注意

Format string %

输出得是双引号带特殊%。

%s，表示字符串

%c, char

%d, integer.

%lu, unsigned long，无符号长整型。

%p，指针地址。pointer，又或者十六进制位

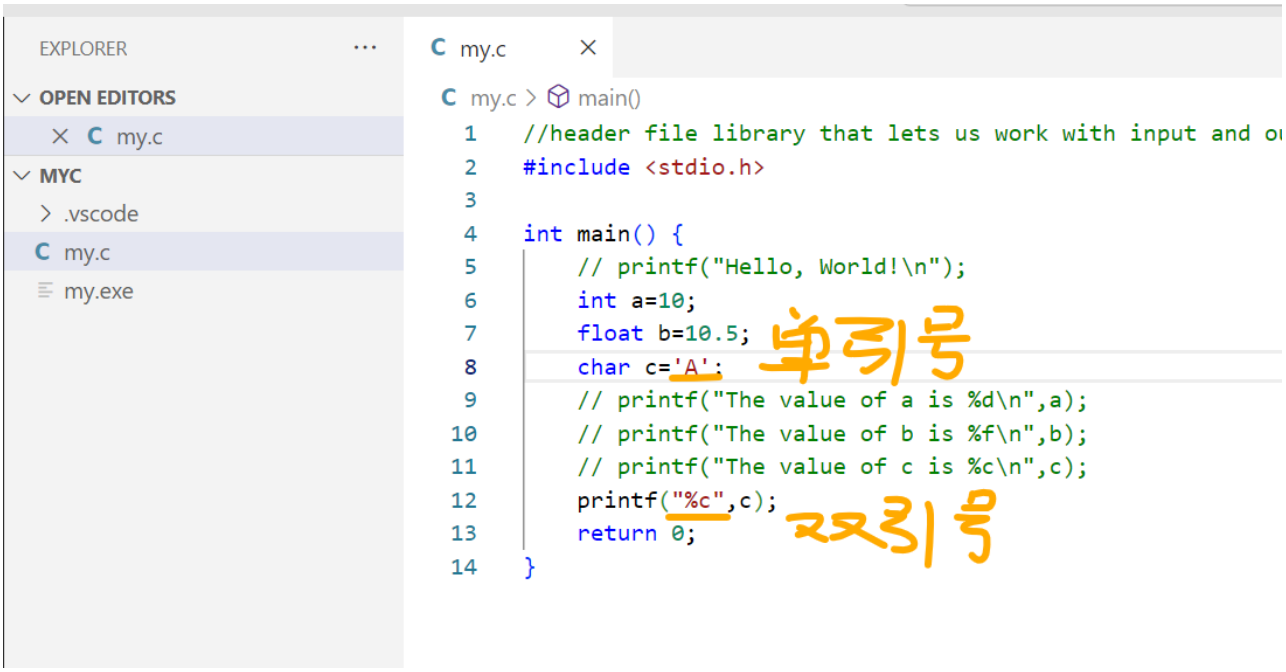
%08x 占位符，换言之，**%就是一个占位符。**

是的，在 C 语言中，**格式化字符串（Format String）** 是指用于格式化输出的字符串，通常包含普通字符和**格式说明符（Format Specifiers）**，比如 `%s`、`%d`、`%x` 等。格式化字符串通常用于 `printf`、`scanf`、`sprintf` 等函数中，用于指定如何格式化输入或输出。

常见的格式说明符

以下是一些常见的格式说明符及其含义：

格式说明符	含义
`%d`	以十进制形式输出带符号整数（`int`）。
`%u`	以十进制形式输出无符号整数（`unsigned int`）。
`%x`	以十六进制形式输出无符号整数（小写字母）。
`%X`	以十六进制形式输出无符号整数（大写字母）。
`%o`	以八进制形式输出无符号整数。
`%f`	以小数形式输出浮点数（`float` 或 `double`）。
`%e`	以科学计数法形式输出浮点数（小写字母 `e`）。
`%E`	以科学计数法形式输出浮点数（大写字母 `E`）。
`%g`	根据值的大小自动选择 `%f` 或 `%e`（更紧凑的形式）。
`%c`	输出单个字符（`char`）。



字符串

```
    // printf("%c",c);

    //用char数组来存储字符串string
    char greeting[]="Hello";
    printf("%s",greeting);
    return 0;
}
```

