10 字符串

表示、整行输入与基本操作

本讲学习目标

- 认识字符串
  - 区分终止符、长度与存储容量。
  - 读入带空格的一行文本，整理换行与回车。
- 操作字符串
  - 查用长度、复制、比较、连接和查找函数。
  - 自主实现长度与复制，了解简单替换。
- 动态存储
  - 已学内存申请与释放后，读懂动态字符串副本。

字符串：从字符序列到文本处理

程序 = 数据对象 + 操作 + 控制 + 组合

- 字符串用带结束标记的字符序列表示文本，使程序能将多个字符作为整体处理。
- 字符串操作使程序能够读取、复制、比较、连接和查找文本内容。

1 字符串的表示

认识终止符，区分长度与存储容量

字符数组中的字符串

```
1 char text[8] = "cat";
2 char letter = 'A';
3 char word[] = "A";
```

字符与字符串

letter 保存一个字符，word 保存 A 和终止符。

字符串的结束

字符串是以首个空字符 '\0' 结束的字符序列。

'\0' 值为零，标记结束，不显示为文字。

text 的内容是 c、a、t，随后是 '\0'。

使用条件

数组中必须有可访问的终止符，才能按字符串读取。

本讲用短英文文本，长度按字节计。

字符串长度与数组容量

对象	元素或含义	本例结果
text[8]	c、a、t、\0、0、0、0、0	8个char元素
字符串长度	首个终止符前的字节数	3
数组容量	可用于保存内容的char元素数	8
复制所需最小容量	内容加一个终止符	4

长度描述内容，**容量**描述可用存储。字符串可以短于数组。

空字符串 "" 的长度为0，仍需要一个终止符。

练习：数组中的结束位置

```
1 char s[6] = {'a', '\0', 'b', 'c', '\0', '\0'};
```

已知规则

在首个 '\0' 处结束，长度不含终止符。

写出结果

s 的容量与字符串长度分别是多少?

复制s，目标最少需要几个char元素?

数组后面的 b、c 会成为复制的内容吗?

解析：数组中的结束位置

- 容量为6，字符串长度为1。
- 首个终止符在 s[1]，有效内容只有 a。
- 目标最少需要2个char元素，保存 a 和 '\0'。
- 后面的 b、c 仍是数组元素，位于当前字符串结束位置之后。

终止符决定内容在哪里结束，容量决定存储范围。

可修改数组与字符串字面量

```
1 char a[] = "cat";
2 const char *p = "cat";
3 a[0] = 'b';
4 // 不通过 p 修改字符串字面量
```

可修改数组

a 有自己的字符元素，修改后内容为 bat。

字符串字面量

p 保存字面量的地址，字面量不可修改。

const char * 表示不能通过这个指针修改所指字符。

数组与指针

声明一个指针不会自动分配字符数组，也不会记录容量。

2 一行文本的输入

保留空格，整理行尾并处理剩余输入

带空格的一行输入

```c
#include <stdio.h>
int main(void) {
    char line[32];
    if (fgets(line, sizeof line, stdin)
            == NULL) {
        return 1;
    }
    printf("[%s]", line);
    return 0;
}
```

输入一行

stdin表示标准输入。

fgets读取一行，空格也保存。

最多读31个字符，成功后再补 '\0'。

读到 '\n' 时保存它，并结束本次读取。

检查结果

返回NULL时，不能继续处理本次输入。

EOF表示输入结束；遇到EOF且未读入字符时返回NULL。

与 %s 对比

scanf("%31s", line)跳过开头空白，遇后续空白停止。

本讲读取整行时使用fgets。

行尾字符与输出

输入流给出的字符	数组中的有效内容	printf("[%s]", line)
a、空格、b、LF	a b 后带 \n	右方括号出现在下一行
只有LF	\n	两个括号之间换行
last，随后EOF	last	[last]

换行符（LF） '\n' 使输出换行，**回车符**（CR） '\r' 通常使光标回到行首；它们都不是终止符。

puts还会另加一个换行；printf("%s", line)按已有内容输出。

去掉末尾换行与回车

```
// line 是成功读入的字符串
size_t n = strlen(line);
if (n > 0 && line[n - 1] == '\n') {
    --n;
    line[n] = '\0';
}
if (n > 0 && line[n - 1] == '\r') {
    --n;
    line[n] = '\0';
}
```

