12 文件

流、文本、二进制与固定记录集合

本讲学习目标

- 连接文件
  - 识别流、打开模式和关闭责任。
- 读写数据
  - 读写文本行与简单格式数据，按返回值判断结果。
  - 读写二进制对象，用固定记录大小定位。
- 保存记录集合
  - 加载固定结构体数组，新增、查找、修改后保存。

文件：从运行时处理到持久保存

程序 = 数据对象 + 操作 + 控制 + 组合

- 文件使程序能够将数据保存到外部存储，并在后续运行中重新读取。
- **持久保存与读取**使数据不再局限于当前运行，扩展了程序管理和复用数据的能力。

两次运行之间怎样保留成绩

- 已有记录
  - 固定结构体数组中有101 Li Ming 80和102 Chen Yu 60。
- 本次操作
  - 把102的成绩改为75，然后退出程序。
  - 第二次运行需要再次取得这两条记录。
- 本讲问题
  - 数组赋值改变了当前内存，数据怎样留到下一次运行?
  - 文件保存写出的数据；本讲建立内存与文件之间的读写联系。

1 流与打开关闭

连接文件，并明确读写与关闭责任

同一写出操作可以连接不同去向

```c
/* printf默认写向stdout */
fprintf(stdout, "%d\n", 80);
FILE *fp = fopen("scores.txt", "w");
if (fp == NULL) return 1;
int wrote = fprintf(fp, "%d\n", 80);
int closed = fclose(fp);
if (wrote < 0 || closed == EOF) return 1;
```

两个去向

stdout是标准输出流，通常连接屏幕。

fopen成功后，fp指向库维护的文件流。

共同联系

流连接读写操作与数据来源、去向。

FILE声明于<stdio.h>，不访问内部成员。

比较代码

两次fprintf的格式和值相同，哪个参数决定写到哪里?

打开失败后，还能使用fp吗?

打开成功后才能读写

```c
FILE *fp = fopen("scores.txt", "r");
if (fp == NULL) {
    puts("open failed");
    return 1;
}
/* 在这里读取 */
if (fclose(fp) == EOF) return 1;
```

比较回收

fprintf的第一个参数选择流，屏幕和文件是不同去向。

stdin通常连接键盘，stderr用于错误输出。

打开

fopen返回FILE *；失败返回NULL。

工作目录是相对文件名的查找起点。

读写

只向成功打开且模式允许的流发起操作。

关闭

fclose提交待写出的数据，释放流资源。

返回0表示成功，EOF表示失败；关闭后不再使用fp。

打开模式决定保留还是清空

模式	用途	文件不存在	原内容
r / rb	读取	打开失败	保留
w / wb	从头写入	创建	清空
a / ab	追加写入	创建	保留，写到末尾

- w成功打开时就清空原文件，不等到第一次写入。
- 最小主线分开打开读流和写流。

练习：为三种操作选择模式

- 独立判断
  - 读取已有成绩文本，应选什么模式?
  - 保留日志并增加一行，应选什么模式?
  - 重写整个二进制记录集合，应选什么模式?
- 说明依据
  - 指出哪种选择会清空已有内容，以及何时清空。

解析：模式与数据保留

- 三个选择
  - 成绩文本：r；保留日志追加：a；重写二进制集合：wb。
- 关键区别
  - w或wb成功打开时清空原内容。
  - a或ab把后续写入放到文件末尾；r或rb不创建缺失文件。
- 打开失败
  - 返回NULL时停止本次操作，不继续读写或关闭这个空指针。

2 文本读写

根据读取结果处理字符与字段

把字符串写成文本行

```c
FILE *fp = fopen("names.txt", "w");
if (fp == NULL) return 1;
if (fputs("Li Ming\nChen Yu\n", fp)
    == EOF) {
    fclose(fp);
    return 1;
}
if (fclose(fp) == EOF) return 1;
```

写入字符串

fputs写出字符串字符，不写末尾的\0。

它不会自动添加换行；这里字符串自带\n。

检查结果

fputs失败返回EOF；成功返回非负值。

失败分支也关闭已打开的流，随后停止操作。

用fgets读取带空格的文本

```
1  FILE *fp = fopen("names.txt", "r");
2  if (fp == NULL) return 1;
3  char line[20];
4  while (fgets(line, sizeof line, fp)
5          != NULL) {
6      fputs(line, stdout);
7  }
8  int failed = ferror(fp);
9  int closed = fclose(fp);
10 if (failed || closed == EOF) return 1;
```

