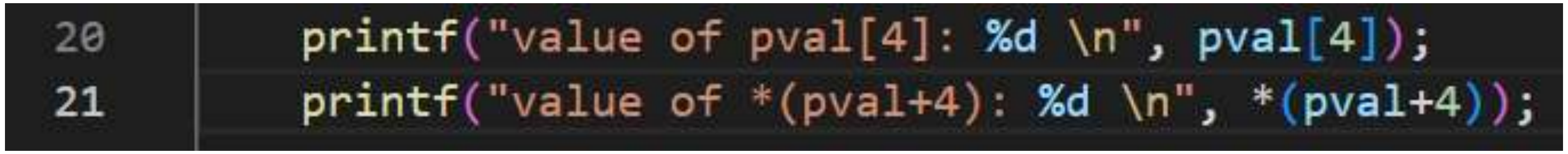
Pointer: dynamic allocation/deallocation



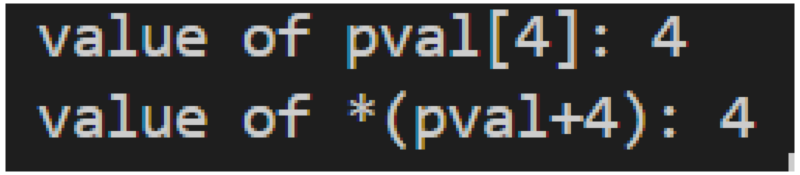
```
    printf("address of pval[4]: %p \n", &pval[4]);
    printf("value of (pval+4): %p \n", (pval+4));
```



```
address of pval[4]: 0000000000971400
value of (pval+4): 0000000000971400
```



```
    printf("value of pval[4]: %d \n", pval[4]);
    printf("value of *(pval+4): %d \n", *(pval+4));
```



```
value of pval[4]: 4
value of *(pval+4): 4
```

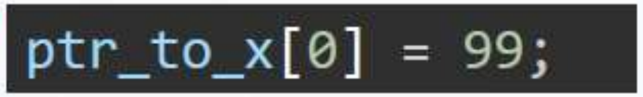
Highlight，2种读取数据的方法

pointer_to_array[offset]

***(pointer_to_array + offset)**

NOTE, 当ptr_to_x 只指向单个变量 x (非数组 / 连续内存块), ...

Unrecommended!



```
ptr_to_x[0] = 99;
```

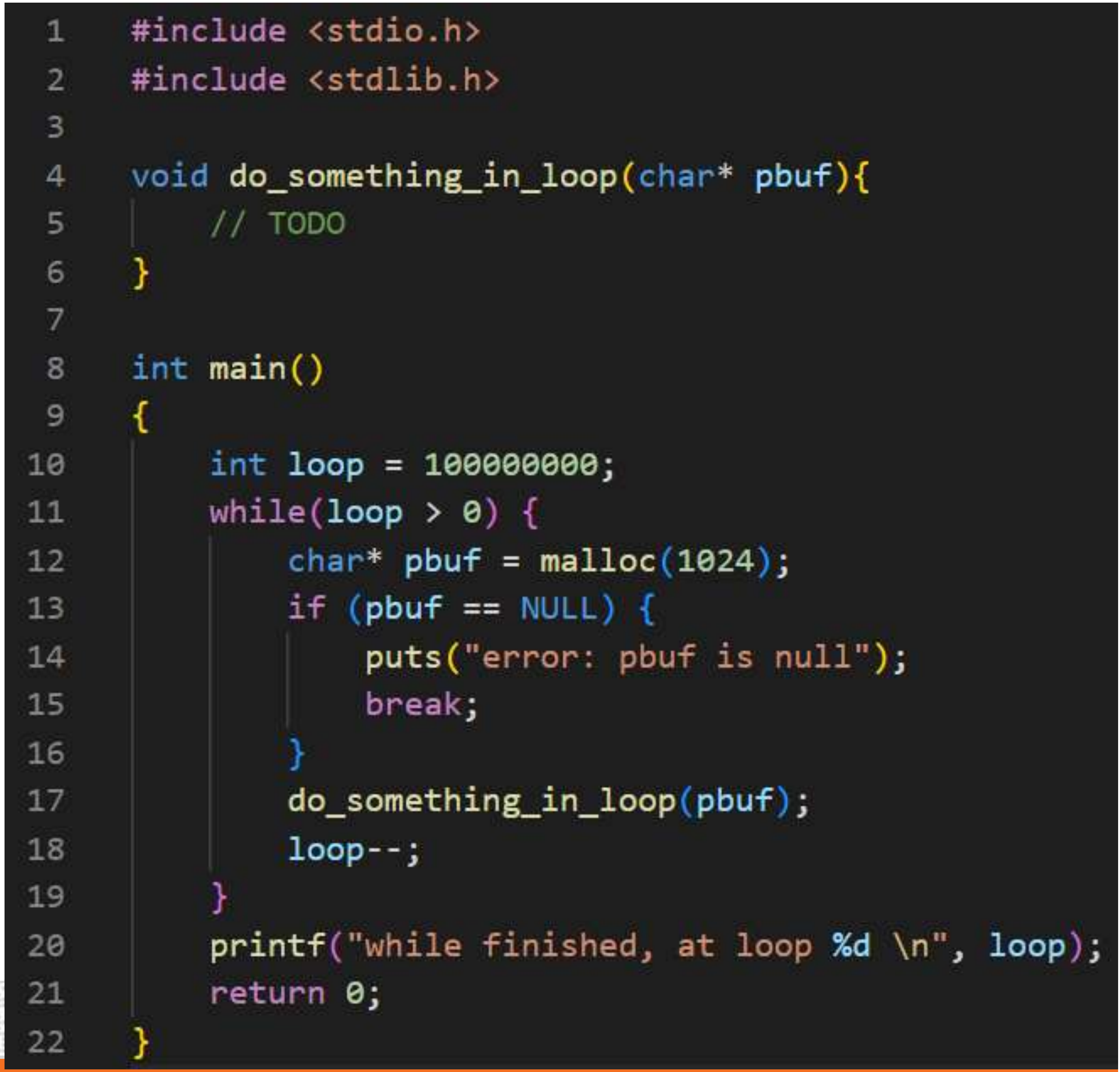
Outline

- Review
- Pointer: dynamic allocation/deallocation
- **Pointer: safe issue, const, …**
- Assignment