只整理行尾

先去末尾LF，再去与其相邻的CR。

a b\r\n：去LF后n由5变4，再去CR变3。

两次终止符分别写入line[4]和line[3]。

边界检查

n > 0 时才能访问 line[n - 1]。

内部空格保留；没有行尾字符则不改内容。

本例也整理单独的末尾CR。

字符串长度

strlen声明在string.h中，长度不含终止符。

size_t 是保存大小的无符号整数类型，输出用%zu。

行尾整理后的字符串

整理前	整理后	长度
a b\n	a b	3
a b\r\n	a b	3
\n	空字符串	0
last	last	4

常见外部行尾有LF和CRLF，文本输入可能做转换；按实际读到的字符处理。

没有读到换行，可能是容量已满，也可能是最后一行到EOF为止。

练习：有限容量的一次读取

```
1 char line[6];
2 if (fgets(line, sizeof line, stdin) == NULL) {
3     return 1;
4 }
```

已知条件

本片段位于main，输入流提供abcdef和一个LF。

写出本次读取后

数组保存的字符与终止符位置。

字符串长度。

输入流中尚未读取的字符。

查用规则

最多读容量减一的字符，成功后补终止符。

解析：有限容量的一次读取

- 数组保存 a、b、c、d、e、'\0'。
- 长度为5，终止符位于line[5]。
- f和LF仍留在输入流，下一次读取会从f开始。
- 本次成功读取，没有越界，但没有读完整行。

数组保存的字符串与输入流剩余内容需要分开追踪。

保留前缀并丢弃本行剩余输入

```c
#include <stdio.h>
#include <string.h>
int main(void) {
    char line[6];
    if (fgets(line, sizeof line, stdin)
            == NULL) {
        return 1;
    }
    if (strchr(line, '\n') == NULL) {
        int ch;
        while ((ch = getchar()) != '\n'
                && ch != EOF) {
        }
    }
    printf("[%s]\n", line);
    return 0;
}
```

本例的策略

数组容量6，只保留最多5个字符。

strchr查找LF，未找到返回NULL。

未读到LF时，继续丢弃至LF或EOF。

已知输入abcdef和LF

保存abcde，丢弃f和LF，输出[abcde]。

getchar与EOF

用int接收字符或EOF，EOF表示没有更多输入。

无末尾LF的最后一行也可在EOF处结束。

数字输入后的剩余换行

```c
// main 中的片段，输入18、LF、Li Ming、LF
int age;
char name[32];
if (scanf("%d", &age) != 1) {
    return 1;
}
if (fgets(name, sizeof name, stdin) == NULL) {
    return 1;
}
```

数字转换

%d读取18，结束在其后的LF之前。

接下来的fgets

读到LF就停止，name保存LF与终止符。

这次没有读取Li Ming。

处理策略

若约定数字占一整行，先丢弃该行剩余输入。

不是无条件多调用一次getchar。

按行输入数字与姓名

```c
#include <stdio.h>
int main(void) {
    int age;
    char name[32];
    if (scanf("%d", &age) != 1) {
        return 1;
    }
    int ch;
    while ((ch = getchar()) != '\n'
            && ch != EOF) {}
    if (fgets(name, sizeof name, stdin)
            == NULL) {
        return 1;
    }
    printf("%d: %s", age, name);
    return 0;
}
```

本例约定

年龄占一行，姓名在下一行；姓名不超过30个字符。

清理当前行

读取至LF或EOF，丢弃该行剩余输入。

姓名输入

输入18和Li Ming，输出18: Li Ming及换行。

后续若需要整理行尾，可沿用前面的处理。

使用边界

不使用fflush(stdin)清理输入。

3 字符串函数与自主实现

区分地址与内容，自主构造长度和复制函数

常见字符串函数

任务	函数	查用条件或结果
求长度	strlen(s)	不包括终止符，返回size_t
复制	strcpy(dst, src)	连同终止符复制，返回dst
比较	strcmp(a, b)	结果为负、零或正
连接	strcat(dst, src)	接到dst末尾，返回dst

声明在string.h中；输入必须是有效字符串。

复制和连接由调用者保证目标可写、容量足够，源与目标不重叠。

复制字符串所需的空间

```
1 #include <string.h>
2 char source[] = "cat";
3 char target[4];
4 strcpy(target, source);
```

复制内容

target得到c、a、t及 '\0'，source保持不变。

目标容量

至少strlen(source) + 1，本例为4。

target[3]用于终止符，少一个元素就不够。

使用条件

两个数组独立，目标可写；strcpy不自动扩大数组。

练习：地址副本与内容副本

```
1 char source[] = "cat";
2 char copy[8];
3 char *alias = source;
4 strcpy(copy, source);
5 alias[0] = 'b';
```

