CS111，C Programming

Lab / Function

黄嘉炜

huangjw3@mail.sustech.edu.cn

Outline

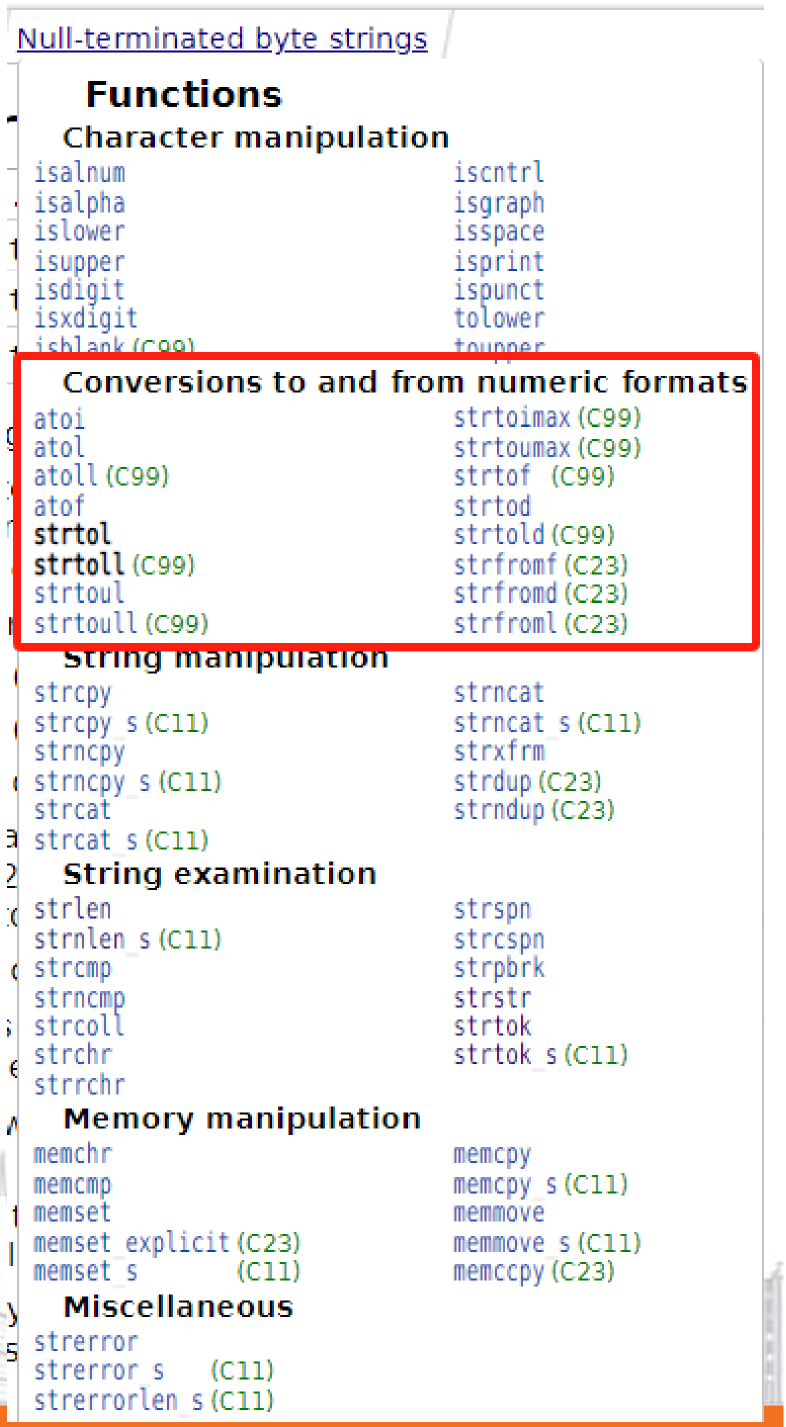
- Review
- Function: More
- Assignment

Review: Extract array from string

How to parse integer value from string?

Given: char* str = “123” ;

1st Idea - Call the function in standard library [if permitted]



Null-terminated byte strings

Functions

Character manipulation

isalnum
isalpha
islower
isupper
isdigit
isxdigit
isblank (C99)
iscntrl
isgraph
isspace
isprint
ispunct
tolower
toupper

Conversions to and from numeric formats

atoi
atol
atoll (C99)
atof
strtol
strtoll (C99)
strtoul
strtoull (C99)
strtoimax (C99)
strtoumax (C99)
strtof (C99)
strtod
strtold (C99)
strfromf (C23)
strfromd (C23)
strfroml (C23)

String manipulation

strcpy
strcpy_s (C11)
strncpy
strncpy_s (C11)
strcat
strcat_s (C11)
strncat
strncat_s (C11)
strxfrm
strdup (C23)
strndup (C23)

String examination

strlen
strnlen_s (C11)
strcmp
strncmp
strcoll
strchr
strrchr
strspn
strcspn
strpbrk
strstr
strtok
strtok_s (C11)

Memory manipulation

memchr
memcmp
memset
memset_explicit (C23)
memset_s (C11)
memcpy
memcpy_s (C11)
memmove
memmove_s (C11)
memccpy (C23)

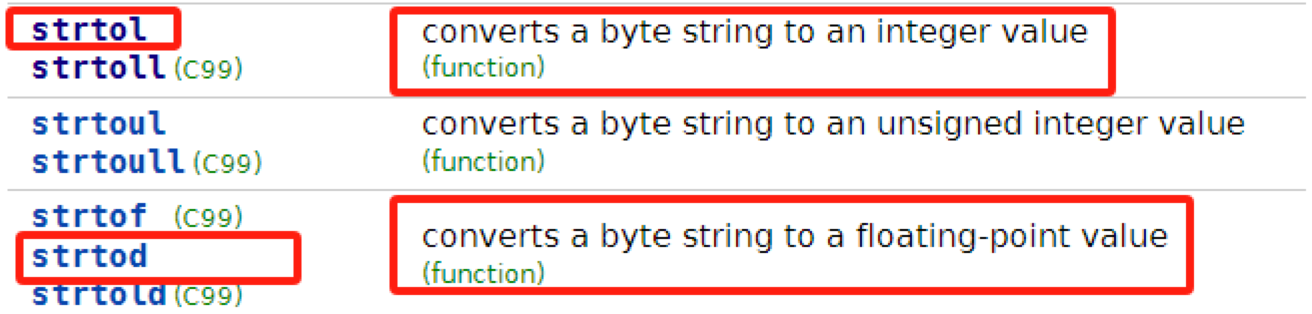
Miscellaneous

strerror
strerror_s (C11)
strerrorlen_s (C11)