每次读取

最多取sizeof line - 1个字符，并补\0。

换行也存入缓冲区；空间足够时取得整行。

空格与长度

空格作为普通字符保留。

长行可能分成多次读取；本例缓冲区足够容纳每行。

循环依据

返回非NULL才使用line；失败或无字符可读时返回NULL。

ferror非0表示流发生读取错误。

EOF是表示结束等情况的常量，不是文件末尾的字符。

读取结果与结束标志的变化

次序	取得内容	返回结果	feof
第1次	Li Ming及换行	非NULL	0
第2次	Chen Yu及换行	非NULL	0
第3次	没有字符	NULL	非0

- 本例两行均含换行、读取无错误。流记录位置和状态，第三次才置结束标志。
- while (!feof(fp))不能保证读到字符。循环依据fgets结果，NULL后查ferror。

练习：自主复制文本内容

```c
int copy_lines(FILE *in, FILE *out);
/* 可使用char line[20] */
```

接口与前提

in和out均已成功打开，分别允许读与写。

本函数不负责打开或关闭，调用者检查关闭结果。

独立编写

循环读取并原样写出；读或写失败返回-1，正常结束返回0。

核对带空格文本、末行无换行和空文件。

解析：按成功读取的内容写出

```c
int copy_lines(FILE *in, FILE *out) {
    char line[20];
    while (fgets(line, sizeof line, in)) {
        if (fputs(line, out) == EOF)
            return -1;
    }
    return ferror(in) ? -1 : 0;
}
```

循环与收束

只有fgets成功取得字符，才调用fputs。

写失败立即返回；读循环结束后检查ferror。

三项核对

带空格文本保持空格；末行无换行也写出。

空文件不进入循环，正常返回0。

按约定格式读写两个整数

```c
/* 已成功打开的写流out */
fprintf(out, "%d %d\n", 101, 80);
/* 另行打开的读流in */
int id, score;
int rc = fscanf(in, "%d %d",
                &id, &score);
if (rc == 2) {
    printf("%d %d\n", id, score);
}
```

文本格式

fprintf把整数转换成可读字符；一行约定为学号与成绩。

fscanf解析字符，返回成功赋值数。

完整记录

返回2才能把两个值作为完整记录使用。

本例不读取姓名；带空格姓名使用前面的fgets路线。

练习：第二条记录是否完整

```c
int id = 0, score = 0;
int a = fscanf(fp, "%d %d",
               &id, &score);
int b = fscanf(fp, "%d %d",
               &id, &score);
```

给定文件

文件第一行是101 80；第二行只有102。

独立判断

连续两次fscanf分别返回什么？

第二次能否把id与score一起作为新记录？

说明依据

区分成功赋值的数量与流结束；不要使用旧score补齐记录。

解析：一个赋值不构成两个字段

- 两次结果
  - 第一次返回2，取得101和80。
  - 第二次返回1，只有id变为102；score仍是上次的80。
- 处理决定
  - 返回1表示记录缺少字段，停止并报告格式不完整。
  - 返回0表示当前字符不符合转换；读取错误需查ferror。
  - 没有任何赋值且遇到结束时才可能返回EOF。

3 二进制读写

选择表示，并使用完整项目数

80写成字符还是对象字节

```c
int x = 80;
/* text以w、binary以wb成功打开 */
fprintf(text, "%d", x);
fwrite(&x, sizeof x, 1, binary);
```

文本写法

fprintf将80转换为字符8、0，文件保存这些字符。

对象字节写法

fwrite复制x所占的sizeof x个字节。

wb避免文本模式对换行等作转换。

可用函数

fscanf解析字符，fread把字节读入对象。

独立选择

两个文件分别应用fscanf还是fread读回整数？

说明读回函数需要恢复哪种表示。

解析：读回方式与文件表示匹配

保存内容	写入	匹配的读回方式
字符8、0	fprintf转换80为字符	fscanf解析为整数80
int对象字节	fwrite复制sizeof x字节	fread读入匹配的int

- 文本也以字节保存，区别在转换或对象复制。后缀用于命名，协议由模式、函数和约定确定。
- 原样对象字节仅供相同程序、类型和环境读回。

按对象大小写入数组

```c
int data[3] = {10, 20, 30};
FILE *fp = fopen("values.bin", "wb");
if (fp == NULL) return 1;
size_t n = fwrite(data, sizeof data[0],
                  3, fp);
int closed = fclose(fp);
if (n != 3 || closed == EOF) return 1;
```