Pointer: safe issue

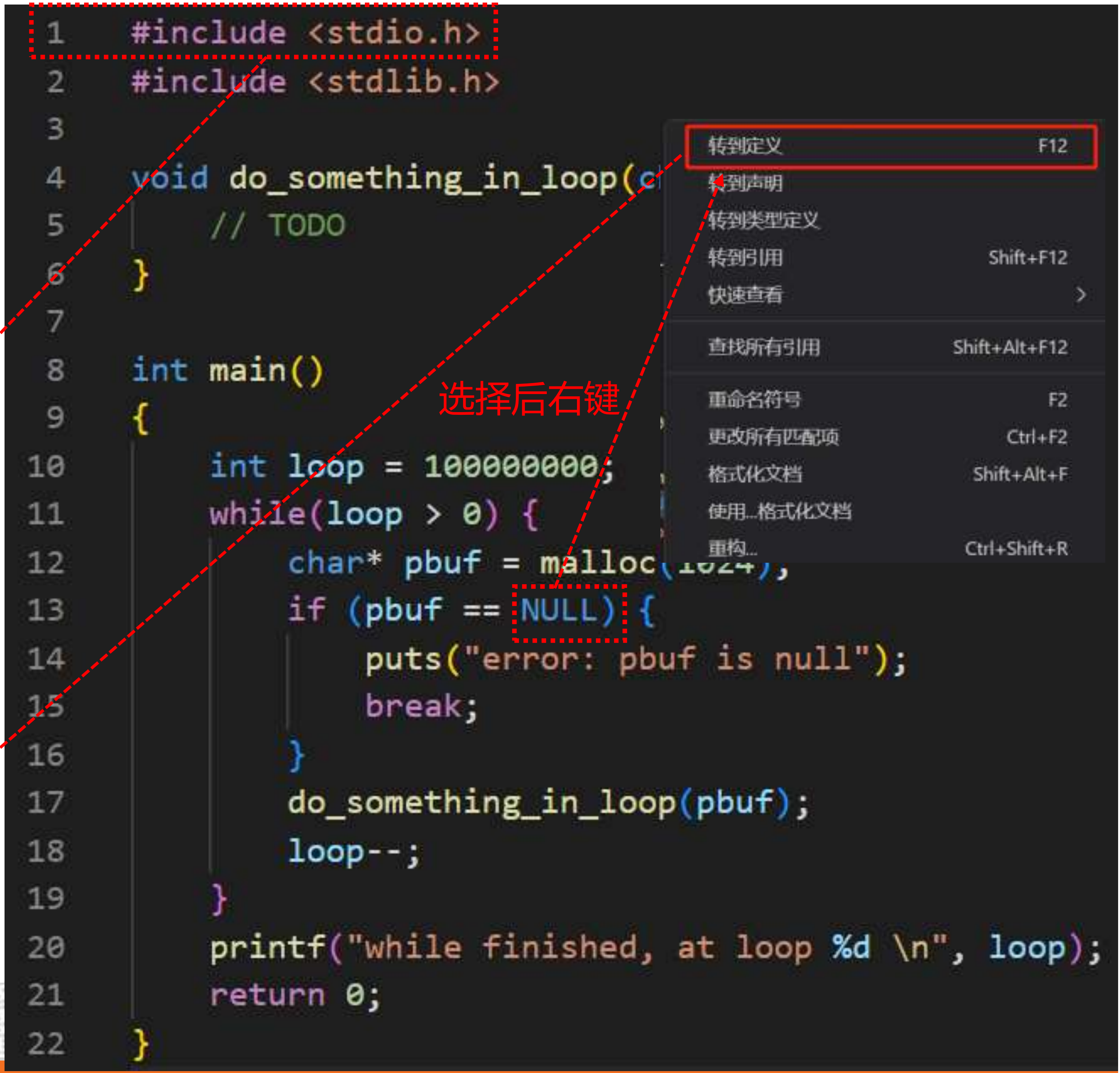
问题：??

结果：??

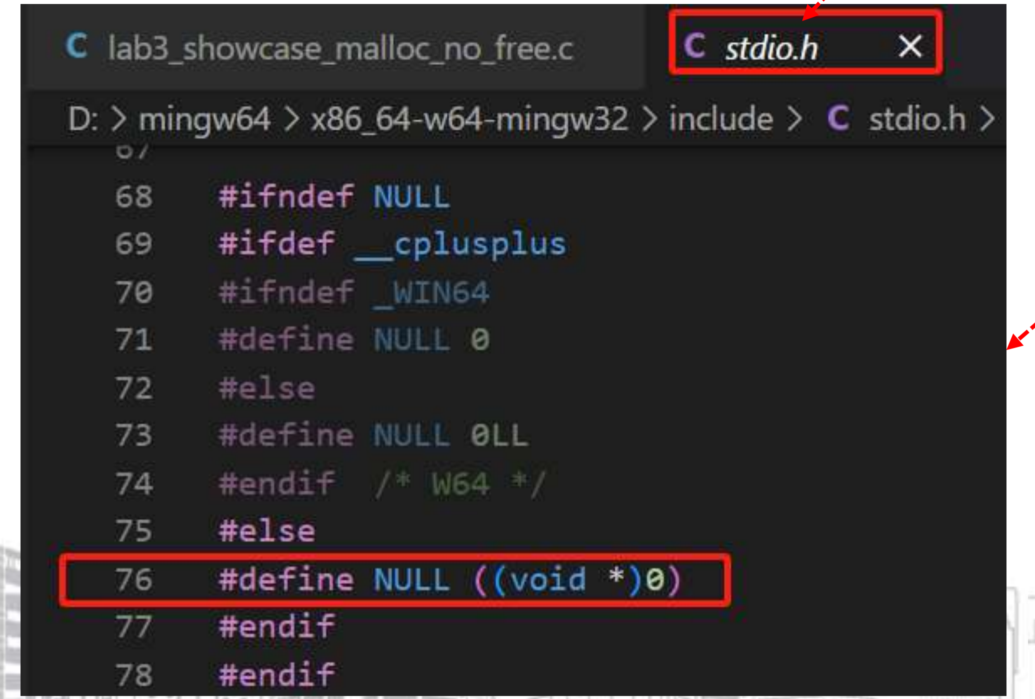


```c
#include <stdio.h>
#include <stdlib.h>

void do_something_in_loop(char* pbuf){
    // TODO
}

int main()
{
    int loop = 100000000;
    while(loop > 0) {
        char* pbuf = malloc(1024);
        if (pbuf == NULL) {
            puts("error: pbuf is null");
            break;
        }
        do_something_in_loop(pbuf);
        loop--;
    }
    printf("while finished, at loop %d \n", loop);
    return 0;
}
```

Appendix, NULL ?



