# 5. File Input/Output

## 5.1 Why File I/O?

So far, all our programs lose their data when they terminate. **File I/O** allows programs to:

- Save data permanently
- Read data from external sources
- Create logs and reports
- Share data between programs

**Python vs C File Operations:**

| Python | C |
|---|---|
| `f = open("file.txt", "r")` | `FILE *f = fopen("file.txt", "r");` |
| `f.write("text")` | `fprintf(f, "text");` |
| `content = f.read()` | `fscanf(f, "%s", buffer);` |
| `f.close()` | `fclose(f);` |

## 5.2 File Pointer

In C, files are accessed through **file pointers** of type `FILE*`:

```
FILE *filePointer;
```

The `FILE` type is defined in `<stdio.h>`.

## 5.3 Opening Files - fopen()

**Function Signature:**

```
FILE *fopen(const char *filename, const char *mode);
```

**File Opening Modes:**

| Mode | Description | If File Exists | If File Doesn't Exist |
|---|---|---|---|
| `"r"` | Read only | Opens file | Returns NULL |

| Mode | Description | If File Exists | If File Doesn't Exist |
|---|---|---|---|
| `"w"` | Write only | Overwrites content | Creates new file |
| `"a"` | Append | Appends to end | Creates new file |
| `"r+"` | Read and write | Opens file | Returns NULL |
| `"w+"` | Read and write | Overwrites content | Creates new file |
| `"a+"` | Read and append | Opens file | Creates new file |
| `"rb"` | Read binary | Opens file | Returns NULL |
| `"wb"` | Write binary | Overwrites content | Creates new file |
| `"ab"` | Append binary | Appends to end | Creates new file |

**Example:**

```
FILE *file;

// Open file for reading
file = fopen("data.txt", "r");

// Always check if file opened successfully
if (file == NULL) {
    printf("Error: Could not open file!\n");
    return 1;
}

// ... file operations ...

fclose(file);
```

# 5.4 Closing Files - fclose()

**Always close files after use** to:

- Free system resources
- Ensure all data is written to disk
- Prevent data corruption

```
int fclose(FILE *filePointer);
```

**Returns:**

- `0` on success
- `EOF` on error

**Example:**

```
FILE *file = fopen("data.txt", "r");
if (file != NULL) {
    // ... operations ...
    fclose(file);
}
```

# 5.5 Reading from Text Files

## 5.5.1 fscanf() - Formatted Input

Similar to `scanf()`, but reads from a file:

```
int fscanf(FILE *stream, const char *format, ...);
```

**Example:**

```
FILE *file = fopen("numbers.txt", "r");
if (file == NULL) {
    printf("Error opening file!\n");
    return 1;
}

int num;
while (fscanf(file, "%d", &num) == 1) {
    printf("Read: %d\n", num);
}

fclose(file);
```

## 5.5.2 fgets() - Line Input

Reads a line from a file:

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

**Example:**

```
FILE *file = fopen("text.txt", "r");
if (file == NULL) {
    printf("Error opening file!\n");
    return 1;
}

char line[256];
while (fgets(line, sizeof(line), file) != NULL) {
    printf("%s", line);
}

fclose(file);
```

### 5.5.3 fgetc() - Character Input

Reads a single character:

```
int fgetc(FILE *stream);
```

**Example:**

```
FILE *file = fopen("text.txt", "r");
if (file == NULL) {
    printf("Error opening file!\n");
    return 1;
}

int ch;
while ((ch = fgetc(file)) != EOF) {
    putchar(ch);
}

fclose(file);
```

## 5.6 Writing to Text Files

### 5.6.1 fprintf() - Formatted Output

Similar to `printf()`, but writes to a file:

```
int fprintf(FILE *stream, const char *format, ...);
```

**Example:**

```
FILE *file = fopen("output.txt", "w");
if (file == NULL) {
    printf("Error opening file!\n");
    return 1;
}

fprintf(file, "Student ID: %d\n", 12345);
fprintf(file, "Name: %s\n", "Alice Johnson");
fprintf(file, "GPA: %.2f\n", 3.75);

fclose(file);
```