整数

```
    // operation
    int num1=10,num2=20;
    num1+=100;
    printf("num1=%d\n",num1);
    return 0;
}
```

bool

```
24
25      // bool
26      bool mybool1=true;
27      bool mybool2=false;
28      printf("%d\n",mybool1); //1
29      printf("%d\n",mybool2); //0
30      return 0;
31  }
```

array与for

```
39
40      // for 与array
41      int array2[]={10,4,2,6,3};
42      for (int i = 0; i <5; i++){
43          printf("%d\n",array2[i]);
44      }
45      printf("\n");
46      char array3[]={'A','B','C','D','E'};
47      for (int i=0; i<5;i++){
48          printf("%c\n",array3[i]);
49      }
50      return 0;
51  }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

```
[Running] cd "d:\myC\" && gcc my.c -o my && "d:\myC\"my
10
4
2
6
3

A
B
C
D
E
```

address * &

配合使用**&**，获得变量的所保存在电脑中的地址。

Memory Address

When a variable is created in C, a memory address is assigned to the variable.

The memory address is the location of where the variable is stored on the computer.

When we assign a value to the variable, it is stored in this memory address.

To access it, use the reference operator (`&`), and the result represents where the variable is stored:

Example

```
int myAge = 43;
printf("%p", &myAge); // Outputs 0x7ffe5367e044
```

```
50
51      // p show address
52      int x=10;
53      printf("%p\n",&x); //0061FF10
54      printf("%lu\n",&x); //6422288
55      return 0;
56  }
```

function

```
5   // write a function
6   // function必须写在main函数之前.
7   void myFunction(){
8       printf("This is my function\n");
9   }
10  int main() {
11      // function
12      myFunction();
13      return 0;
14  }
```

变量

```c

#include <stdio.h>
#include <stdbool.h> //bool

// write a function
// function必须写在main函数之前.
void myFunction(char name[],int age) {
    printf("This is my name %s, his age: %d \n",name,age);
}
int main() {
    // function
    myFunction("Casey",19);
    return 0;
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

```
[Running] cd "d:\myC\" && gcc my.c -o my && "d:\myC\"my
This is my name Casey, his age: 19

[Done] exited with code=0 in 0.259 seconds
```

指针数组 参数

要指针数组

```c
void myFunc2(int argc, char *argv[]){
    printf("argc:%d\n",argc);
    for (int i = 0; i < argc; i++)
    {
        printf("%s\n",argv[i]);
    }

}
int main() {
    //func2
    char *myWords[] = {"hello","world"};
    myFunc2(2,myWords);
    return 0;
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

```
[Done] exited with code=0 in 0.246 seconds

[Running] cd "d:\myC\" && gcc my.c -o my && "d:\myC\"my
argc:2
hello
world
```

Pointer

指针，是一个**存储地址**的**变量**

地址，是计算机内存中**每个byte的编号。**

指针（Pointer）与地址（Address）的关系

在 C 语言中，**指针（pointer）就是存储地址的变量。**本质上，指针是一个特殊的变量，它的值是 **内存地址**，指向程序中的某个位置。

---

1. 地址（Address）

地址 指的是计算机内存中每个字节的唯一编号。例如，假设一个整数 `a` 存储在内存中，操作系统会给它分配一个地址：

```c
int a = 10;
printf("%p\n", &a);  // 输出 a 的地址
```

为什么指针，需要指明数据类型？

在 C 语言中，**初始化指针时需要指明数据类型（如 `int*` 或 `char*`）的主要原因是指针的解引用行为和内存地址计算依赖于数据类型的大小**。具体来说，有以下几个关键原因：

---

1. 确定指针解引用时的数据类型

指针的本质是存储内存地址，但不同类型的指针在解引用（`*p`）时，编译器需要知道**应该读取多少个字节的数据。**

示例：

```c
int a = 10;
char c = 'A';

int *p1 = &a;  // 指向 int 类型的指针
char *p2 = &c; // 指向 char 类型的指针

printf("%d\n", *p1);  // 访问的是 4 字节的整数
printf("%c\n", *p2);  // 访问的是 1 字节的字符
```

- `*p1` 访问 `int`，一般占 **4 字节。**
- `*p2` 访问 `char`，占 **1 字节。**
- 如果 `p1` 和 `p2` 都是 `void*`（无类型指针），编译器就无法知道该访问多少字节的数据。

数组与指针

数组的起始点地址，也是第一个元素的地址。