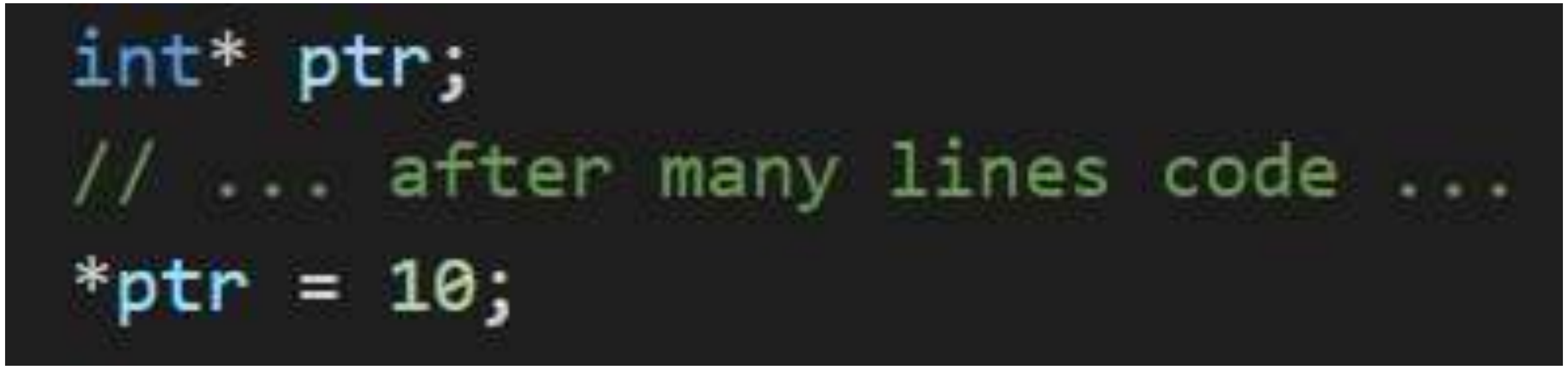
```c
#include <stdio.h>
#include <stdlib.h>

void do_something_in_loop(c
    // TODO
}

int main()
{
    int loop = 100000000;
    while(loop > 0) {
        char* pbuf = malloc(1024);
        if (pbuf == NULL) {
            puts("error: pbuf is null");
            break;
        }
        do_something_in_loop(pbuf);
        loop--;
    }
    printf("while finished, at loop %d \n", loop);
    return 0;
}
```



lab3_showcase_malloc_no_free.c
stdio.h
D: > mingw64 > x86_64-w64-mingw32 > include > stdio.h >

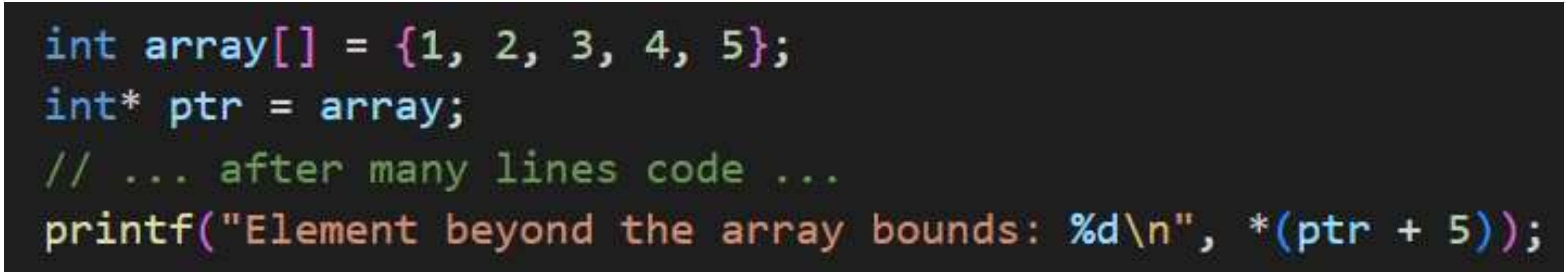
```c
#ifndef NULL
#ifdef __cplusplus
#ifndef _WIN64
#define NULL 0
#else
#define NULL 0LL
#endif  /* W64 */
#else
#define NULL ((void *)0)
#endif
#endif
```

Pointer: safe issue / 野指针



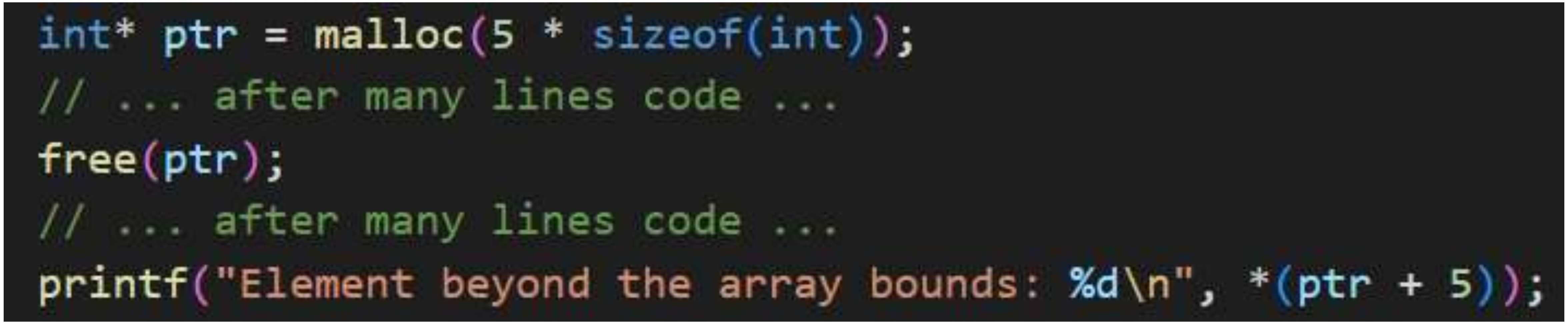
```
int* ptr;
// ... after many lines code ...
*ptr = 10;
```

Pointer: safe issue / 指针越界



```c
int array[] = {1, 2, 3, 4, 5};
int* ptr = array;
// ... after many lines code ...
printf("Element beyond the array bounds: %d\n", *(ptr + 5));
```