判断内容变化

写出修改后的source与copy内容。

说明alias[0]修改的是哪个数组。

已知条件

strcpy连同终止符复制字符。

copy有独立空间，容量足够。

解析：地址副本与内容副本

访问方式	访问哪个数组	修改前	修改后
source	source数组	cat	bat
alias	source数组（alias → source）	cat	bat
copy	copy数组，独立于source	cat	cat

alias[0]修改source[0]，两种访问方式看到同一次变化。

指针赋值复制地址，strcpy复制字符；独立副本的修改互不影响。

字符串内容的比较

```
1 char a[] = "cat";
2 char b[] = "cat";
3 printf("%d\n", a == &b[0]);
4 printf("%d\n", strcmp(a, b) == 0);
```

两种比较

a与b为两个独立数组，a == &b[0]比较地址，结果为0。

strcmp比较内容，本例相同，第二行输出1。

strcmp结果

零表示内容相同。

负或正表示按字符次序小于或大于。

只依赖结果符号，不假定固定返回-1或1。

本例为短ASCII文本，不介绍中文排序。

字符串连接与目标容量

```
1 char text[8] = "cat";
2 strcat(text, "s");
3 puts(text);
```

连接的位置

s覆盖原终止符，后面再保存一个终止符。

输出cats，连接后长度为4。

容量要求

原长度加追加长度再加一，本例至少5。

text必须已经是有效字符串。

目标容量不足时不可调用strcat。

练习：自主实现长度函数

```
1 size_t my_length(const char s[]);
```

编写要求

自行写出完整函数，不调用strlen。

s为有效字符串，返回终止符前的字节数。

可查语法及size_t含义，不提供循环体。

边界检查

空字符串应返回多少?

"a b"中的空格是否计入长度?

解析：自主实现长度函数

```
1 size_t my_length(const char s[]) {
2     size_t i = 0;
3     while (s[i] != '\0') {
4         ++i;
5     }
6     return i;
7 }
```

遍历与返回

从i = 0开始，逐个检查s[i]，遇终止符停止。

返回i，不把终止符计入长度。

数组下标定位字符，循环按内容决定何时停止

函数封装该处理，供不同字符串调用。

边界与数组参数

空串不进入循环，返回0；"a b"返回3。

数组参数按指针处理，sizeof s不是原数组容量。

练习：自主实现复制函数

```
1 void my_copy(char dst[], const char src[]);
```

编写要求

自行写出完整函数，不调用strcpy。

调用前提

src是有效字符串，dst可写。

两个区域不重叠。

dst容量至少为源长度加一。

结果检查

src不变，dst为相同内容的有效字符串。

说明空串及cat复制到容量4时保存哪些元素。

解析：自主实现复制函数

```
1 void my_copy(char dst[], const char src[]) {
2     size_t i = 0;
3     while (src[i] != '\0') {
4         dst[i] = src[i];
5         ++i;
6     }
7     dst[i] = '\0';
8 }
```

复制内容与终止符

循环复制终止符前的各个字符。

循环结束后，dst[i]保存 '\0'。

边界与调用前提

空串也把dst[0]写成 '\0'。

cat写入dst[0]至dst[3]，最后一个是终止符。

调用者保证容量和不重叠，函数不自动检查。

两个自定义函数的边界检查

源字符串	目标容量	复制后内容	my_length结果
空字符串	4	dst[0]为\0	0
cat	4	c、a、t、\0	3
a b	4	a、空格、b、\0	3

用三组输入检查自己的函数，核对返回值、目标字符与终止符位置；可手算或运行。

说明空串的停止与终止符保存；只检查满足前提的输入，接受符合约定的不同写法。

4 查找与替换

理解匹配位置和首处替换的容量变化

查找字符与子串

```
1 char text[] = "red cat";
2 char *space = strchr(text, ' ');
3 char *match = strstr(text, "cat");
```

两个接口声明在string.h中

strchr查找字符首次出现的位置。

子串是原字符串中连续的一段字符。

strstr查找子串首次出现的位置。

返回结果

本例space指向text[3]，match指向text[4]。

未找到返回NULL，使用前检查。

返回的是原串中的指针，没有创建新字符串。

练习：查找返回的位置

```
1 char text[] = "a cat sleeps";
2 char *p = strstr(text, "cat");
3 char *q = strstr(text, "dog");
4 if (p != NULL) {
5     puts(p);
6 }
```

判断匹配位置

写出p指向的下标，以及puts(p)的输出。

q是否匹配成功？为什么不能直接用puts(q)?