## 5.6.2 fputs() - String Output

Writes a string to a file:

```
int fputs(const char *str, FILE *stream);
```

**Example:**

```
FILE *file = fopen("output.txt", "w");
if (file == NULL) {
    printf("Error opening file!\n");
    return 1;
}

fputs("Hello, World!\n", file);
fputs("This is a test.\n", file);

fclose(file);
```

## 5.6.3 fputc() - Character Output

Writes a single character:

```
int fputc(int char, FILE *stream);
```

**Example:**

```
FILE *file = fopen("output.txt", "w");
if (file == NULL) {
    printf("Error opening file!\n");
    return 1;
}

for (char ch = 'A'; ch <= 'Z'; ch++) {
    fputc(ch, file);
}

fclose(file);
```

# 5.7 File Position Functions

## 5.7.1 fseek() - Move File Pointer

```
int fseek(FILE *stream, long offset, int origin);
```

**Origin values:**

- `SEEK_SET` - Beginning of file
- `SEEK_CUR` - Current position
- `SEEK_END` - End of file

**Example:**

```
FILE *file = fopen("data.txt", "r");

// Move to beginning
fseek(file, 0, SEEK_SET);

// Move 10 bytes from current position
fseek(file, 10, SEEK_CUR);

// Move to end of file
fseek(file, 0, SEEK_END);
```

## 5.7.2 ftell() - Get Current Position

```
long ftell(FILE *stream);
```

**Example:**

```
FILE *file = fopen("data.txt", "r");
long position = ftell(file);
printf("Current position: %ld\n", position);
```

## 5.7.3 rewind() - Reset to Beginning

```
void rewind(FILE *stream);
```

**Example:**

```
FILE *file = fopen("data.txt", "r");

// Read some data...

rewind(file);  // Go back to beginning
```

# 5.8 Checking End of File - feof()

```
int feof(FILE *stream);
```

**Returns non-zero if end of file is reached.**

**Example:**

```
FILE *file = fopen("data.txt", "r");
char ch;

while (!feof(file)) {
    ch = fgetc(file);
    if (ch != EOF) {
        putchar(ch);
    }
}

fclose(file);
```

# 5.9 Practical Example: Student Records Manager

```
#include <stdio.h>
#include <string.h>

typedef struct {
    int id;
    char name[50];
    float gpa;
    int age;
} Student;

// Save student to file
void saveStudent(const char *filename, Student s) {
    FILE *file = fopen(filename, "a");  // Append mode
    if (file == NULL) {
        printf("Error opening file!\n");
        return;
    }

    fprintf(file, "%d,%s,%.2f,%d\n", s.id, s.name, s.gpa, s.age);
    fclose(file);

    printf("Student record saved successfully!\n");
}

// Load all students from file
void loadStudents(const char *filename) {
    FILE *file = fopen(filename, "r");
    if (file == NULL) {
        printf("No records found!\n");
        return;
    }

    Student s;
    printf("\n=== Student Records ===\n");

    while (fscanf(file, "%d,%49[^,],%f,%d\n",
```

```
                &s.id, s.name, &s.gpa, &s.age) == 4) {
        printf("ID: %d | Name: %s | GPA: %.2f | Age: %d\n",
               s.id, s.name, s.gpa, s.age);
    }

    fclose(file);
}

// Search for student by ID
int searchStudent(const char *filename, int searchID) {
    FILE *file = fopen(filename, "r");
    if (file == NULL) {
        printf("Error opening file!\n");
        return 0;
    }

    Student s;
    while (fscanf(file, "%d,%49[^,],%f,%d\n",
                &s.id, s.name, &s.gpa, &s.age) == 4) {
        if (s.id == searchID) {
            printf("\n--- Student Found ---\n");
            printf("ID: %d\n", s.id);
            printf("Name: %s\n", s.name);
            printf("GPA: %.2f\n", s.gpa);
            printf("Age: %d\n", s.age);
            fclose(file);
            return 1;
        }
    }

    fclose(file);
    printf("Student not found!\n");
    return 0;
}

int main() {
    int choice;
    Student s;

    do {
```

```
printf("\n=== Student Management System ===\n");
printf("1. Add Student\n");
printf("2. View All Students\n");
printf("3. Search Student by ID\n");
printf("4. Exit\n");
printf("Enter choice: ");
scanf("%d", &choice);

switch (choice) {
    case 1:
        printf("\nEnter Student ID: ");
        scanf("%d", &s.id);

        printf("Enter Student Name: ");
        scanf(" %[^\n]", s.name);

        printf("Enter Student GPA: ");
        scanf("%f", &s.gpa);

        printf("Enter Student Age: ");
        scanf("%d", &s.age);

        saveStudent("students.txt", s);
        break;

    case 2:
        loadStudents("students.txt");
        break;

    case 3: {
        int searchID;
        printf("\nEnter Student ID to search: ");
        scanf("%d", &searchID);
        searchStudent("students.txt", searchID);
        break;
    }

    case 4:
        printf("Exiting program...\n");
        break;
```

```
            default:
                printf("Invalid choice!\n");
        }
    } while (choice != 4);

    return 0;
}
```