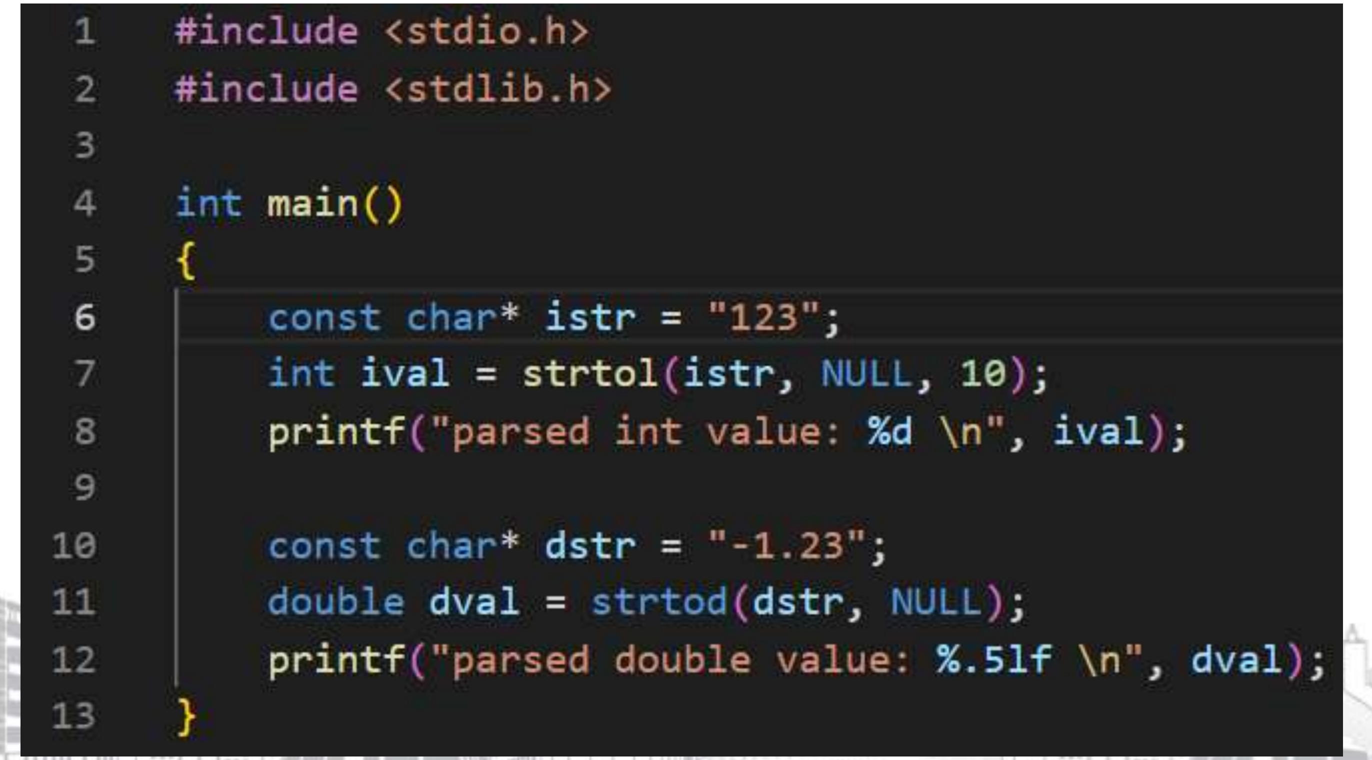
Review: Extract array from string

How to parse integer value from string?

Given: char* str = “123” ;

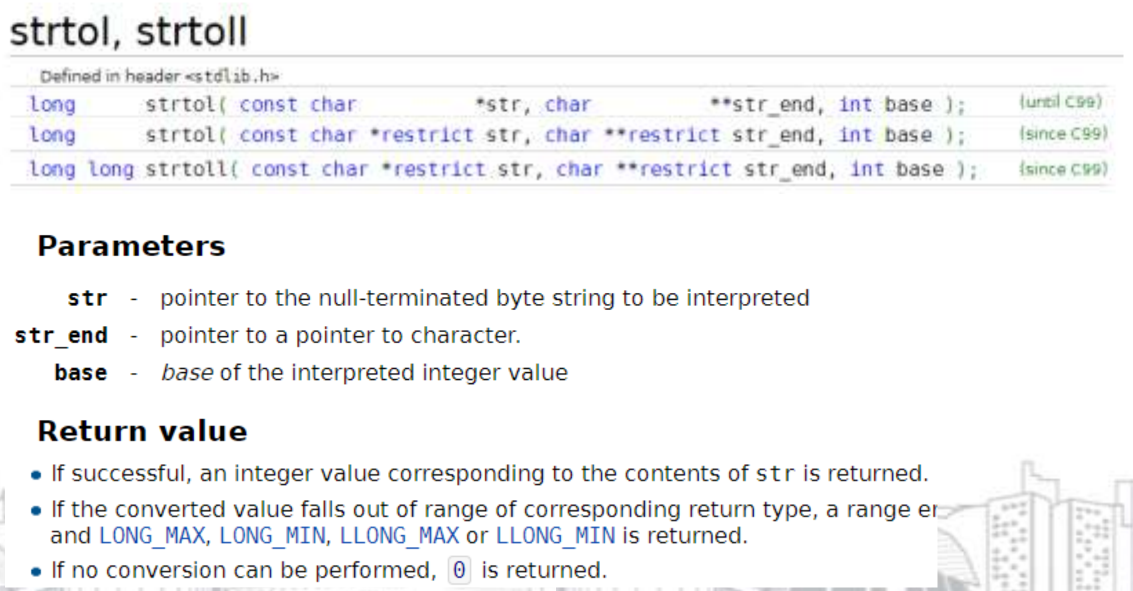


`strtol` `strtoll` (C99)	converts a byte string to an integer value (function)
`strtoul` `strtoull` (C99)	converts a byte string to an unsigned integer value (function)
`strtof` (C99) `strtod` `strtold` (C99)	converts a byte string to a floating-point value (function)



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

int main()
{
    const char* istr = "123";
    int ival = strtol(istr, NULL, 10);
    printf("parsed int value: %d \n", ival);

    const char* dstr = "-1.23";
    double dval = strtod(dstr, NULL);
    printf("parsed double value: %.5lf \n", dval);
}
```

https://en.cppreference.com/w/c/string/byte/strtol
https://en.cppreference.com/w/c/string/byte/strtof



strtol, strtoll

Defined in header `<stdlib.h>`

```c
long      strtol( const char          *str, char          **str_end, int base );   (until C99)
long      strtol( const char *restrict str, char **restrict str_end, int base );   (since C99)
long long strtoll( const char *restrict str, char **restrict str_end, int base );  (since C99)
```

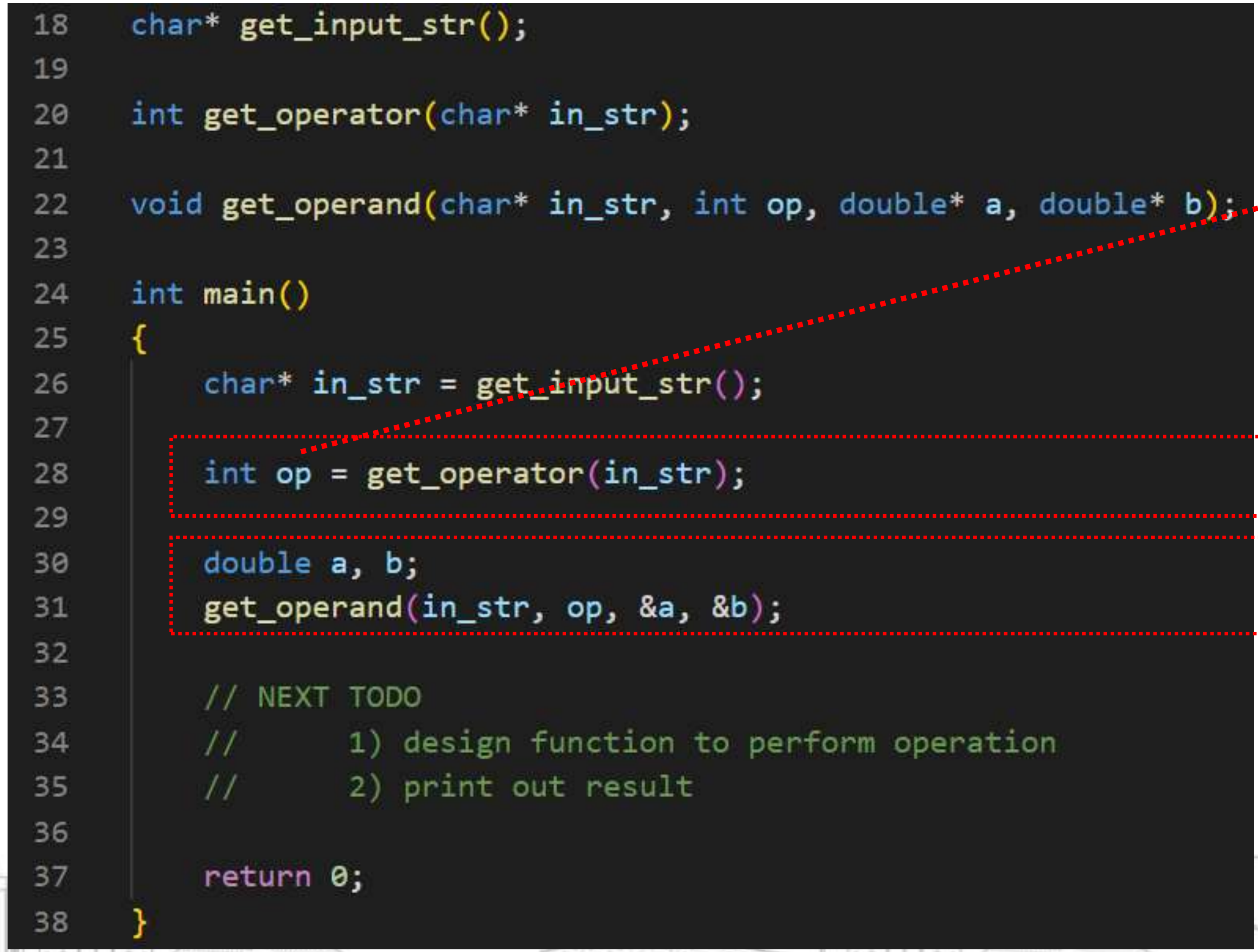
Parameters

str	-	pointer to the null-terminated byte string to be interpreted
str_end	-	pointer to a pointer to character.
base	-	*base* of the interpreted integer value

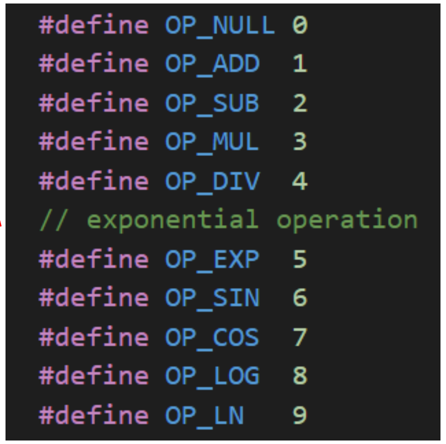
Return value

- If successful, an integer value corresponding to the contents of `str` is returned.
- If the converted value falls out of range of corresponding return type, a range er and `LONG_MAX`, `LONG_MIN`, `LLONG_MAX` or `LLONG_MIN` is returned.
- If no conversion can be performed, `0` is returned.

Review：Simple Calculator