查用规则

strstr返回首处匹配位置，未找到返回NULL。

解析：查找返回的位置

- p指向text[2]，这里是c。
- puts从p的位置输出到终止符，打印cat sleeps并换行。
- 没有dog，q为NULL，不能作为字符串传给puts。
- p仍指向text内部，不拥有新数组，也不能用于free。

匹配位置可以作为后续读取或修改的起点。

替换一个字符

```
1 char text[] = "red cat";
2 char *p = strchr(text, ' ');
3 if (p != NULL) {
4     *p = '_';
5 }
6 puts(text);
```

替换位置

找到空格后，通过*p修改同一数组元素。

输出red_cat，长度保持7。

使用条件

text可修改，p不为NULL。

本例只替换第一个匹配字符。

标准C库没有通用字符串替换函数，需要自行组合操作。

首处子串替换的组成

部分	原内容	替换后保存的内容
匹配之前	red及空格	red及空格
首处匹配	cat	kitten
匹配之后	空格及sleeps	空格及sleeps

结果为red kitten sleeps：新长度14 − 3 + 6 = 17，最少需要18个char元素。

用独立目标空间依次保存前段、新内容和后段，最后保留终止符。

首处子串替换的保存过程

```
1  // source为"red cat sleeps", old为"cat"
2  // replacement为"kitten", result容量32
3  const char *match = strstr(source, old);
4  if (match == NULL) {
5      strcpy(result, source);
6  } else {
7      size_t prefix = (size_t)(match - source);
8      size_t i;
9      for (i = 0; i < prefix; ++i) {
10         result[i] = source[i];
11     }
12     strcpy(result + i, replacement);
13     strcpy(result + i + strlen(replacement),
14            match + strlen(old));
15 }
```

结果的组成

match指向source[4]，相减得到前段长度4。

循环保存前4个字符，结束时i为4。

从result[4]复制kitten，长度6。

后段从source[7]读入，接到result[10]。

最后一个strcpy保存后段及终止符。

限定条件

old非空，仅替换首处；未找到则复制原串。

result独立可写，容量32足够保存本例。

新长度可能变化，不能直接假定原容量够用。

学习要求

只读懂机制，不要求自主实现通用替换。

5 动态字符串

先学PF-09，再迁移申请、复制、使用与释放

动态字符串的申请、复制与释放

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(void) {
    char source[] = "cat";
    char *copy = malloc(strlen(source) + 1);
    if (copy == NULL) {
        puts("Allocation failed");
        return 1;
    }
    strcpy(copy, source);
    copy[0] = 'b';
    printf("%s %s\n", source, copy);
    free(copy);
    return 0;
}
```

先修条件

已学PF-09的malloc、NULL检查与free。

申请空间

源长度3，申请4个char元素，包含终止符。

失败输出提示并结束，成功后才复制。

独立副本

strcpy建立内容，修改copy后输出cat bat。

释放

最后使用后free(copy)，不释放source。

此后不再访问copy指向的存储。

固定数组与动态字符串

比较项	固定数组	动态字符串
字符存储	声明确定数组容量	运行时成功申请
字符串结束	首个\0	首个\0
复制容量	至少为源长度加一	申请源长度加一
结束存储	按数组对象的生存期	按约定调用free

两种存储遵循相同的终止符规则；容量与对象生存期需要另外保证。

char指针本身保存地址，不自动保存长度或负责释放。

本讲总结

- 检查表示
  - 有效字符串以 '\0' 结束，长度不含终止符。
- 检查输入
  - 整行用fgets，检查结果，整理行尾并关注剩余输入。
- 检查操作
  - 复制含终止符，目标容量至少为源长度加一。
  - 区分地址副本与内容副本，比较内容用strcmp。
- 检查存储
  - 动态副本需成功申请，最后使用后释放。
  - 能自主写出长度与复制的循环及结束条件。

教材阅读要求

本讲内容	C Primer Plus第六版	阅读要求
表示、数组与指针	§11.1	重点阅读
输入输出与行尾	§11.2 ~ 11.3	重点阅读fgets及输出
长度、复制、比较、连接	§11.5.1 ~ 11.5.5	重点接口按本讲范围
字符与子串查找	§11.5.7	查阅strchr、strstr
动态存储	§12.4及PF-09	学完内存章节后复习

长度与复制自主实现、首处替换为本讲配套材料；暂不要求命令行参数及完整函数族。