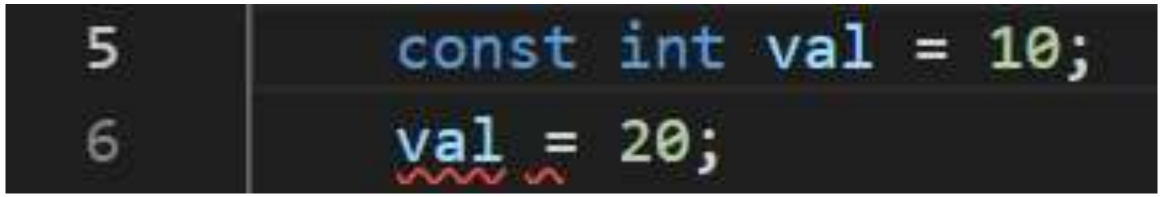
Pointer: safe issue



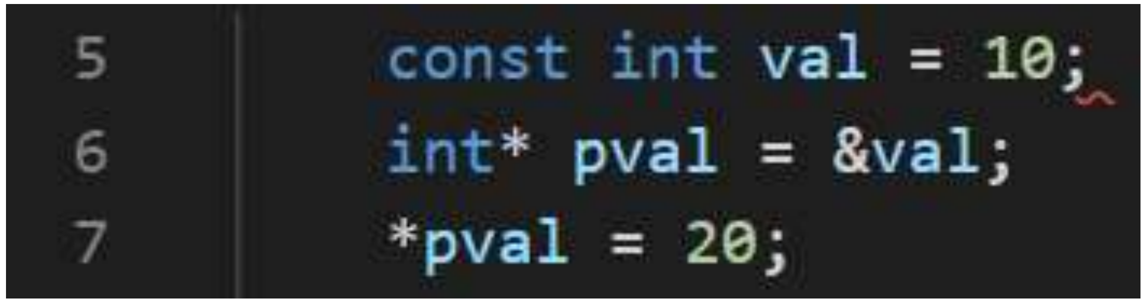
```
int* ptr = malloc(5 * sizeof(int));
// ... after many lines code ...
free(ptr);
// ... after many lines code ...
printf("Element beyond the array bounds: %d\n", *(ptr + 5));
```

How about this ?

Pointer: const



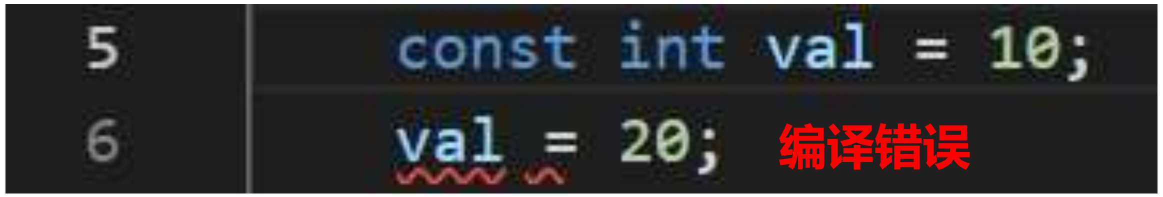
```
    const int val = 10;
val = 20;
```



```
    const int val = 10;
int* pval = &val;
*pval = 20;
```

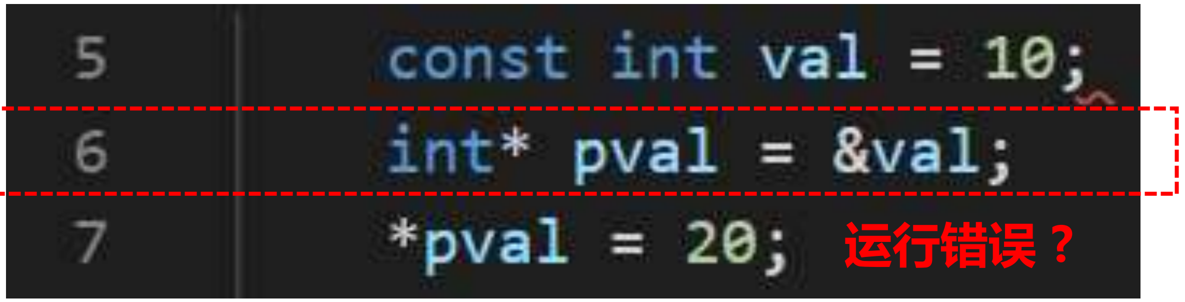
Bug ??

Pointer: const



```
const int val = 10;
val = 20;
```

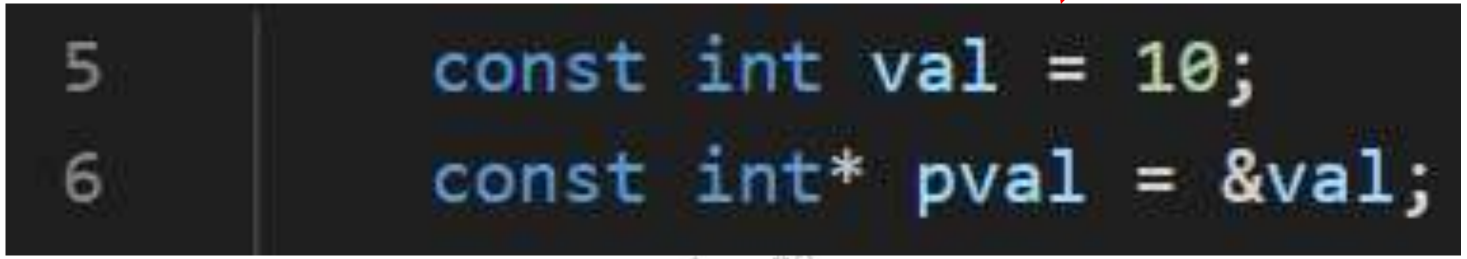
编译错误



```
const int val = 10;
int* pval = &val;
*pval = 20;
```