```c
char* get_input_str();

int get_operator(char* in_str);

void get_operand(char* in_str, int op, double* a, double* b);

int main()
{
    char* in_str = get_input_str();

    int op = get_operator(in_str);

    double a, b;
    get_operand(in_str, op, &a, &b);

    // NEXT TODO
    //      1) design function to perform operation
    //      2) print out result

    return 0;
}
```



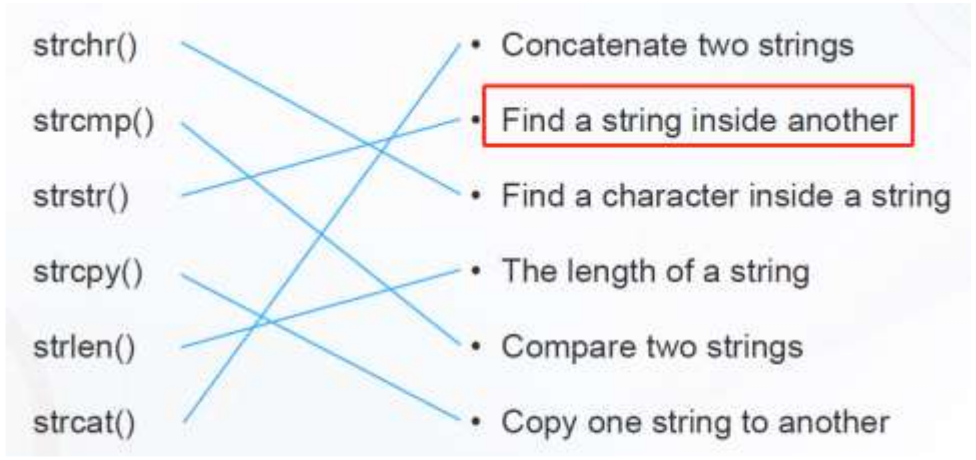
```c
#define OP_NULL 0
#define OP_ADD  1
#define OP_SUB  2
#define OP_MUL  3
#define OP_DIV  4
// exponential operation
#define OP_EXP  5
#define OP_SIN  6
#define OP_COS  7
#define OP_LOG  8
#define OP_LN   9
```

字符串匹配？

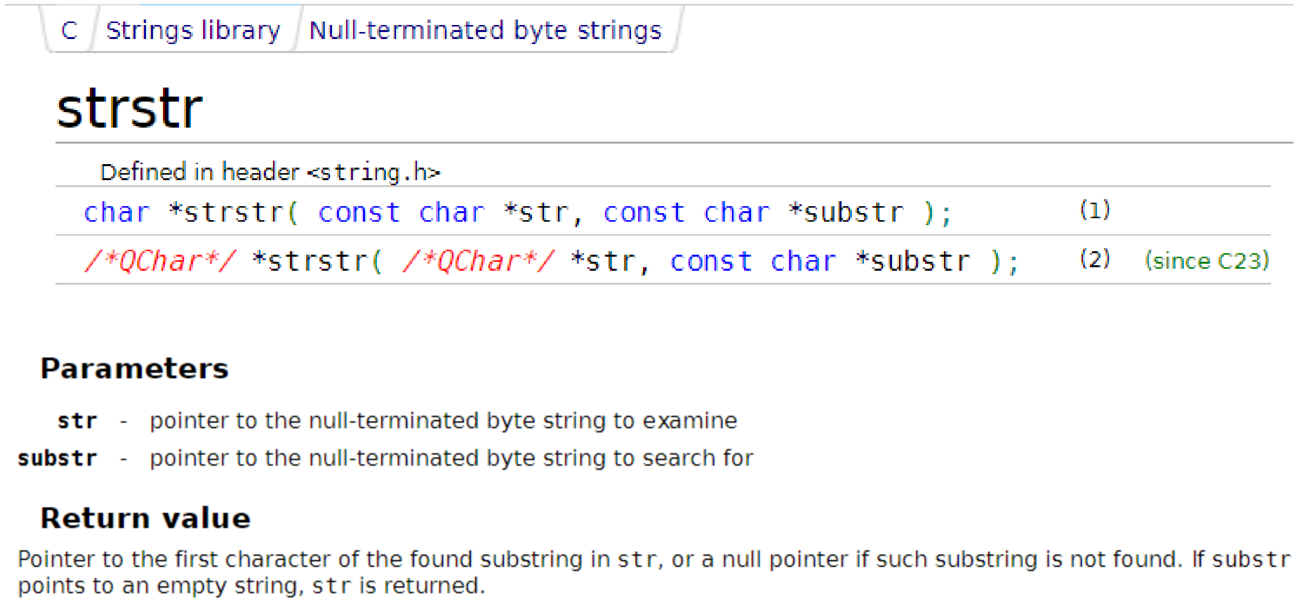
问题进一步拆分：

- 在字符串中，找到数字的起始位置
- 数字的字符串，转化为 double

Q: find a string inside another ?



strchr()
strcmp()
strstr()
strcpy()
strlen()
strcat()
• Concatenate two strings
• Find a string inside another
• Find a character inside a string
• The length of a string
• Compare two strings
• Copy one string to another


C | Strings library | Null-terminated byte strings

strstr

Defined in header <string.h>