```c
}
int main() {
    // function
    int myArray[5]={18,17,3,4,5};
    printf("%p\n",myArray);//0061FF0C
    printf("%p\n",myArray[0]);
    printf("%p\n",&myArray[0]);
    return 0;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMM
[Running] cd "d:\myC\" && gcc my.c -o my && "d:\myC\"my
0061FF0C
00000012
0061FF0C
```

" * "的使用

通过指针，来访问其地址背后的值。

```c
int main() {
    // function
    int myArray[5]={18,17,3,4,5};
    printf("%p\n",myArray);//0061FF0C
    printf("%d\n",*myArray);// * 可通过指针，访问其元素
    return 0;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS
[Running] cd "d:\myC\" && gcc my.c -o my && "d:\myC\"my
0061FF0C
18
```

指针数组，指向指针的指针。

main(int argc, char *argv[]) 为例子

char *argv[]，该数组每个元素都是一个char*指针。*

为什么是指针的指针，**因为数组本身，也是一个指针。**

argc，计算输入的参数的总个数。

```bash
./myprogram hello world
```

那么：

- `argc == 3`（包括程序名 `./myprogram`）
- `argv[0] == "./myprogram"`
- `argv[1] == "hello"`
- `argv[2] == "world"`

1. 语法解析

```c
int main(int argc, char *argv[])
```

- `argc` (**argument count**)：表示命令行参数的 **个数**（包括程序本身的名称）。
- `argv` (**argument vector**)：是一个 **字符串数组**，用于存储所有命令行参数。
  - `argv[0]` 是 **程序的名称。**
  - `argv[1]` 到 `argv[argc - 1]` 是 **用户输入的参数。**

`char *argv[]` 和 `char **argv` 本质上是一样的，都是 **指向指针的指针。**

```c
#include <stdio.h>

int main(int argc, char *argv[]) {
    printf("参数个数: %d\n", argc);
    for (int i = 0; i < argc; i++) {
        printf("argv[%d]: %s\n", i, argv[i]);
    }
    return 0;
}
```

运行方式

如果程序名是 `program`，运行：

```sh
./program hello world 123
```

那么 `argc = 4`，参数数组 `argv` 解析如下：

```less
argv[0]: ./program
argv[1]: hello
argv[2]: world
argv[3]: 123
```

function 中的指针数组参数

要指针数组

```c
void myFunc2(int argc, char *argv[]){
    printf("argc:%d\n",argc);
    for (int i = 0; i < argc; i++)
    {
        printf("%s\n",argv[i]);
    }

}
int main() {
    //func2
    char *myWords[] = {"hello","world"};
    myFunc2(2,myWords);
    return 0;
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

```
[Done] exited with code=0 in 0.246 seconds

[Running] cd "d:\myC\" && gcc my.c -o my && "d:\myC\"my
argc:2
hello
world
```

完美解释 —— 指向指针的指针。

指针保存的是地址，比如 ptrA --->ptrB。ptrA指针，存储了ptrB指针的地址。

指向指针的指针，其含义是不是a指针指向了b指针的地址?

是的，**指向指针的指针（Pointer to Pointer）** 其实就是一个指针，它存储了另一个指针的地址。也就是说：

- **一级指针**：存储变量的地址。
- **二级指针**（指向指针的指针）：存储一级指针的地址。

```c
int main(int argc, char **argv) { ... }
```

本质上，`argv` 是一个**指向字符串指针的指针**：

```lua
argv
 ↓
+-----------+      +-------------+
| argv[0]   | -->  | "./program" |
+-----------+      +-------------+
| argv[1]   | -->  | "hello"     |
+-----------+      +-------------+
| argv[2]   | -->  | "world"     |
+-----------+      +-------------+
| argv[3]   | -->  | "123"       |
+-----------+      +-------------+
| argv[4]   | -->  | NULL        |  // 结束符
+-----------+
```

%n

计算字符的总数

%n是什么意思?

在 C 语言的 `printf` 函数中，`%n` 是一个特殊的格式说明符，它**不会打印任何内容**，而是**将当前已输出的字符数存入指定的变量。**

计算字符总数

---

1. 语法

```c
printf("%n", &变量);
```

- `%n` **不会输出任何东西**，但会将当前 `printf` 已经输出的字符个数存入 `&变量` 指向的地址。
- `&变量` 必须是 `int*` 类型的指针，否则会导致错误。

---

strncpy函数

指令，strncpy是什么含义?