运行错误？

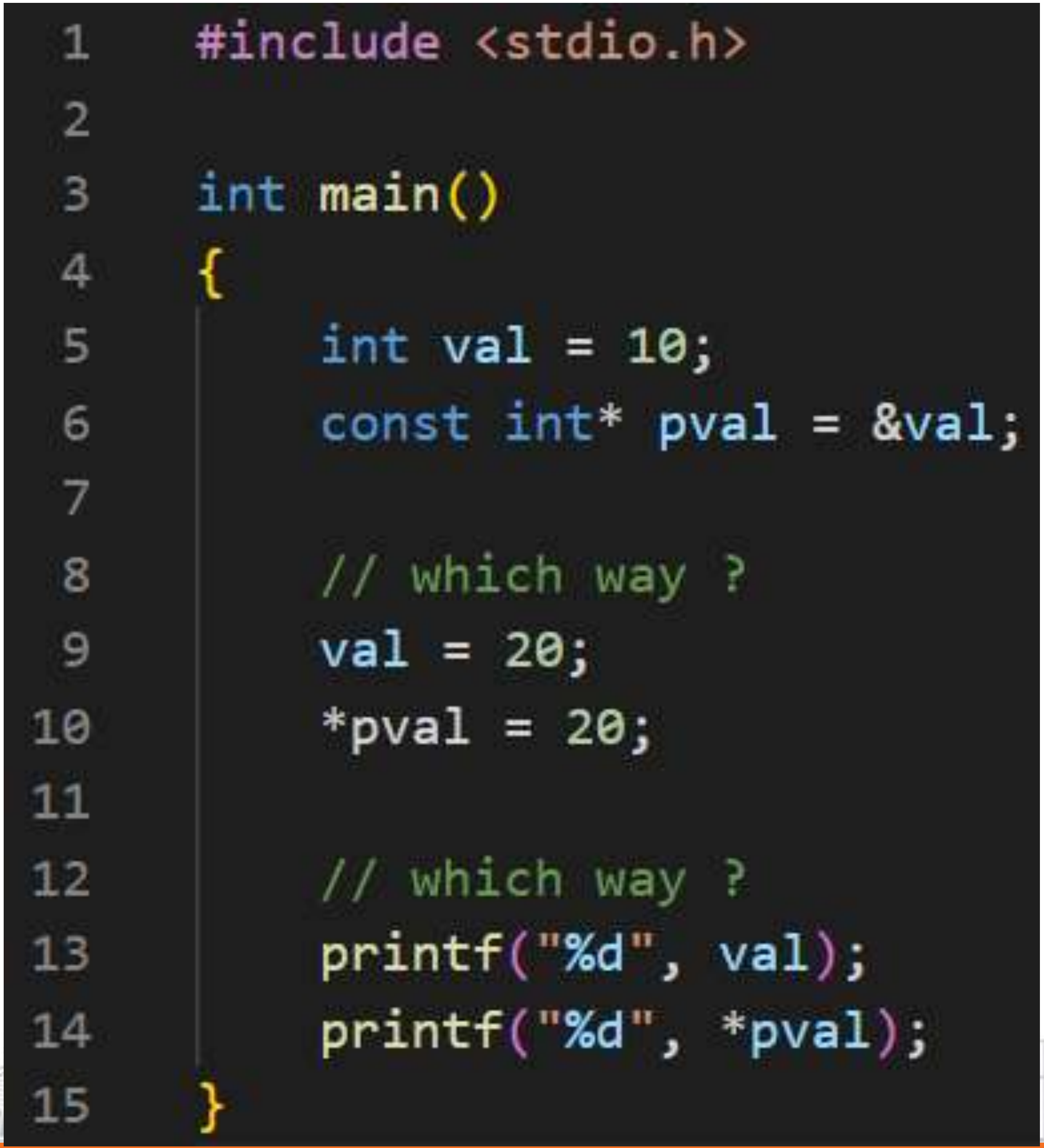
指向 常量 的指针，设置为常数指针，避免修改



```
const int val = 10;
const int* pval = &val;
```

Pointer: const

Which ?



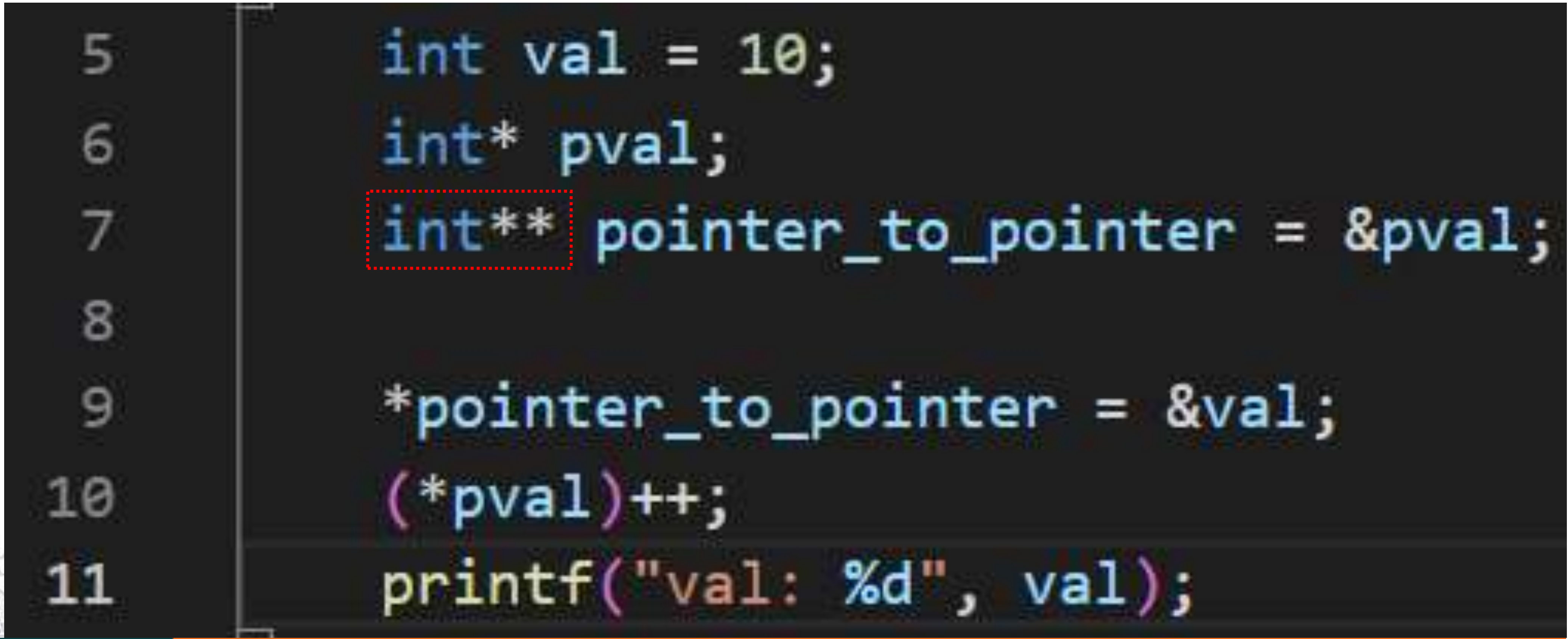
```c
#include <stdio.h>

int main()
{
    int val = 10;
    const int* pval = &val;

    // which way ?
    val = 20;
    *pval = 20;

    // which way ?
    printf("%d", val);
    printf("%d", *pval);
}
```