```
char *strstr( const char *str, const char *substr );                 (1)
/*QChar*/ *strstr( /*QChar*/ *str, const char *substr );             (2) (since C23)
```

Parameters

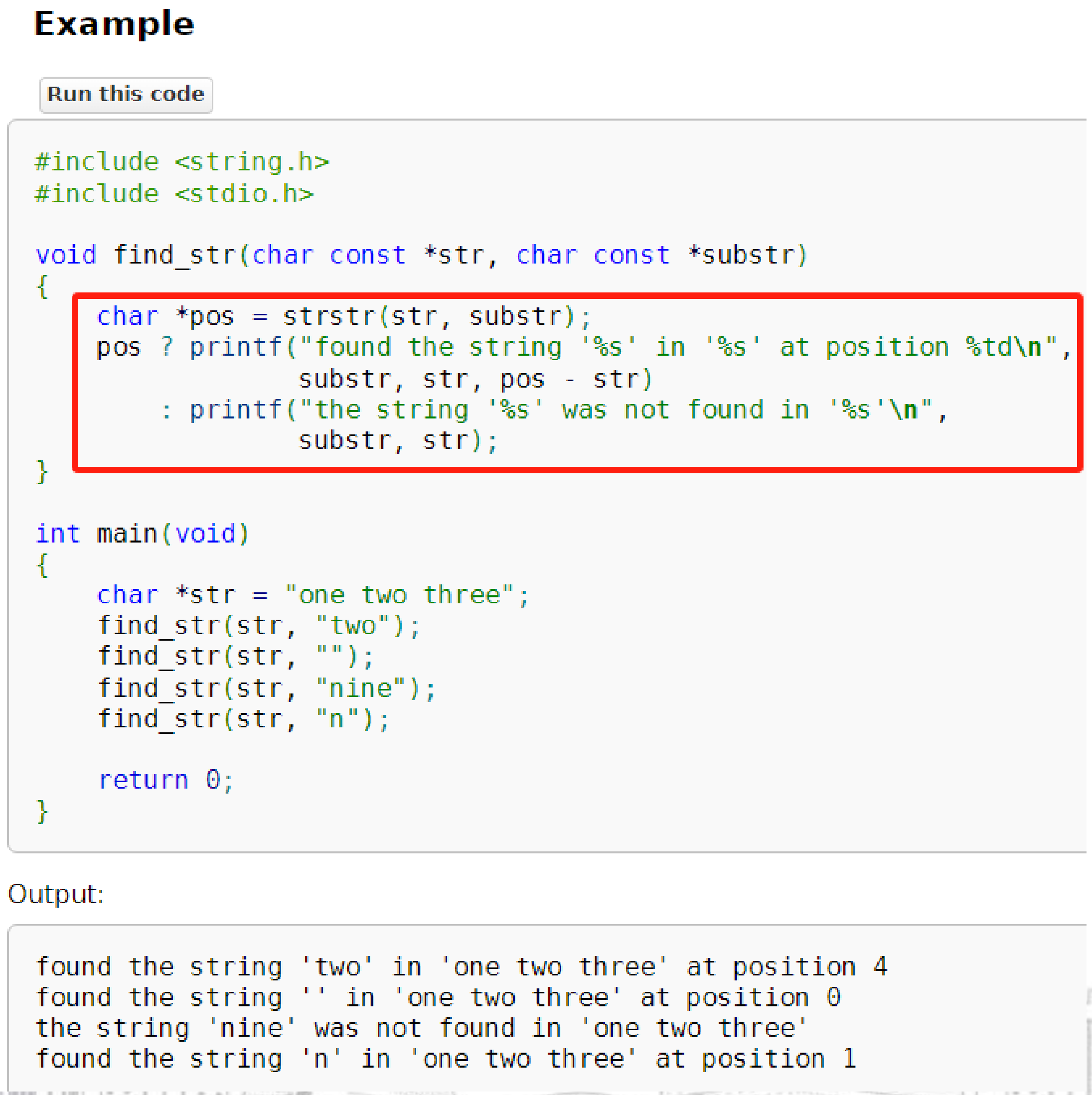
str	-	pointer to the null-terminated byte string to examine
substr	-	pointer to the null-terminated byte string to search for

Return value

Pointer to the first character of the found substring in `str`, or a null pointer if such substring is not found. If `substr` points to an empty string, `str` is returned.

https://en.cppreference.com/w/c/string/byte/strstr

Q: find a string inside another ?



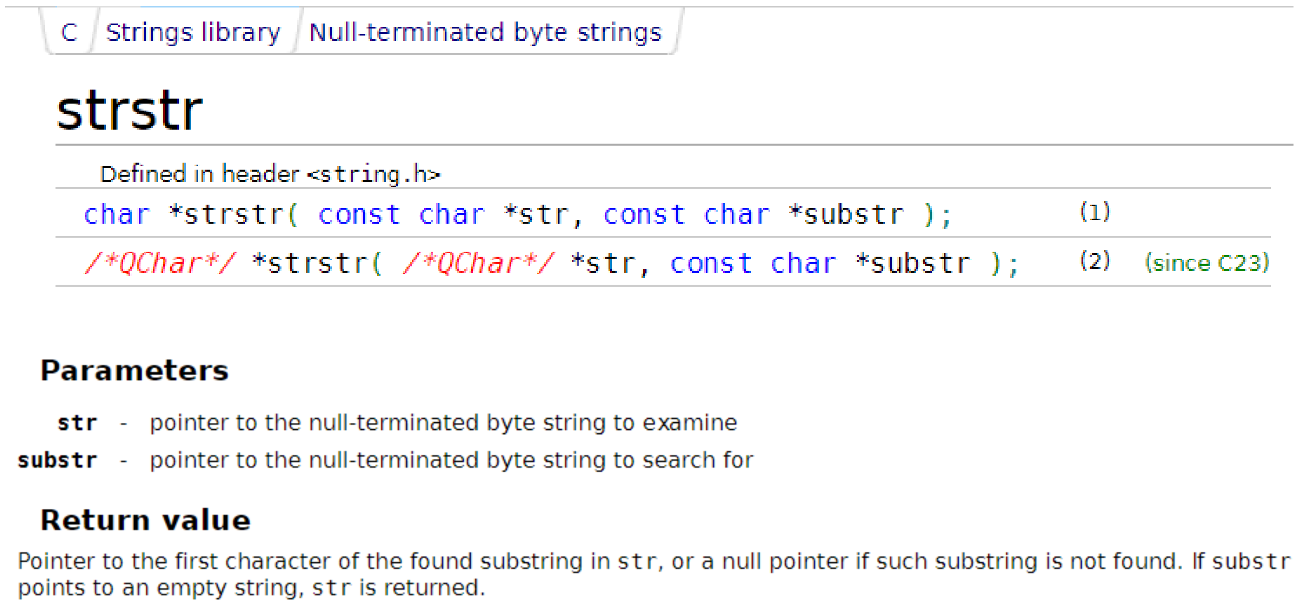
Example

Run this code

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

void find_str(char const *str, char const *substr)
{
    char *pos = strstr(str, substr);
    pos ? printf("found the string '%s' in '%s' at position %td\n",
                 substr, str, pos - str)
        : printf("the string '%s' was not found in '%s'\n",
                 substr, str);
}

int main(void)
{
    char *str = "one two three";
    find_str(str, "two");
    find_str(str, "");
    find_str(str, "nine");
    find_str(str, "n");

    return 0;
}
```

Output:

```
found the string 'two' in 'one two three' at position 4
found the string '' in 'one two three' at position 0
the string 'nine' was not found in 'one two three'
found the string 'n' in 'one two three' at position 1
```



C | Strings library | Null-terminated byte strings

strstr

Defined in header `<string.h>`

`char *strstr( const char *str, const char *substr );`	(1)	
`/*QChar*/ *strstr( /*QChar*/ *str, const char *substr );`	(2)	(since C23)

Parameters

str	-	pointer to the null-terminated byte string to examine
substr	-	pointer to the null-terminated byte string to search for