大小与数量

size_t是表示大小和数量的无符号整数类型，输出用%zu。

三个参数职责

data指向待写数据；sizeof data[0]是每项字节数。

3是项目数；返回值也是成功写出的项目数。

成功条件

这里必须写出3项，且fclose成功。

先取得两个结果，再判断，确保总会关闭流。

按实际读回数量使用数组

```
1 int data[5];
2 FILE *fp = fopen("values.bin", "rb");
3 if (fp == NULL) return 1;
4 size_t got = fread(data, sizeof data[0],
5                    5, fp);
6 int failed = ferror(fp);
7 int closed = fclose(fp);
8 if (failed || closed == EOF) return 1;
9 for (size_t i = 0; i < got; ++i)
10     printf("%d\n", data[i]);
```

读取请求

最多读5项，每项sizeof data[0]字节。

文件由上页写出，实际只有3项，got为3。

使用边界

只使用下标小于got的元素。

少于请求数可能因结束或错误，需要检查ferror。

练习：请求数不等于有效数

```
1 int data[5];
2 size_t got = fread(data, sizeof data[0],
3                    5, fp);
4 for (size_t i = 0; i < 5; ++i)
5     printf("%d\n", data[i]);
```

给定条件

文件只有完整的10、20、30三个int，读取时请求5项。

独立修正

下方循环有什么问题？改写循环边界。

说明应检查哪种流状态，以及为什么不能直接断言读取失败。

解析：只访问成功读回的三项

```
for (size_t i = 0; i < got; ++i)
    printf("%d\n", data[i]);
if (ferror(fp)) {
    /* 报告读取失败，再关闭流 */
}
```

实际结果

got为3；只允许使用data[0]到data[2]。

修正与判断

循环使用i < got，输出10、20、30。

ferror非0表示读取错误；本例正常结束。

返回值单位

sizeof data[0]是项大小，got是完整项目数，不是总字节数。

4 文件定位

固定大小联系下标、偏移和目标记录

连续读取与文件位置

```c
/* fp以rb打开，文件含10、20、30 */
int first, second;
if (fread(&first, sizeof first, 1, fp)
    != 1) return 1;
if (fread(&second, sizeof second, 1, fp)
    != 1) return 1;
```

连续读取

从开头读出10后，下一次取得20。

文件位置表示后续读写发生的位置。

位置对应

三项起点：0、sizeof(int)、2*sizeof(int)

连续成功读取后，位置移动到下一项开头。

定位目标

数组下标给出第几项，文件偏移给出距开头的字节数。

已知固定大小，就能直接定位目标项。

用fseek定位固定大小的项目

```
1 long offset = (long)sizeof(int);
2 if (fseek(fp, offset, SEEK_SET) != 0)
3     return 1;
4 int value;
5 if (fread(&value, sizeof value, 1, fp)
6     != 1) return 1;
7 printf("%d\n", value);
```

定位公式

下标k从0开始；从开头的偏移是k乘每项字节数。

SEEK_SET表示从开头算；成功返回0。

成功后读取

fseek成功只说明位置已调整，不保证那里有完整数据。

fread返回1才取得目标int，本例得到20。

查询位置与重新从开头读取

```c
long pos = ftell(fp);
if (pos == -1L) return 1;
rewind(fp);
int first;
if (fread(&first, sizeof first, 1, fp)
    != 1) return 1;
/* first为10 */
```

ftell

成功返回当前位置；失败返回-1L。

本讲二进制文件的位置以从开头的字节偏移理解。

rewind

回到文件开头，并清除错误与结束标志。

它没有返回值，重新读取仍检查fread。

文本边界

文本模式的位置值不能普遍当作字符数做任意偏移计算。

练习：直接读取第三条记录

```c
struct Student {
    int id;
    char name[20];
    int score;
};
struct Student one;
/* 补全定位和读取，失败返回-1 */
```

给定条件

fp以rb打开，文件有3条完整Student记录。

下标从0开始，失败交调用者处理并关闭流。

独立编写

读取第三条到one，写出偏移表达式。

检查fseek和fread两个调用结果。

解析：第三条的下标是2

```
long offset = 2L * (long)sizeof one;
if (fseek(fp, offset, SEEK_SET) != 0)
    return -1;
if (fread(&one, sizeof one, 1, fp) != 1)
    return -1;
```

位置与读取

第三条之前有2条，偏移为2乘一条记录大小。

用sizeof one取得包含填充在内的实际大小。

边界

本讲只处理同环境生成的小型固定记录文件。

变长文本行不能按此公式定位。

5 记录集合管理

加载、内存处理与保存连接两次运行

用固定数组管理学生记录

```c
#define CAP 4
struct Student {
    int id;
    char name[20];
    int score;
};
struct Student data[CAP];
size_t n = 0;
```

