# Lec 14 File I/O in C

## 流（Stream）和缓冲（Buffering）

### 流的概念

- **定义**：流是程序与I/O设备之间的抽象接口，可以是输入流（如从键盘读取）或输出流（如向屏幕输出）。
- **作用**：通过流，程序无需直接操作底层硬件，而是与一个统一的接口交互。

### 缓冲机制

- 输入缓冲：
  - 数据从输入设备（如键盘）进入缓冲区。
  - 用户输入数据后，只有按下Enter键，缓冲区的内容才会释放到流中，供程序读取。
- 输出缓冲：
  - 程序输出的数据先存储在缓冲区中。
  - 当程序提交一个换行符（`\n`）或程序结束时，缓冲区的内容才会释放到输出设备（如显示器）。
- 缓冲的作用：
  - 解耦生产者（数据源）和消费者（数据处理者），避免频繁的直接I/O操作。
  - 提高I/O效率，减少对底层设备的直接访问。

## 输入缓冲示例

### 函数：`getchar`

- **功能**：从标准输入（`stdin`，通常是键盘）读取一个ASCII字符。
- **工作原理**：每次调用读取缓冲区中的一个字符。

### 示例1

- 代码：

```
char in1 = getchar();
char in2 = getchar();
char in3 = getchar();
printf("%c%c%c\n", in1, in2, in3);
```

**用户输入**：`ABCDe` 并按Enter。

**结果**：

- `in1 = 'A', in2 = 'B', in3 = 'C'`
- `'D', 'e'` 和 `Enter` 键仍留在缓冲区中，未被读取。

## 示例2

- 用户分次输入：
  - 输入 A 并按Enter，`in1 = 'A'`。
  - 输入 B 并按Enter，`in2 = 'B'`。
  - 输入 C 并按Enter，`in3 = 'C'`。
- **特点**：`getchar()` 会阻塞程序，等待用户输入并按Enter。

# 输出缓冲

## 函数：`putchar`

- **功能**：向标准输出（`stdout`，通常是显示器）写入一个ASCII字符。
- **工作原理**：字符先进入输出缓冲区，等待刷新。

## 输出缓冲的行为

- 数据存储在缓冲区中，直到遇到换行符（\n）或程序结束，才显示在屏幕上。

## 示例

- **代码**：

```
putchar('a');
sleep(5); // 暂停5秒
putchar('b');
putchar('\n');
```

- 问题：你会看到什么?
  1. 5秒后显示 `ab`。
  2. `a`，5秒后显示`b`
  3. 段错误。
- **答案**：1：在得到`\n`的字符之后一并输出。
- 输出缓冲可能导致显示延迟，刷新时机（如`\n`或程序结束）很重要。

# 基本I/O函数

**创建I/O流**

- `fopen`：打开/创建文件，返回FILE*指针。
- `fclose`：关闭文件，释放资源。

**按字符I/O**

- `fgetc`：从流中读取一个字符。

- `fputc`：向流中写入一个字符。
- `getchar`：从键盘读取字符。
- `putchar`：向屏幕写入字符。

**按行I/O**

- `fgets`：从流中读取一行。
- `fputs`：向流中写入一行。

**格式化I/O**

- `fprintf`：向流中写入格式化字符串。
- `fscanf`：从流中读取格式化数据。

# 文件的基本概念

- **文件**：存储在设备上的ASCII字符序列。
- **流**：每个文件关联一个流（输入、输出或双向）。
- **文件指针**：用`FILE*`类型声明，用于操作文件。（`FILE`类型在`<stdio.h>`库中定义，因此要先`include`）

```
FILE *infile;
```

- **文件操作步骤**：
  1. 打开文件（`fopen`）。
  2. 读写操作。
  3. 关闭文件（`fclose`）。

# 创建和关闭I/O流

## fopen

- **语法**：`FILE* fopen(char* filename, char* mode)`
- 参数：
  - `filename`：文件名。
  - `mode`：
    - "r"：读
    - "w"：写（从头开始，清空文件）
    - "a"：追加（从末尾写入）
- 返回值：
  - 成功：`FILE*`指针
  - 失败：`NULL`