Return value

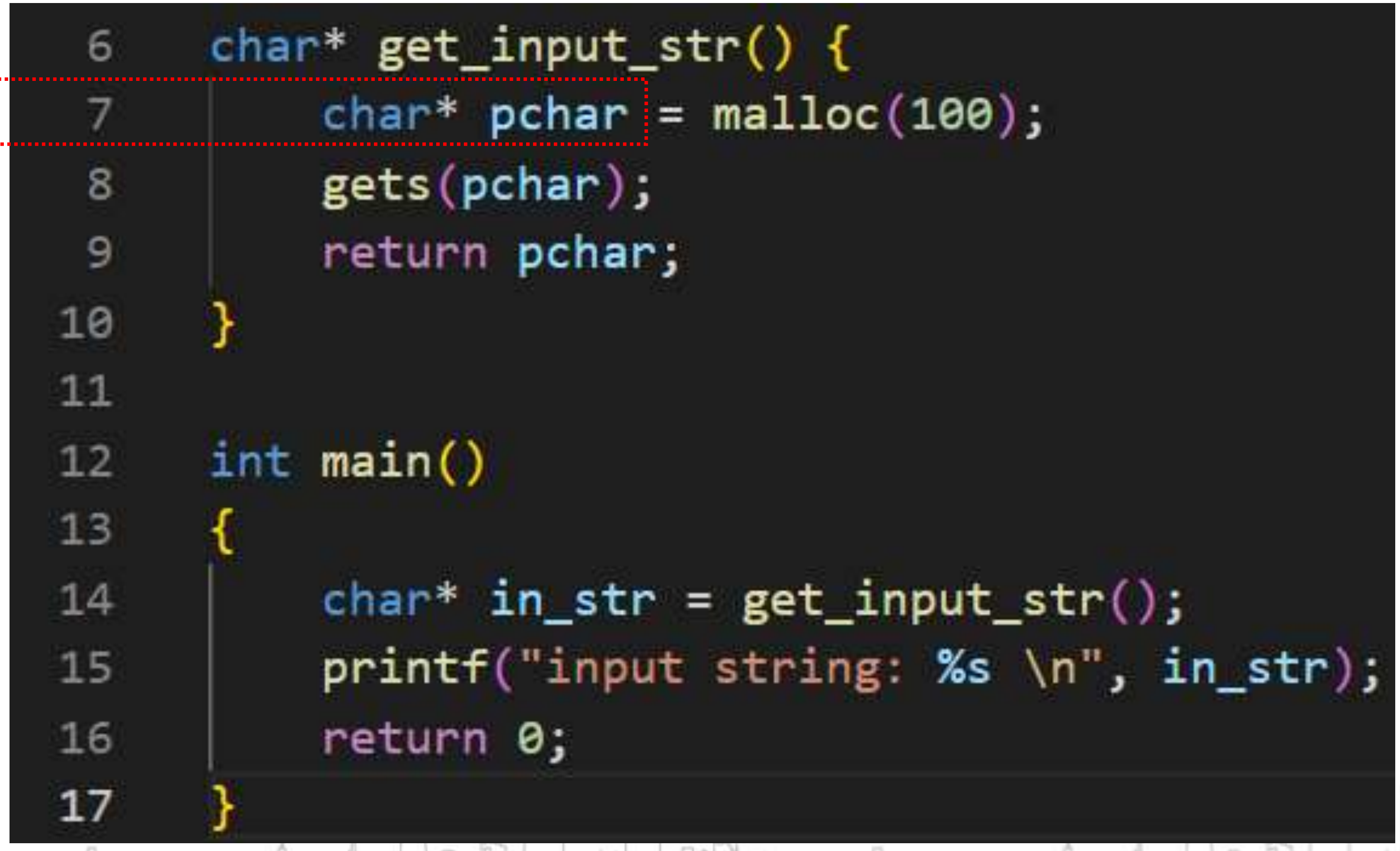
Pointer to the first character of the found substring in `str`, or a null pointer if such substring is not found. If `substr` points to an empty string, `str` is returned.

https://en.cppreference.com/w/c/string/byte/strstr

Outline

- Review
- **Function: More**
- Assignment

Function: Local & Global Variable

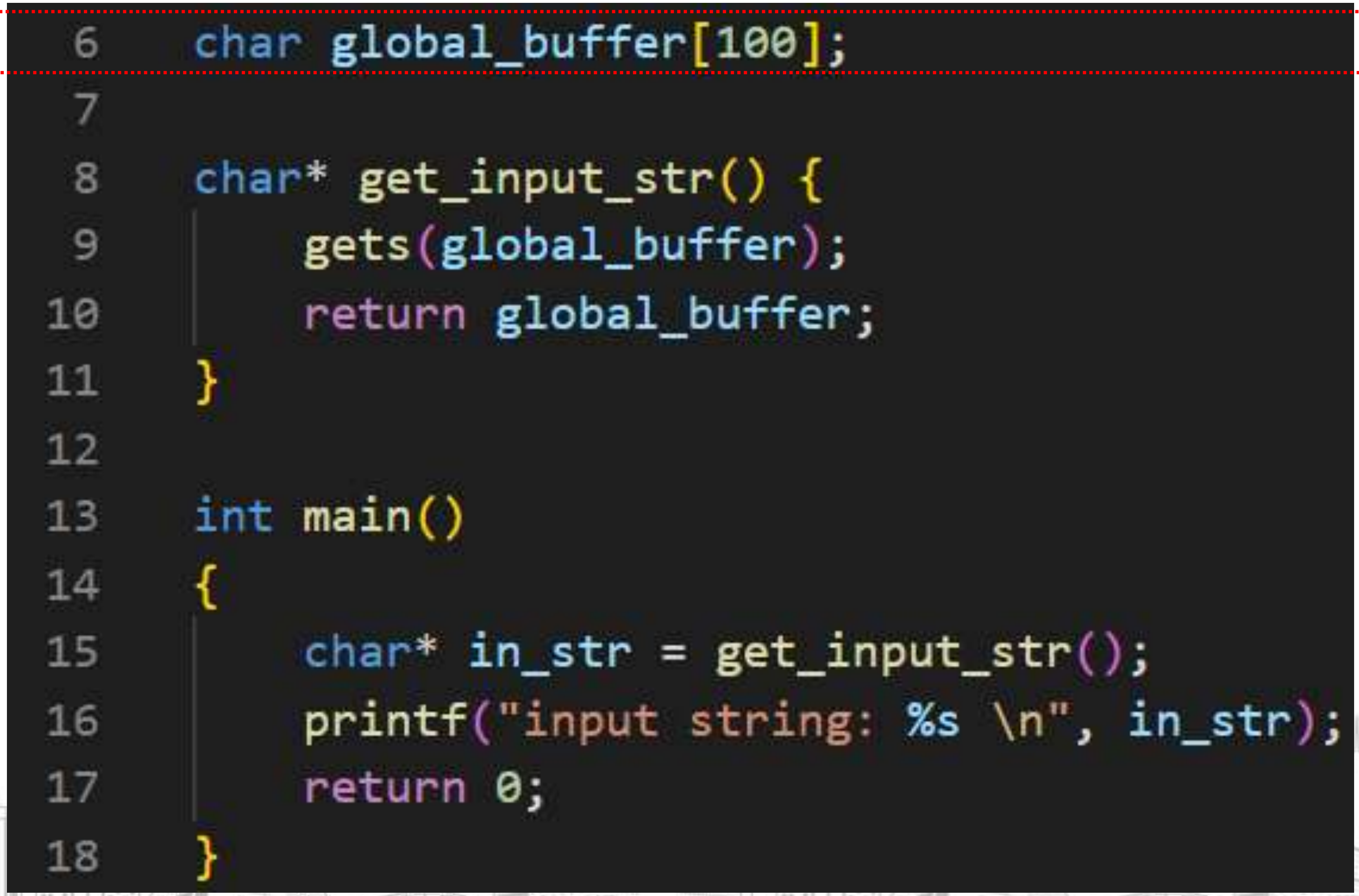


```c
char* get_input_str() {
    char* pchar = malloc(100);
    gets(pchar);
    return pchar;
}

int main()
{
    char* in_str = get_input_str();
    printf("input string: %s \n", in_str);
    return 0;
}
```

函数内部的局部变量

思考：存在问题?