Pointer: pointer to pointer

如何理解：int, 等同于 (int*)***

- **把 int* 理解为一个特殊变量**
- **int** 则是指向 int* 变量的指针**

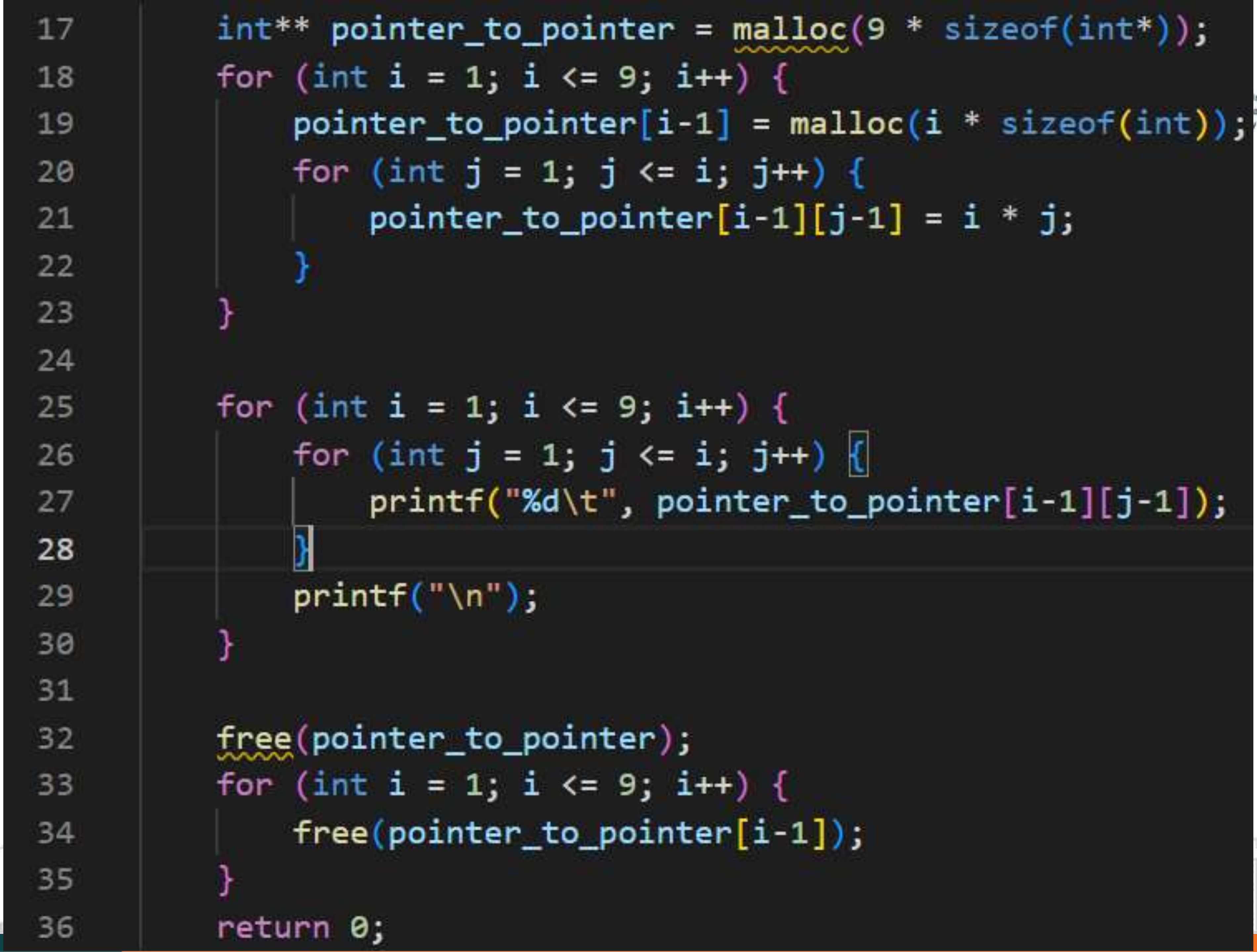
output ?



```
int val = 10;
int* pval;
int** pointer_to_pointer = &pval;

*pointer_to_pointer = &val;
(*pval)++;
printf("val: %d", val);
```

Pointer:

- What this do?
- Bug ??



```c
    int** pointer_to_pointer = malloc(9 * sizeof(int*));
    for (int i = 1; i <= 9; i++) {
        pointer_to_pointer[i-1] = malloc(i * sizeof(int));
        for (int j = 1; j <= i; j++) {
            pointer_to_pointer[i-1][j-1] = i * j;
        }
    }

    for (int i = 1; i <= 9; i++) {
        for (int j = 1; j <= i; j++) {
            printf("%d\t", pointer_to_pointer[i-1][j-1]);
        }
        printf("\n");
    }

    free(pointer_to_pointer);
    for (int i = 1; i <= 9; i++) {
        free(pointer_to_pointer[i-1]);
    }
    return 0;
```

Outline

- Review
- Pointer: dynamic allocation/deallocation
- Pointer: safe issue, const, …
- **Assignment**

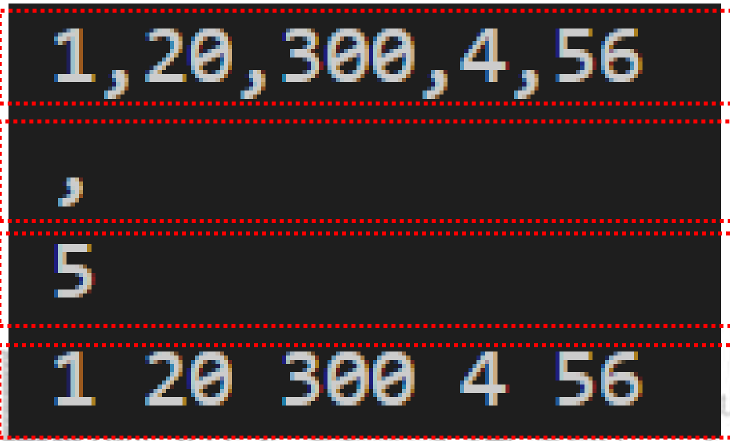
Assignment 1) 字符串转换为整数数组

write a function that: extract an integer array from a string.
The input string only contains numbers and delimiter character. Such as “1|20|300|4|56”, that is series of integers separated by ‘|’ (as delimiter here). And we need to extract integer array, such as [1, 20, 300, 4, 56].

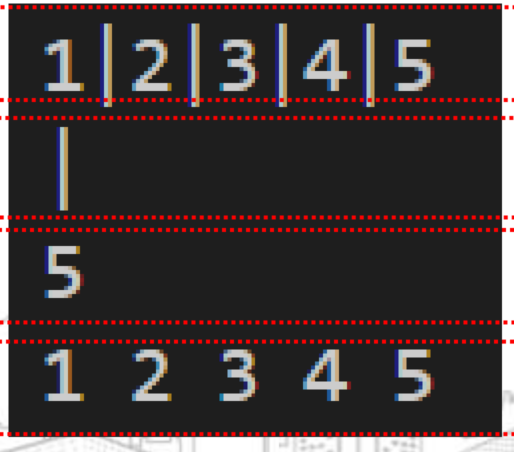
Hint:

- Function define: int* extract_integers_from_string(const char* str, const char delimiter, int* count)
  where: count is a pointer to an integer to store the length of output array
- The return value is integer array, which need to dynamically allocate memory to store the results.