已有数据结构

每条记录含学号、固定姓名数组和成绩。

CAP是容量，n是**有效记录数**。

本例0≤n≤4。

贯穿记录

101 Li Ming 80与102 Chen Yu 60，修改目标为102的成绩75。

文件约定

students.bin只保存n条有效记录。

本讲只读取自己的同环境程序生成的文件。

先加载，再处理，最后保存

阶段	数据所在位置	本讲操作
加载	文件 → data	取得完整记录和有效数n
处理	data	新增、按学号查找、修改成绩
保存	data → 文件	重写n条有效记录

- 加载成功后才处理；修改内存不会自动更新文件。
- 保存失败须报告，不能声称数据已保存；数组组织记录，赋值修改数据，分支检查结果，循环逐项处理，函数组织加载与保存。

先用接口连接加载、处理与保存

接口	返回结果	数据影响
load(path, data, &n)	返回0，取得记录和n	文件读入内存
find_id(data, n, id)	返回下标，未找到为-1	查询内存
add(data, &n, one)	返回0，增加n	改变内存
save(path, data, n)	返回0，写出n条	内存写入文件

- load成功后才能处理，处理后检查save结果，再重新读取核对。
- 此处先使用给定接口，后续逐个解释实现。

按学号找到内存中的记录

```c
int find_id(const struct Student data[],
            size_t n, int id) {
    for (size_t i = 0; i < n; ++i)
        if (data[i].id == id)
            return (int)i;
    return -1;
}
/* 调用者 */
int i = find_id(data, n, 102);
if (i >= 0) data[i].score = 75;
```

复用数组循环

只遍历n条有效记录，比较id。

返回下标，没找到返回-1。

修改对象

找到后改变data[i].score，只改变内存。

下一阶段仍需调用保存函数。

新增记录前检查容量与学号

```c
int add(struct Student data[], size_t *n,
        struct Student one) {
    if (*n == CAP) return -1;
    if (find_id(data, *n, one.id) >= 0)
        return -1;
    data[*n] = one;
    ++*n;
    return 0;
}
```

两个检查

集合已满时拒绝新增；已有同学号时也拒绝。

本函数只操作内存，返回0成功、-1拒绝。

实际新增

整体赋值复制固定成员，再增加有效数。

沿用第11讲结构体复制，不分配动态成员。

数量变化

空集合n为0；成功新增101后为1，再新增102后为2。

保存时写出有效记录数

```
1 FILE *fp = fopen(path, "wb");
2 if (fp == NULL) return -1;
3 size_t wrote = fwrite(data, sizeof data[0],
4                       n, fp);
5 int closed = fclose(fp);
6 return wrote == n && closed == 0 ? 0 : -1;
```

写入集合

wb重写文件；fwrite的数量参数是n。

不能使用CAP，否则会保存空余槽位。

结果与关闭

写出n条且关闭成功，函数才返回0。

失败时不声称保存成功；原文件也可能已被清空或部分改写。

加载入口区分缺失与其他失败

```c
/* load内；包含<stdio.h>和<errno.h> */
FILE *fp = fopen(path, "rb");
if (fp == NULL) {
    if (errno == ENOENT) {
        *n = 0;
        return 0;
    }
    return -1;
}
size_t count = 0;
```

缺失文件

errno是运行库提供的错误编号；ENOENT表示路径不存在。

本例固定文件位于现有工作目录，首次缺失可从空集合开始。

其他失败

权限或其他打开失败返回-1，调用者停止。

不能把所有NULL都当作空集合，否则后续保存可能覆盖数据。

环境约定

errno行为按本课程Windows运行库核验，不宣称所有C实现完全相同。

完整记录经过检查后进入数组

```
1  /* load内，fp已打开，count为0 */
2  for (;;) {
3      struct Student one;
4      size_t got = fread(&one, 1,
5                         sizeof one, fp);
6      if (got == 0 && !ferror(fp)) break;
7      if (got != sizeof one || ferror(fp)
8          || count == CAP) {
9          fclose(fp);
10         return -1;
11     }
12     data[count++] = one;
13 }
```

本次读取

项大小为1，got按字节计数，完整一条应为sizeof one。

三个分支

0字节且无错误：正常结束循环。

不完整、出错或已满：关闭并返回失败。

完整且有空间：放入data，再增加count。

本例核对

两条完整记录使count从0变为2。

加载成功后交回有效记录数

```
1 /* load内，上页循环正常结束之后 */
2 if (fclose(fp) == EOF) return -1;
3 *n = count;
4 return 0;
```