# 5.10 Binary File Operations

Binary files store data in raw binary format, which is more efficient for storing structures.

## 5.10.1 Writing Binary Data - fwrite()

```
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);
```

**Parameters:**

- `ptr` - Pointer to data to write
- `size` - Size of each element
- `nmemb` - Number of elements
- `stream` - File pointer

**Example:**

```
typedef struct {
    int id;
    char name[50];
    float gpa;
} Student;

Student s = {12345, "Alice Johnson", 3.75};

FILE *file = fopen("students.dat", "wb");
if (file != NULL) {
    fwrite(&s, sizeof(Student), 1, file);
    fclose(file);
}
```

## 5.10.2 Reading Binary Data - fread()

```
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
```

**Parameters:**

- `ptr` - Pointer to memory where data will be stored
- `size` - Size of each element
- `nmemb` - Number of elements
- `stream` - File pointer

**Example:**

```
Student s;

FILE *file = fopen("students.dat", "rb");
if (file != NULL) {
    while (fread(&s, sizeof(Student), 1, file) == 1) {
        printf("ID: %d, Name: %s, GPA: %.2f\n",
               s.id, s.name, s.gpa);
    }
    fclose(file);
}
```

## 5.10.3 Binary vs Text Files

| Aspect | Text Files | Binary Files |
|---|---|---|
| **Human Readable** | Yes | No |
| **Size** | Larger | Smaller |
| **Speed** | Slower | Faster |
| **Portability** | More portable | Less portable |
| **Precision** | May lose precision | Full precision |
| **Best for** | Configuration files, logs | Large data, structures |