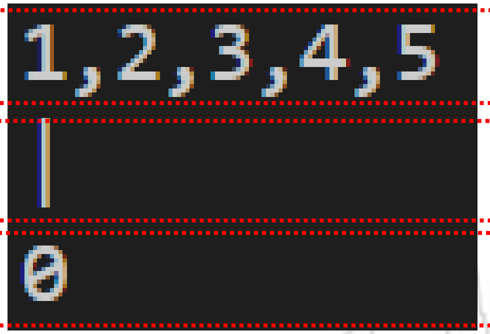
NOTE: Code template is given in next page!



```
1,20,300,4,56
,
5
1 20 300 4 56
```



```
1|2|3|4|5
|
5
1 2 3 4 5
```



```
1,2,3,4,5
|
0
```

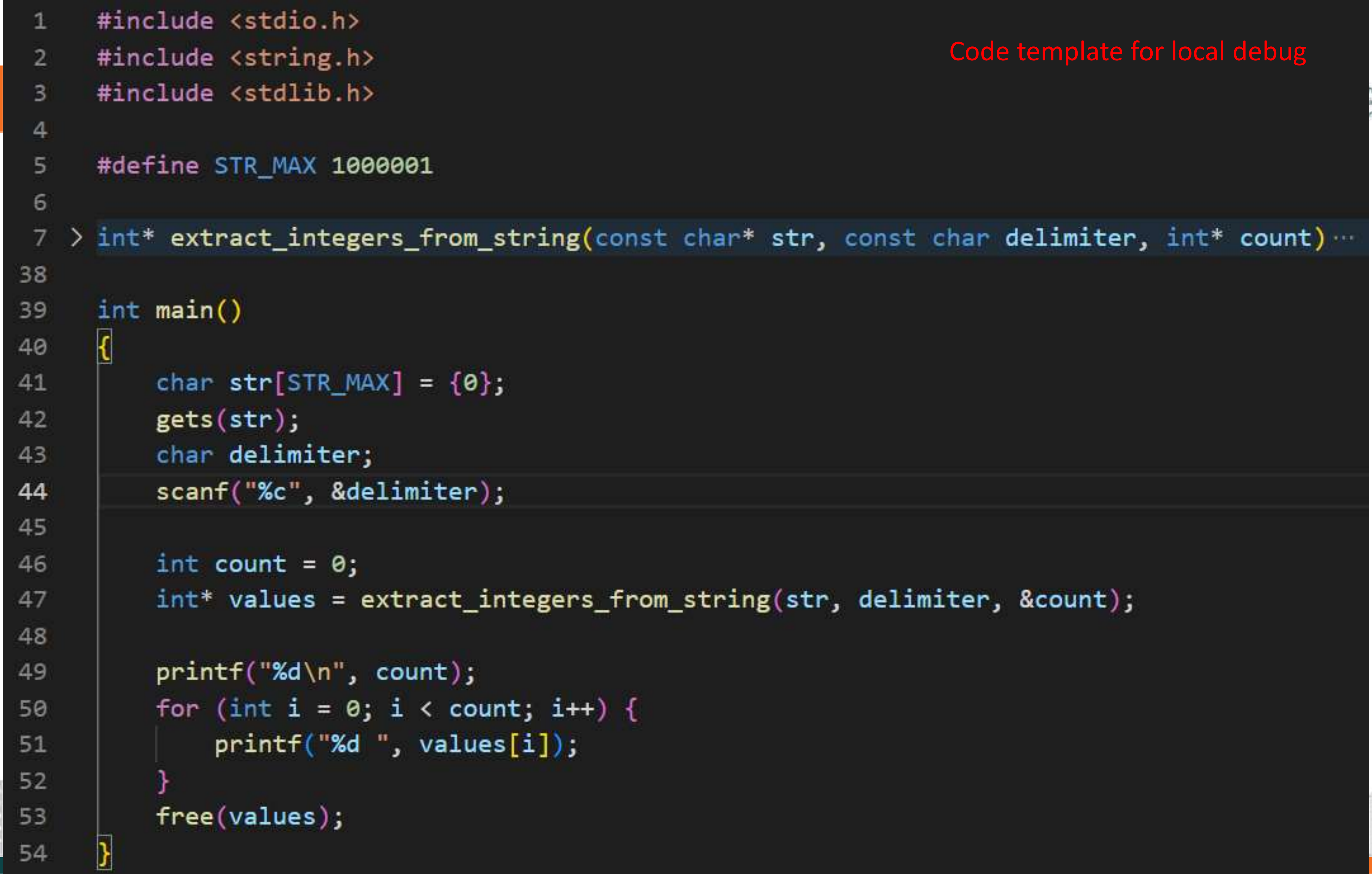
Input: str

Input: delimiter

Output: length of array

Output: integer array

Code template for local debug



```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define STR_MAX 1000001

int* extract_integers_from_string(const char* str, const char delimiter, int* count)⋯

int main()
{
    char str[STR_MAX] = {0};
    gets(str);
    char delimiter;
    scanf("%c", &delimiter);

    int count = 0;
    int* values = extract_integers_from_string(str, delimiter, &count);

    printf("%d\n", count);
    for (int i = 0; i < count; i++) {
        printf("%d ", values[i]);
    }
    free(values);
}
```

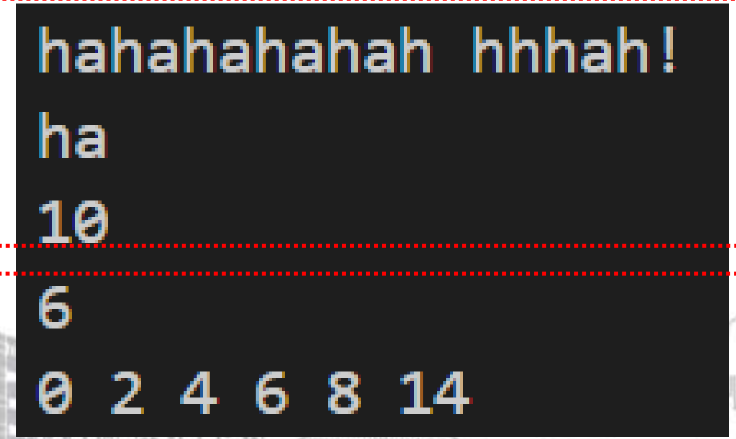
Assignment 2) 搜索关键词（不区分大小写）

write a function that: search keyword case insensitive, and return top-N occurrences.