正常完成

两条记录已进入data，count为2。

关闭成功后，*n取得2，函数返回0。

失败处理

读取或关闭失败时，调用者停止处理。

部分加载的数组不能作为本次成功结果使用。

主线回收

load取得记录和有效数，内存处理随后使用这两个结果。

首次运行：加载空集合并新增

```
const char *path = "students.bin";
struct Student data[CAP]; size_t n = 0;
if (load(path, data, &n) != 0) return 1;
if (n == 0) {
    struct Student a = {101, "Li Ming", 80};
    struct Student b = {102, "Chen Yu", 60};
    if (add(data, &n, a) != 0
        || add(data, &n, b) != 0) return 1;
}
```

演示前提

工作目录存在，students.bin首次缺失。

load按本例约定成功得到n为0。

新增结果

两次add成功后，n为2，数组中有两条记录。

新增失败立即停止，尚未保存文件。

代码连续性

下一页沿用path、data和n继续运行。

继续运行：修改、保存并重新读取

```
int i = find_id(data, n, 102);
if (i >= 0) data[i].score = 75;
if (save(path, data, n) != 0) return 1;
if (load(path, data, &n) != 0) return 1;
struct Student one;
if (read_at(path, n, 1, &one) != 0)
    return 1;
printf("n=%zu\n%d %s %d\n", n,
       one.id, one.name, one.score);
```

保存与核对

修改102的成绩，保存2条，再重新加载。

read_at沿用定位规则，读取下标1的记录。

可核对结果

输出n=2以及102 Chen Yu 75。

再次运行仍从文件取得已保存的新成绩。

运行材料

完整函数与调用见examples/student_files.c。

练习：补全修改与保存流程

```c
const char *path = "students.bin";
struct Student data[CAP]; size_t n = 0;
/* 补全：加载、修改102、保存 */
```

给定初态与支持

文件有101 Li Ming 80和102 Chen Yu 60。

已给load、find_id和save接口，CAP为4。

独立编写

加载成功后将102的成绩改为75，再保存；失败返回1。

说明保存数量选n还是CAP。

状态判断

若只修改后退出，下次加载会得到什么成绩?

保存成功后，下次加载又会得到什么成绩?

解析：成功加载和保存连接两次运行

```
1 if (load(path, data, &n) != 0) return 1;
2 int i = find_id(data, n, 102);
3 if (i >= 0) data[i].score = 75;
4 if (save(path, data, n) != 0) return 1;
```

顺序与数量

加载成功后才查找修改，保存失败停止。

写出n为2条，CAP为4仅表示数组容量。

两次运行

未保存时，文件成绩仍为60，下次读到60。

保存成功后，下次读到75。

解释依据

赋值改变内存，save建立内存到文件的联系。

文件成绩与内存成绩分别变化

操作完成后	文件中102成绩	内存中102成绩
load成功	60	60
数组赋值	60	75
save成功	75	75
再次load成功	75	75

- load读取文件，赋值改变内存，save成功后下次取得75。
- 保存失败时，wb可能已清空或部分改写文件，不能声称保存成功或保证回滚。

用边界输入核对集合管理

- 有效数与容量
  - 空文件加载得到0；4条完整记录可以装满数组。
  - 第5条被拒绝，已有4条时新增也被拒绝。
- 记录与失败
  - 重复学号新增被拒绝；不存在的学号找不到。
  - 最后一条字节不完整时加载失败，不继续保存。
- 定位读取
  - 合法下标才能读一条；超出有效数先拒绝。
  - fseek成功后仍需检查fread与fclose。

本讲总结

- 流与操作
  - 流连接数据来源、去向；成功打开后才能按模式操作。
  - 读写推进位置，流记录结束和错误状态，最后检查关闭。
- 表示与结果
  - 文本转换字符，对象读写复制字节，读回方式须匹配。
  - 按返回字段数或项目数使用数据，读取失败时不使用新结果。
  - 固定记录按下标乘大小定位，定位后仍检查读取。
- 两次运行之间
  - 加载取得内存记录，赋值改变内存，保存成功后文件取得新内容。
  - 容量与有效数分别判断，写出有效记录数。

教材对应与阅读要求

本讲内容	C Primer Plus第13章	要求
流、打开与关闭	13.1 ~ 13.2	理解模式及失败处理
文本读写	13.4	使用fgets/fputs与简单格式
定位	13.5	fseek；识别ftell/rewind
二进制与状态	13.7	fread/fwrite与返回值

- 最小应用以本讲源码与任务为准。
- 不要求更新模式、复杂文件系统或通用序列化。