Function: Local & Global Variable



```
char global_buffer[100];

char* get_input_str() {
    gets(global_buffer);
    return global_buffer;
}

int main()
{
    char* in_str = get_input_str();
    printf("input string: %s \n", in_str);
    return 0;
}
```

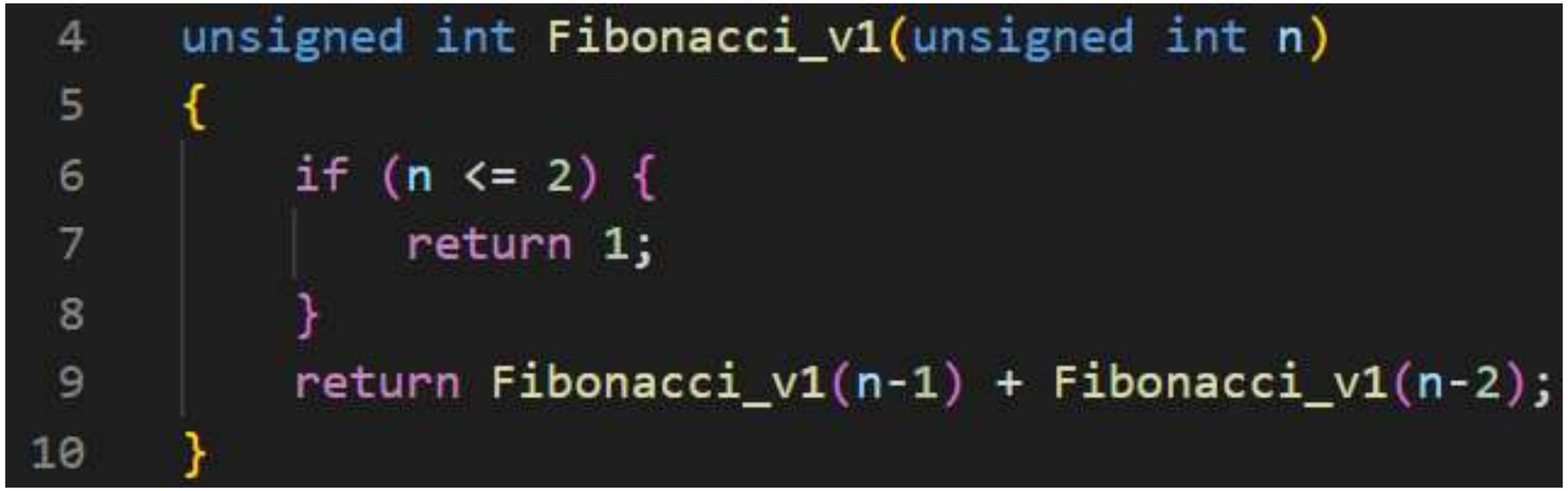
全局变量

- 函数外部定义的变量
- 程序执行期间都存在
- 任何函数内部可访问

作用：跨函数的数据共享

Function: Recursion

经典问题：斐波那契数列



```
unsigned int Fibonacci_v1(unsigned int n)
{
    if (n <= 2) {
        return 1;
    }
    return Fibonacci_v1(n-1) + Fibonacci_v1(n-2);
}
```

递归：函数内部调用自身

⇒ 注意：中止条件处理

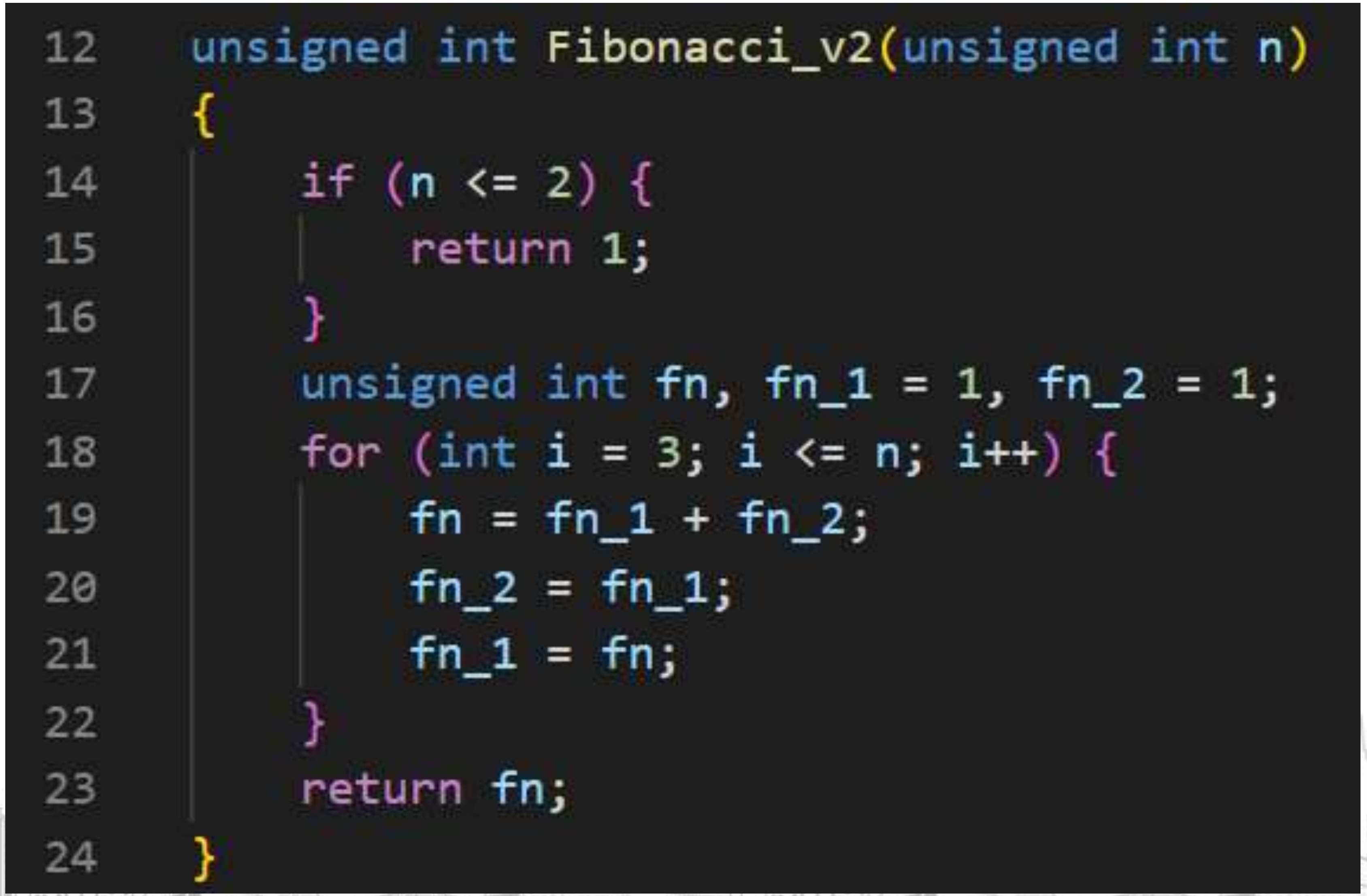
思想：分而治之

⇒ 代码简洁

⇒ 逻辑清晰，容易保证正确性

⇒ 自动处理层次结构

Function: Recursion



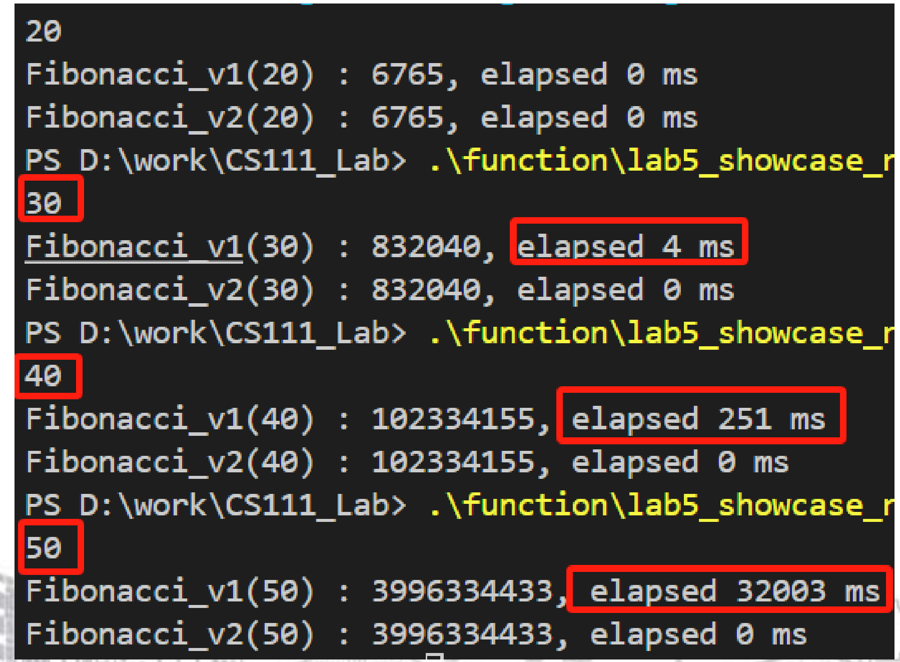
```
unsigned int Fibonacci_v2(unsigned int n)
{
    if (n <= 2) {
        return 1;
    }
    unsigned int fn, fn_1 = 1, fn_2 = 1;
    for (int i = 3; i <= n; i++) {
        fn = fn_1 + fn_2;
        fn_2 = fn_1;
        fn_1 = fn;
    }
    return fn;
}
```

Which better ?