## fclose

- **语法**：`int fclose(FILE* stream)`
- **参数**：文件指针
- 返回值：
  - 成功：`0`
  - 失败：`EOF`（通常为-1）

## 示例

```
FILE *myfile = fopen("test.txt", "w");
if (myfile == NULL) {
    printf("Cannot open file for write.\n");
    return -1;
}
fclose(myfile);
return 0;
```

# 按字符I/O

## `fgetc`

- **语法**：`int fgetc(FILE* stream)`
- **功能**：读取一个字符并前进到下一个
- 返回值：
  - 成功：当前字符
  - 失败：`EOF`

## `fputc`

- **语法**：`int fputc(int character, FILE* stream)`
- **功能**：写入一个字符
- 返回值：
  - 成功：写入的字符
  - 失败：`EOF`

## 示例：文件复制（只复制数字）

```
char c;
FILE *fp1;
FILE *fp2;
if((fp1=fopen("original.txt", "r")) == NULL){
  printf("Unable to open a file.\n");
```

```
  return -1;
}
if((fp2=fopen("modified.txt", "w")) == NULL){
  printf("Unable to open a file.\n");
  return -1;
}

do {
    c = fgetc(fp1);
    if (c >= '0' && c <= '9') fputc(c, fp2);
} while (c != EOF);

fclose(fp1);
fclose(fp2);
```

# 按行I/O

## fgets

- **语法**：`char* fgets(char* string, int num, FILE* stream)`
- 参数：
  - `string`：目标数组
  - `num`：最大读取字符数（包括\0）
  - `stream`：输入流
- 返回值：
  - 成功：指向`string`的指针
  - 失败：`NULL`
- **特点**：读取直到`num-1`个字符、换行符或文件末尾

## fputs

- **语法**：`int fputs(const char* string, FILE* stream)`
- **功能**：写入字符串
- 返回值：
  - 成功：非负值
  - 失败：`EOF`

## fgets vs scanf

- **fgets**：读取整行，直到换行符。
- **scanf**：读取直到空白字符（空格、换行等）。

## 示例：缓冲区剩余问题

```c
#define BUF_SIZE 6
char buf1[BUF_SIZE], buf2[BUF_SIZE];
printf("Enter 4 digits (****): ");
fgets(buf1, BUF_SIZE, stdin); // 输入"12345", buf1="1234", "5\n"留在缓冲区
printf("Enter 4 digits (****): ");
fgets(buf2, BUF_SIZE, stdin); // buf2="5\n"
```

# 格式化I/O

## fprintf

- **语法**：`int fprintf(FILE* stream, const char* format, ...)`
- 参数：
  - `stream`：输出流。
  - `format`：格式字符串（如`%d, %s`）。
  - 其他：替换格式说明符的变量。
- 返回值：
  - 成功：写入字符数。
  - 失败：负数。

## fscanf

- **语法**：`int fscanf(FILE* stream, const char* format, ...)`
- 参数：
  - `stream`：输入流。
  - `format`：格式字符串。
  - 其他：存储数据的指针。
- 返回值：
  - 成功：读取的项目数。
  - 失败：`EOF`

## 示例：数据格式转换

- 输入文件`data.txt`：

```
4311 Alice 3.42
1133 Bob 4.0
```

- 代码：

```c
FILE *fp_in = fopen("data.txt", "r");
FILE *fp_out = fopen("swapped.txt", "w");
int uid; char name[20]; double gpa;
while (fscanf(fp_in, "%d %s %lf", &uid, name, &gpa) != EOF)
    fprintf(fp_out, "%s %d %lf\n", name, uid, gpa);
fclose(fp_in);
fclose(fp_out);
```

- 输出文件`sqapped.txt`：

```
Alice 4311 3.42
Bob 1133 4.0
```

`fprintf`和`fscanf`支持格式化数据处理，常用于结构化文件操作。

## 总结

- **流和缓冲**：抽象接口和效率优化机制。
- **基本函数**：`fopen, fclose, fgetc, fputc, fgets, fputs, fprintf, fscanf`等。
- **操作步骤**：打开、读写、关闭。
- **应用示例**：字符处理、行处理、格式化数据、矩阵转置等。