`strncpy` 是 C 语言标准库中的一个 **字符串拷贝函数**，用于从 **源字符串** 复制 **最多 n 个字符** 到 **目标字符串**。它定义在 `<string.h>` 头文件中。

---

1. 语法

```c
char *strncpy(char *dest, const char *src, size_t n);
```

参数

- `dest`：目标字符串（接收拷贝的数据）。
- `src`：源字符串（提供要拷贝的数据）。
- `n`：最多拷贝的字符数量。

gets函数

读取一行输入。

> The C library **gets()** function is used to read a line from the standard input stream (stdin) and store it into the string pointed to by str. It continues reading characters from the input stream until a newline character is encountered or the end-of-file is reached. The newline character is then replaced by a null terminator, and the resulting string is stored in str.
>
> ## Syntax

fgets()

简单来说，就是用于读取数据。

C 库函数 - fgets()

C 标准库 - <stdio.h>

描述

C 库函数 **char *fgets(char *str, int n, FILE *stream)** 从指定的流 stream 读取一行，并把它存储在 **str** 所指向的字符串内。当读取 **(n-1)** 个字符时，或者读取到换行符时，或者到达文件末尾时，它会停止，具体视情况而定。

声明

下面是 fgets() 函数的声明。

```
char *fgets(char *str, int n, FILE *stream)
```

参数

- **str** -- 这是指向一个字符数组的指针，该数组存储了要读取的字符串。
- **n** -- 这是要读取的最大字符数（包括最后的空字符）。通常是使用以 str 传递的数组长度。
- **stream** -- 这是指向 FILE 对象的指针，该 FILE 对象标识了要从中读取字符的流。

返回值

如果成功，该函数返回相同的 str 参数。如果到达文件末尾或者没有读取到任何字符，str 的内容保持不变，并返回一个空指针。
如果发生错误，返回一个空指针。

解引用 * (...)

用于，获取指针所指向的value.

🌟 你提到的两部分分别是：

1. `long *para`
   ✅ 表示：para 是一个指向 `long` 类型的指针。
2. `*(long *)para`
   🔍 这是一种类型转换 + 解引用的组合操作，咱们一层一层来理解。

🧠 拆解 `*(long *)para` 的含义：

它等价于：

```c
(long *)para    // 强制类型转换，把 para 当成 long 类型的指针
*(...)          // 对转换后的指针进行解引用，获取它指向的值
```