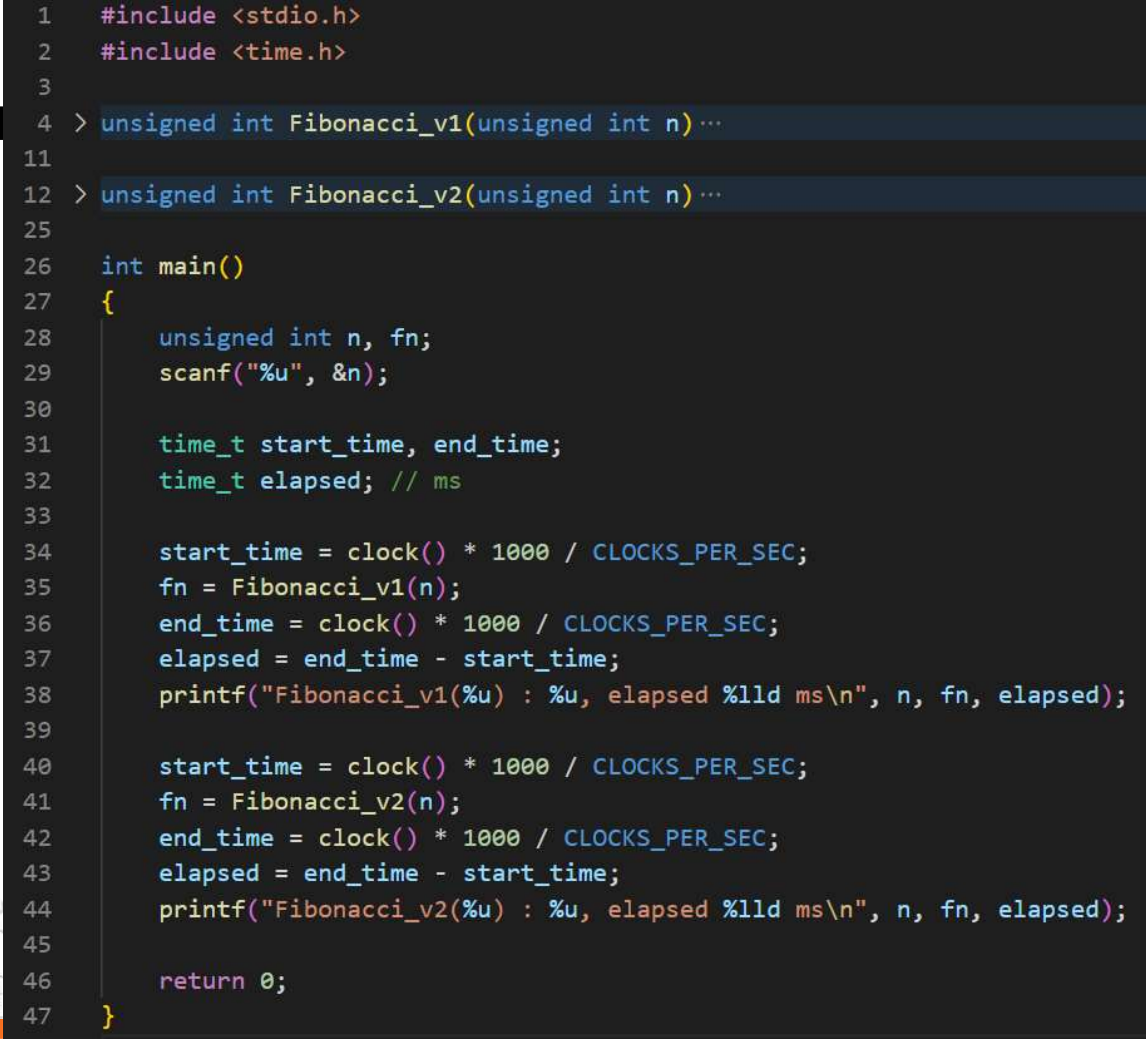
从2个方面思考
⇒ 代码简洁、逻辑清晰
⇒ 性能：内存空间、耗时

Function: Recursio

Confuse? → Test !

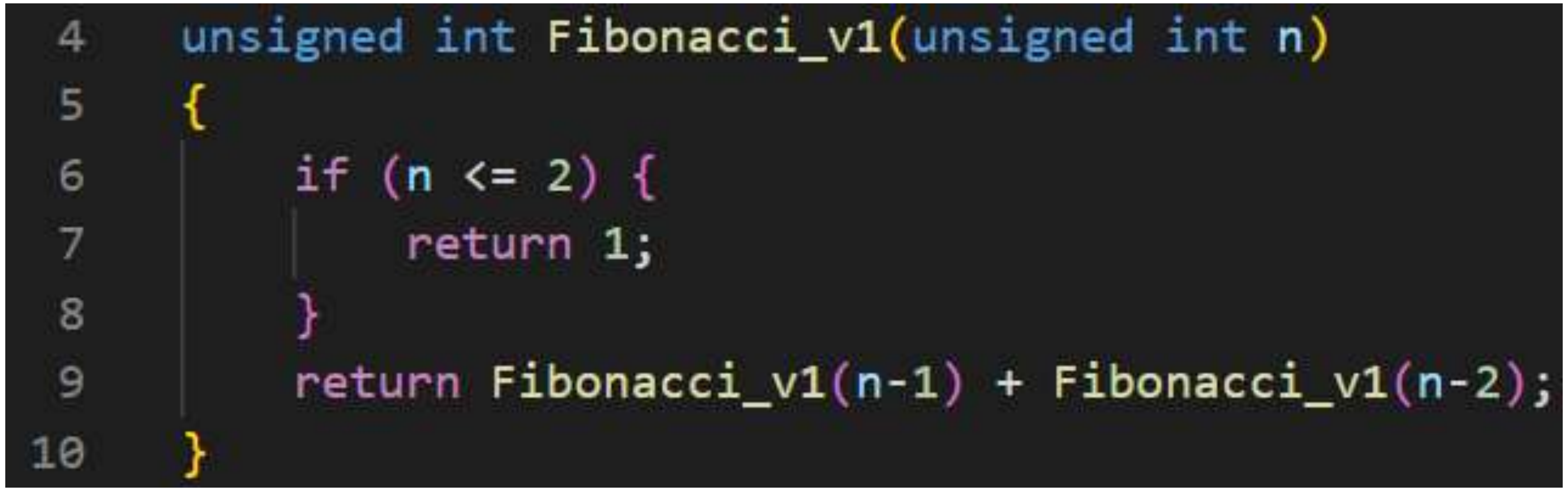


```
20
Fibonacci_v1(20) : 6765, elapsed 0 ms
Fibonacci_v2(20) : 6765, elapsed 0 ms
PS D:\work\CS111_Lab> .\function\lab5_showcase_r
30
Fibonacci_v1(30) : 832040, elapsed 4 ms
Fibonacci_v2(30) : 832040, elapsed 0 ms
PS D:\work\CS111_Lab> .\function\lab5_showcase_r
40
Fibonacci_v1(40) : 102334155, elapsed 251 ms
Fibonacci_v2(40) : 102334155, elapsed 0 ms
PS D:\work\CS111_Lab> .\function\lab5_showcase_r
50
Fibonacci_v1(50) : 3996334433, elapsed 32003 ms
Fibonacci_v2(50) : 3996334433, elapsed 0 ms
```