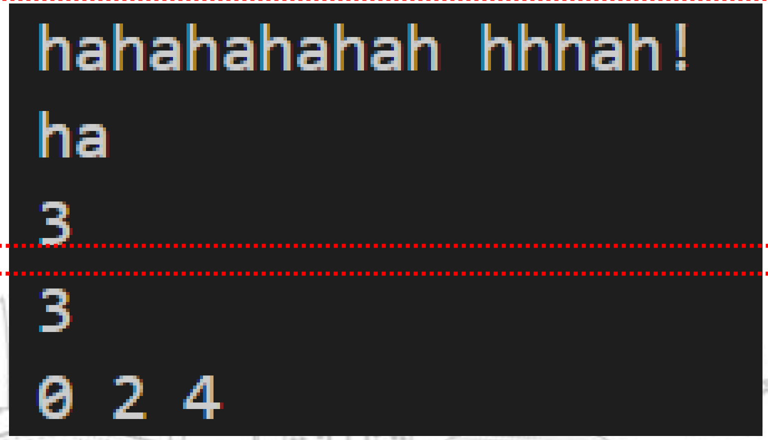
Hint:

- Function define: int search_keyword_case_insensitive(const char* str, const char* keyword, int top_n, int* positions)
  where: positions is a pointer to an integer array to store the starting positions of {top_n} occurrences
- The return value (int) is the length of valid value in {positions} array after searching. And the return value <= {top_n}.

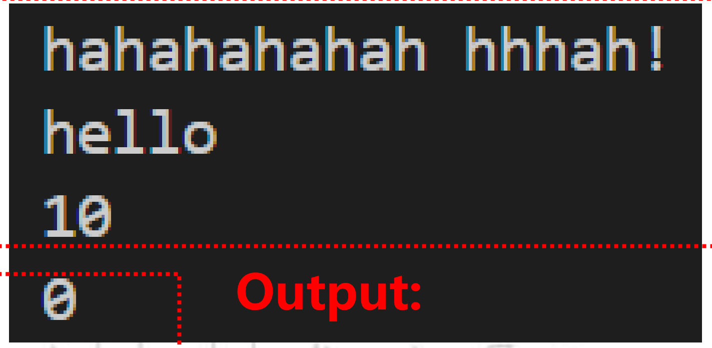
NOTE: Code template is given in next page!



```
hahahahahah hhhah!
ha
10
6
0 2 4 6 8 14
```



```
hahahahahah hhhah!
ha
3
3
0 2 4
```



```
hahahahahah hhhah!
hello
10
0
```

Input:

str

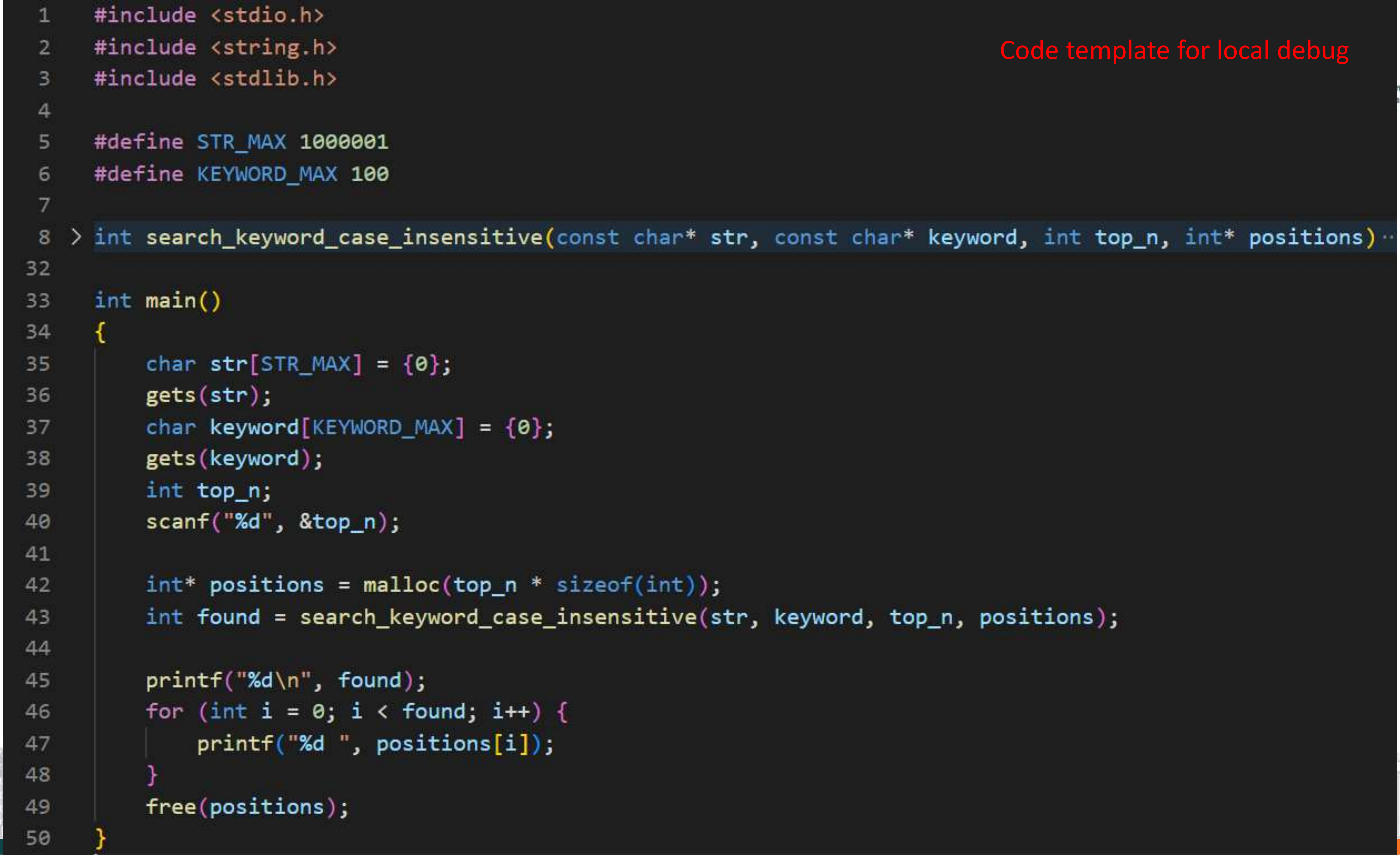
keyword

top_n

Output:

Length of positions

Values of positions



Code template for local debug

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define STR_MAX 1000001
#define KEYWORD_MAX 100

int search_keyword_case_insensitive(const char* str, const char* keyword, int top_n, int* positions)···

int main()
{
    char str[STR_MAX] = {0};
    gets(str);
    char keyword[KEYWORD_MAX] = {0};
    gets(keyword);
    int top_n;
    scanf("%d", &top_n);

    int* positions = malloc(top_n * sizeof(int));
    int found = search_keyword_case_insensitive(str, keyword, top_n, positions);

    printf("%d\n", found);
    for (int i = 0; i < found; i++) {
        printf("%d ", positions[i]);
    }
    free(positions);
}
```

THANK YOU