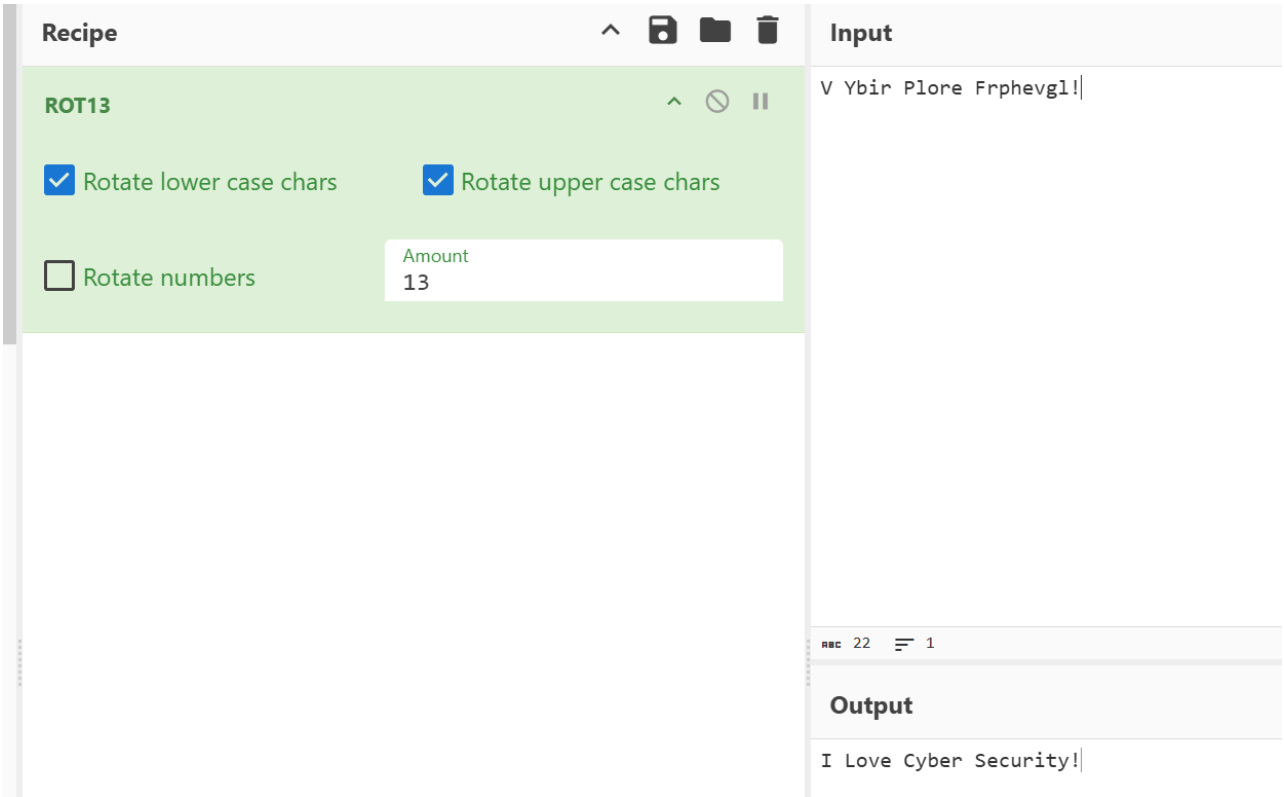
strcmp() 比较字符串

比较两字符串，如果str1的第一个字母比str2的第一个字母大，则return >0，反之return <0

若相同，则return 0

> ### Definition and Usage
>
> The `strcmp()` function compares two strings and returns an integer indicating which one is greater.
>
> For this comparison characters at the same position from both strings are compared one by one, starting from the left until one of them does not match or the end of a string has been reached. There are three possible scenarios:
>
> - If the end of both strings has been reached without any mismatches then the function returns zero.
> - At the first mismatch, if the ASCII value of the character in the first string is greater then the function returns a positive number.
> - At the first mismatch, if the ASCII value of the character in the second string is greater then the function returns a negative number.
>
> The `strcmp()` function is defined in the `<string.h>` header file.

rot rotate

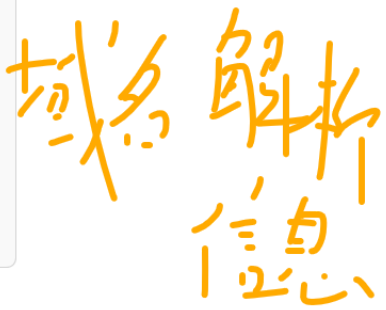


hostent

🔍 一、什么是 `hostent`？

`hostent` 是一个结构体（`struct hostent`），在 C 的 `<netdb.h>` 头文件中定义，用于表示一个**域名解析后的信息（主机信息）**。

```c
struct hostent {
    char  *h_name;       // 主机的官方名字
    char **h_aliases;    // 主机的别名列表
    int    h_addrtype;   // 地址类型（通常是 AF_INET）
    int    h_length;     // 地址长度（通常是 4 字节）
    char **h_addr_list;  // 一个 IP 地址的列表（可以有多个 IP）
};
```



例如，当你调用 `gethostbyname("www.example.com")` 时，它会返回一个 `struct hostent *` 指针，其中就包含了这个域名对应的 IP 地址等信息。

```c
struct hostent *phVar1;
phVar1 = gethostbyname(".csf.is.a.great.course.yay");
if (phVar1 == (hostent *)0x0) {
    puts("Sorry, no secret for you.");
} else {
    print_secret();
}
```

分析逻辑：

- `gethostbyname()` 尝试解析主机名 `".csf.is.a.great.course.yay"`。
- 如果解析失败，返回值是 `NULL`（即 `(struct hostent *)0x0`）。
- 所以这个判断：

✅ 方法一：修改 `/etc/hosts` 文件（最常见最有效）

这是 **最经典的绕过方法**，我们让系统"以为"这个域名是有效的！

步骤：

1. 打开 Linux 的 hosts 文件：

bash 複製 編輯

```
sudo nano /etc/hosts
```

2. 加入一行（127.0.0.1 是任意本地可用 IP）：

bash 複製 編輯

```
127.0.0.1 .csf.is.a.great.course.yay
```

3. 保存退出。
4. 重新运行你的程序！

💥 现在 `gethostbyname()` 会返回合法指针（因为它能解析这个主机名了），就会执行 `print_secret()`！