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

unsigned int Fibonacci_v1(unsigned int n)···

unsigned int Fibonacci_v2(unsigned int n)···

int main()
{
    unsigned int n, fn;
    scanf("%u", &n);

    time_t start_time, end_time;
    time_t elapsed; // ms

    start_time = clock() * 1000 / CLOCKS_PER_SEC;
    fn = Fibonacci_v1(n);
    end_time = clock() * 1000 / CLOCKS_PER_SEC;
    elapsed = end_time - start_time;
    printf("Fibonacci_v1(%u) : %u, elapsed %lld ms\n", n, fn, elapsed);

    start_time = clock() * 1000 / CLOCKS_PER_SEC;
    fn = Fibonacci_v2(n);
    end_time = clock() * 1000 / CLOCKS_PER_SEC;
    elapsed = end_time - start_time;
    printf("Fibonacci_v2(%u) : %u, elapsed %lld ms\n", n, fn, elapsed);

    return 0;
}
```

Function: Recursion



```
unsigned int Fibonacci_v1(unsigned int n)
{
    if (n <= 2) {
        return 1;
    }
    return Fibonacci_v1(n-1) + Fibonacci_v1(n-2);
}
```

递归：函数内部调用自身

思想：分而治之

⇒ 代码简洁

⇒ 逻辑清晰，容易保证正确性

⇒ 自动处理层次结构

缺点：性能不佳

⇒ 内存空间消耗（深度递归可能导致内存空间溢出）

⇒ 时间复杂度高（子问题可能会被重复计算）

Outline

- Review
- Function: More
- **Assignment**

Assignment)

排序 与 搜索

OJ 初步测试
+
现场 Code Review

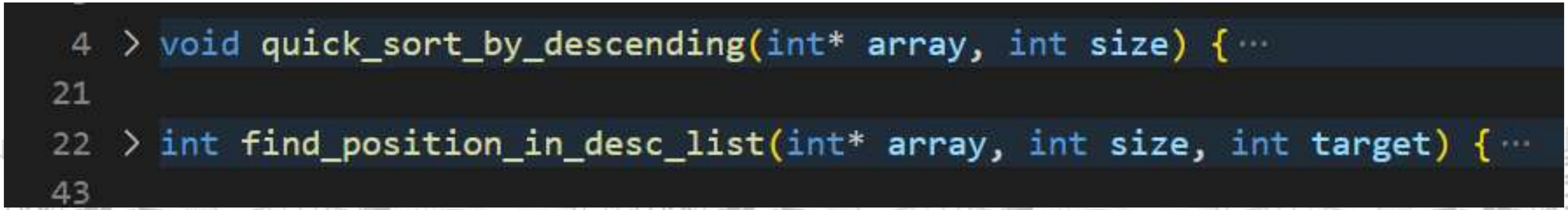
Write 2 functions about:

- quick sort by descending: The input unsorted integer array will be sorted in descending order upon processing.
- search target value in sorted list. Input: integer array, size of array (n), target value. Return: position in input array. if not found, return -1. if have multiple target (same) values, return first occurrence position.

Hints:

- About quick sort , refer to lecture 11th. Modify the sample code to descending sort.
- About search, use binary search, refer to lecture 4th. Recommend to use recursive function.

Note: the function definition as following, and local code template is given.



```
> void quick_sort_by_descending(int* array, int size) {···

> int find_position_in_desc_list(int* array, int size, int target) {···

```

Assignment)

排序 与 搜索

输入数据 1

```
5
1 2 3 4 5
5 1 6
```

Copy

输出数据 1

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

输入数据 2

```
5
1 3 3 3 5
3 6
```

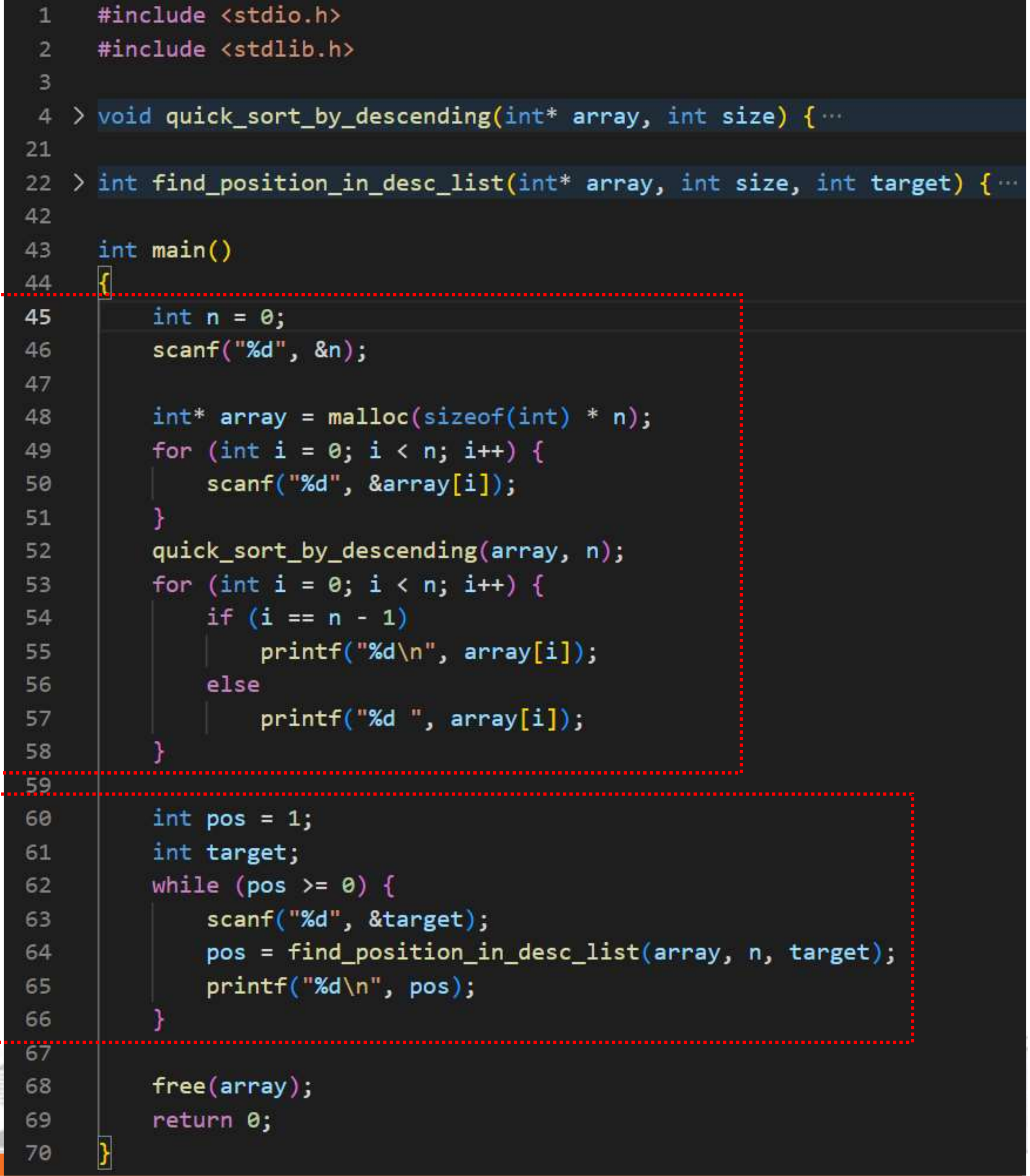
Copy

输出数据 2

```
5 3 3 3 1
1
-1
```

读取输入数组，排序

读取目标值，搜索位置



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

void quick_sort_by_descending(int* array, int size) {···

int find_position_in_desc_list(int* array, int size, int target) {···

int main()
{
    int n = 0;
    scanf("%d", &n);

    int* array = malloc(sizeof(int) * n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &array[i]);
    }
    quick_sort_by_descending(array, n);
    for (int i = 0; i < n; i++) {
        if (i == n - 1)
            printf("%d\n", array[i]);
        else
            printf("%d ", array[i]);
    }

    int pos = 1;
    int target;
    while (pos >= 0) {
        scanf("%d", &target);
        pos = find_position_in_desc_list(array, n, target);
        printf("%d\n", pos);
    }

    free(array);
    return 0;
}
```

SUSTech Southern University of Science and Technology

THANK YOU