输入读取

读取整行字符串（包括空格）

```c
char line[200];
fgets(line, sizeof(line), stdin);
```

连续读多行直到 EOF

```c
char s[200];
while (fgets(s, sizeof(s), stdin)) {
 // 处理每一行
}
```

处理单行多元素输入

用 `strtok` 分割：

```c
char s[200];
fgets(s, sizeof(s), stdin); char *p = strtok(s, ",");
while (p != NULL) {
 int x = atoi(p);
 // 用 x
 p = strtok(NULL, ","); // 从原位置继续往后切
}
```

用空格分开时将`strtok`的分隔符设置成空格即可。

对于存入数组的情况：

```c
char line[200];
int arr[50], cnt = 0;
fgets(line, sizeof(line), stdin);
char *p = strtok(line, " ");
while (p != NULL) {
 arr[cnt++] = atoi(p);
 p = strtok(NULL, " ");
}
```

连续读取未知数量数字直到换行

```c
int x;
while (scanf("%d", &x) == 1) {
 // 用 x
 if (getchar() == '\n')
      break;
      // 换行就停
}
```

读取带空格的单词，直到遇到换行

示例输入：

`apple banana orange`

```c
char s[100];
while (scanf("%s", s) == 1) {
 // 处理 s
 if (getchar() == '\n')
      break;
}
```

scanf 与 fgets 混用

此时使用以下写法防止相互干扰：

```c
int n; char line[200];
scanf("%d", &n);
getchar(); // 吃掉\n
fgets(line, sizeof(line), stdin);
```

字符串 → 数组 → 排序

```c
char line[200];
int arr[100], cnt = 0;

// 读取整行
if (fgets(line, sizeof(line), stdin) == NULL) return 0;

// strtok 分割（空格分隔）
char *p = strtok(line, " \n"); // 顺便把 '\n' 也当分隔符
while (p != NULL) {
    arr[cnt++] = atoi(p);
    p = strtok(NULL, " \n");
```

```
}

// 排序
qsort(arr, cnt, sizeof(int), cmp);
```

字符串处理

记得要引入 `string.h` 才能使用下列方法

长度/比较

```
#include <string.h>
size_t len = strlen(s);
int r = strcmp(a, b); // <0, =0, >0
```

子串检索

```
char* p = strchr(s, 'x');   // 第一次出现 'x' 的位置
char* q = strstr(s, "abc"); // 第一次出现子串 "abc"的位置
```

分割字符串

```
#include <string.h>
char line[200];
fgets(line, sizeof(line), stdin);
for(char* p = strtok(line, " ,\n"); p; p = strtok(NULL, " ,\n"))
{
 // p 是每个 token
}
```

> **Important**
>
> 这里for循环中的 `p` 等价于 `p≠NULL`，后面传入NULL表示从上一次 `strtok` 停下的位置继续往后切。

类型处理

记得要引入 `ctype.h` 才能使用下列方法

- `isdigit`：是否是数字字符 `'0'..'9'`
- `isalnum`：是否是字母或数字（混合判断）

- `isspace`：是否是空白字符（空格、换行、Tab 等）
- `islower` / `isupper`：是否是小写 / 大写字母
- `tolower` / `toupper`：大小写转换

这几个因为用法太显而易见了就不给示例了。

妙妙工具

qsort

记得引入`stdlib.h`。基本用法形如：

```
qsort(base, n, size, cmp);
```

- `base`：数组首地址（比如 `arr`）
- `n`：元素个数
- `size`：每个元素的字节数（通常 `sizeof(arr[0])`）
- `cmp`：比较函数指针（你自己写，决定升序/降序/按什么字段排）

使用示例：

给 int 数组升序排序

```
#include <stdio.h>
#include <stdlib.h>
int cmp_int_asc(const void *a, const void *b) {
 int x = *(const int*)a;
 int y = *(const int*)b;
 return (x > y) - (x < y); // 避免 x-y 溢出
}

int main() {
 int arr[] = {10, 3, 25, -7, 8};
 int n = sizeof(arr) / sizeof(arr[0]);
 qsort(arr, n, sizeof(arr[0]), cmp_int_asc);

 for (int i = 0; i < n; i++)
      printf("%d ", arr[i]);
 return 0;
}
```

> `cmp` 返回值的规则

比较函数 `cmp(a,b)` 必须满足：

- 返回 **<0**：表示 `a` 应该排在 `b` 前面
- 返回 **0**：表示相等
- 返回 **>0**：表示 `a` 应该排在 `b` 后面

排结构体

假设要按 `score` 升序排，score 相同按 `id` 升序：

```c
typedef struct {
 int id;
 int score;
} Node;

int cmp_node(const void *a, const void *b) {
 const Node *x = (const Node*)a;
 const Node *y = (const Node*)b;
 if (x->score != y->score)
      return (x->score > y->score) - (x->score < y->score);
 return (x->id > y->id) - (x->id < y->id);
}

int main() {
 qsort(nodes, n, sizeof(nodes[0]), cmp_node);
}
```

printf 格串

补零（或者补别的什么）

```c
printf("%05d\n", 42);   // 00042
printf("%02d:%02d\n", 3, 7); // 03:07
```