# 5.11 Complete Binary File Example

```
#include <stdio.h>
#include <string.h>

typedef struct {
    int id;
    char name[50];
```

```
    float gpa;
    int age;
} Student;

// Save student to binary file
void saveStudentBinary(const char *filename, Student s) {
    FILE *file = fopen(filename, "ab");  // Append binary
    if (file == NULL) {
        printf("Error opening file!\n");
        return;
    }

    fwrite(&s, sizeof(Student), 1, file);
    fclose(file);

    printf("Student saved to binary file!\n");
}

// Load all students from binary file
void loadStudentsBinary(const char *filename) {
    FILE *file = fopen(filename, "rb");  // Read binary
    if (file == NULL) {
        printf("No records found!\n");
        return;
    }

    Student s;
    printf("\n=== Student Records (Binary) ===\n");

    while (fread(&s, sizeof(Student), 1, file) == 1) {
        printf("ID: %d | Name: %s | GPA: %.2f | Age: %d\n",
               s.id, s.name, s.gpa, s.age);
    }

    fclose(file);
}

// Count number of records in binary file
int countRecords(const char *filename) {
    FILE *file = fopen(filename, "rb");
    if (file == NULL) {
```

```
        return 0;
    }

    // Seek to end of file
    fseek(file, 0, SEEK_END);

    // Get file size
    long fileSize = ftell(file);

    fclose(file);

    // Calculate number of records
    return fileSize / sizeof(Student);
}

// Update student record by ID
int updateStudent(const char *filename, int id, Student newData) {
    FILE *file = fopen(filename, "rb+");  // Read and write binary
    if (file == NULL) {
        printf("Error opening file!\n");
        return 0;
    }

    Student s;
    int found = 0;
    long position = 0;

    // Search for student
    while (fread(&s, sizeof(Student), 1, file) == 1) {
        if (s.id == id) {
            // Move back to the position of this record
            fseek(file, position, SEEK_SET);

            // Write updated data
            fwrite(&newData, sizeof(Student), 1, file);

            found = 1;
            printf("Student record updated!\n");
            break;
        }
        position = ftell(file);
```

```
    }

    fclose(file);

    if (!found) {
        printf("Student ID %d not found!\n", id);
    }

    return found;
}

// Delete student record by ID
int deleteStudent(const char *filename, int id) {
    FILE *file = fopen(filename, "rb");
    FILE *temp = fopen("temp.dat", "wb");

    if (file == NULL || temp == NULL) {
        printf("Error opening files!\n");
        return 0;
    }

    Student s;
    int found = 0;

    // Copy all records except the one to delete
    while (fread(&s, sizeof(Student), 1, file) == 1) {
        if (s.id == id) {
            found = 1;
            continue;  // Skip this record
        }
        fwrite(&s, sizeof(Student), 1, temp);
    }

    fclose(file);
    fclose(temp);

    // Replace original file with temp file
    remove(filename);
    rename("temp.dat", filename);

    if (found) {
```

```
        printf("Student record deleted!\n");
    } else {
        printf("Student ID %d not found!\n", id);
    }

    return found;
}

int main() {
    int choice;
    Student s;

    do {
        printf("\n=== Student Management System (Binary Files) ===\n");
        printf("1. Add Student\n");
        printf("2. View All Students\n");
        printf("3. Count Records\n");
        printf("4. Update Student\n");
        printf("5. Delete Student\n");
        printf("6. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("\nEnter Student ID: ");
                scanf("%d", &s.id);

                printf("Enter Student Name: ");
                scanf(" %[^\n]", s.name);

                printf("Enter Student GPA: ");
                scanf("%f", &s.gpa);

                printf("Enter Student Age: ");
                scanf("%d", &s.age);

                saveStudentBinary("students.dat", s);
                break;

            case 2:
```

```
            loadStudentsBinary("students.dat");
            break;

        case 3: {
            int count = countRecords("students.dat");
            printf("\nTotal records: %d\n", count);
            break;
        }

        case 4: {
            int updateID;
            printf("\nEnter Student ID to update: ");
            scanf("%d", &updateID);

            printf("Enter new Student Name: ");
            scanf(" %[^\n]", s.name);

            printf("Enter new Student GPA: ");
            scanf("%f", &s.gpa);

            printf("Enter new Student Age: ");
            scanf("%d", &s.age);

            s.id = updateID;
            updateStudent("students.dat", updateID, s);
            break;
        }

        case 5: {
            int deleteID;
            printf("\nEnter Student ID to delete: ");
            scanf("%d", &deleteID);
            deleteStudent("students.dat", deleteID);
            break;
        }

        case 6:
            printf("Exiting program...\n");
            break;

        default:
```

```
                printf("Invalid choice!\n");
        }
    } while (choice != 6);

    return 0;
}
```

---

Revision #1
Created 2025-10-05 14:46:22 UTC by DS
Updated 2025-10-05 14:48:36 UTC by DS