精度（保留位数）

浮点：控制小数位数（四舍五入）

```c
printf("%.2f\n", 3.14159); // 3.14
printf("%.0f\n", 3.9);     // 4
```

字符串：最大输出长度

```
printf("%.3s\n", "abcdef"); // abc
```

动态宽度/精度

宽度由参数给：

```
int w = 8;
printf("^%0*d^\n", w, 123); // "^00000123^"
```

内存操作

`malloc`：申请一块未初始化的内存

特点：申请到的内容是垃圾值，要手动初始化。

```
int n = 10;
int *a = (int*)malloc(n * sizeof(int)); // 申请10个int
if (a == NULL) {
 // 内存不足
}
```

`calloc`：申请 + 自动清零

```
int n = 10;
int *a = (int*)calloc(n, sizeof(int)); // 10个int，全部为0
```

（这俩都是用完要`free()`的，你最好别忘了）

memset

```
int a[100]; memset(a, 0, sizeof(a));
```

memcpy

```
int a[5] = {1,2,3,4,5};
int b[5];
memcpy(b, a, sizeof(a));
```

文化常识（确信）

ASCII相关

计数排序必需品：

- 大写字母范围是 **65 ~ 90**
- 小写字母范围是 **97 ~ 122**
- `'0'` = **48**
- `'9'` = **57**

标准 ASCII 可打印字符是：

- **32 ~ 126**（共 95 个）
- 其中 **32 是空格** `' '`

位运算

`&`	**按位与操作，按二进制位进行"与"运算**	**`(A & B)` 将得到 `12` 即为 0000 1100**
`|`	按位或运算符，按二进制位进行"或"运算	`(A | B)` 将得到 `61` 即为 0011 1101
`^`	异或运算符，按二进制位进行"异或"运算	`(A ^ B)` 将得到 `49` 即为 0011 0001
`~`	取反运算符，按二进制位进行"取反"运算	`(~A)` 将得到 `-61` 即为 1100 0011
`<<`	二进制左移运算符	`A << 2` 将得到 `240` 即为 1111 0000
`>>`	二进制右移运算符	`A >> 2` 将得到 `15` 即为 0000 1111

break & continue

break跳出整个循环体，continue步出单次循环

数组指针 & 指针数组

指针数组

```
1  int *p[3];
```

含义：`p` 是一个长度为 3 的数组，`p[i]` 是 `int*`

```
1  char *s[3] = {"aa", "bb", "cc"};
2  printf("%s\n", s[1]); // bb
```

数组指针

```c
int (*p)[3];
```

```c
int a[2][3] = {{1,2,3},{4,5,6}};
int (*p)[3] = a;      // 指向第0行
printf("%d\n", p[1][2]); // 6
```

二维数组

```c
int a[2][3] = {
    {1, 2, 3},
    {4, 5, 6}
};
// a 是一个2个元素的数组，每个元素是一个3个元素的数组
// 也就是：int[3] t[2]; 本质上 [3] 和 int 配对
```

行指针 & 列指针

行指针

行指针 = 指向“一整行”的指针
也就是：指向 `int[列数]` 这种数组的指针。

如果有 4 列：

```c
int (*row)[4];
```

- `row` 指向第0行
- `row + 1` 指向第1行（因为它知道“一行有4个int”）
- `row[i][j]` 可以直接当二维数组用

列指针（其实就是普通的元素指针）

```c
int *p = &a[0][0];
```

此时 `p + n` 指向的元素是：

- 第几行：`row = n / C`
- 第几列：`col = n % C`

结构体

结构体的声明（反正我直接用typedef了）

```
typedef struct {
 int x;
 int y;
} Point; // 相当于将一个匿名结构体绑定给 Point 了

Point p = {0, 0};
```

允许函数以结构体为返回值，此时发生拷贝。

判断傻逼类型注释的技巧

从右向左螺旋读，遇到括号时被“挡住”。 例如对于这个愚蠢至极的数组指针定义：

```
int (*p)[3];
```

这次括号把 `*p` 绑定在一起了：

从 `p` 开始读：

- `(*p)`：说明 **p 是指针**（因为要先解引用）
- `(*p)[3]`：解引用后得到的东西是数组（长度 3）

所以它是**数组指针**：`p` 是指针，指向“3个int的数组”