Module 5 : Data Types (Struct, Enum, TypeDef) & File I/O

By the end of this module, students will be able to:

- Understand and implement user-defined data types using `struct`

- Utilize `enum` for creating readable constant sets

- Apply `typedef` to create type aliases for better code readability

- Perform file input/output operations in C

- Handle text and binary files effectively

- Design programs that persist data using files

- Combine structs with file I/O for data management systems

1. Introduction to User-Defined Data Types

1.1 Why User-Defined Types?

In previous modules, we learned about basic data types like `int`, `float`, `char`, etc. These are sufficient for simple programs, but real-world applications often require more complex data structures.

Example Problem: Suppose you want to store information about a student:

```
// Without user-defined types - difficult to manage
int student_id = 12345;
char student_name[50] = "Alice Johnson";
float student_gpa = 3.75;
int student_age = 20;
char student_major[30] = "Electrical Engineering";

// If you have 100 students, you need 500 variables!
```

User-defined types solve this problem by allowing us to group related data together.

2. Structures (struct)

2.1 What is a Structure?

A **structure** is a user-defined data type that groups variables of different types under a single name. Think of it as creating your own custom data type.

Python vs C Comparison:

Python	C
Uses classes or dictionaries	Uses `struct`
`student = {"name": "Alice", "age": 20}`	`struct Student student;`
Dynamic typing	Static typing

2.2 Declaring a Structure

Basic Syntax:

```
struct structure_name {
    data_type member1;
    data_type member2;
    // ... more members
};
```

Example - Student Structure:

```
struct Student {
    int id;
    char name[50];
    float gpa;
    int age;
    char major[30];
};
```

Important Notes:

- Structure declaration ends with a semicolon `;`
- Members can be of any data type (including other structures)

- The structure declaration itself doesn't allocate memory

2.3 Creating Structure Variables

Method 1: After Structure Declaration

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

// Create variables
struct Student student1;
struct Student student2, student3;
```

Method 2: During Structure Declaration

```
struct Student {
    int id;
    char name[50];
    float gpa;
} student1, student2;
```

Method 3: Anonymous Structure (less common)

```
struct {
    int id;
    char name[50];
    float gpa;
} student1, student2;
```

2.4 Initializing Structure Variables

Method 1: Member-by-Member Assignment

```
struct Student s1;
s1.id = 12345;
strcpy(s1.name, "Alice Johnson");  // Note: Use strcpy for strings
s1.gpa = 3.75;
```

Method 2: Initialization at Declaration

```
struct Student s1 = {12345, "Alice Johnson", 3.75};
```

Method 3: Designated Initializers (C99 and later)

```
struct Student s1 = {
    .id = 12345,
    .name = "Alice Johnson",
    .gpa = 3.75
};
```

Method 4: Partial Initialization

```
struct Student s1 = {12345};  // Only id is initialized, others are 0/NULL
```

2.5 Accessing Structure Members

Use the **dot operator (`.`)** to access structure members:

```
struct Student s1;

// Writing to members
s1.id = 12345;
s1.gpa = 3.75;
strcpy(s1.name, "Alice Johnson");

// Reading from members
printf("Student ID: %d\n", s1.id);
printf("Student Name: %s\n", s1.name);
printf("Student GPA: %.2f\n", s1.gpa);
```

2.6 Nested Structures

Structures can contain other structures as members:

```
struct Date {
    int day;
    int month;
    int year;
```

```
};

struct Student {
    int id;
    char name[50];
    float gpa;
    struct Date birthDate;  // Nested structure
};

// Usage
struct Student s1;
s1.id = 12345;
s1.birthDate.day = 15;
s1.birthDate.month = 8;
s1.birthDate.year = 2003;

printf("Birth Date: %d/%d/%d\n",
       s1.birthDate.day,
       s1.birthDate.month,
       s1.birthDate.year);
```

2.7 Array of Structures

You can create arrays of structures to handle multiple records:

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

// Array of 100 students
struct Student students[100];

// Accessing elements
students[0].id = 12345;
strcpy(students[0].name, "Alice");
students[0].gpa = 3.75;
```

```
// Loop through all students
for (int i = 0; i < 100; i++) {
    printf("Student %d: %s (GPA: %.2f)\n",
           students[i].id,
           students[i].name,
           students[i].gpa);
}
```

2.8 Pointers to Structures

You can use pointers with structures:

```
struct Student s1 = {12345, "Alice", 3.75};
struct Student *ptr = &s1;

// Method 1: Using (*ptr).member
printf("ID: %d\n", (*ptr).id);

// Method 2: Using ptr->member (preferred)
printf("ID: %d\n", ptr->id);
printf("Name: %s\n", ptr->name);
printf("GPA: %.2f\n", ptr->gpa);
```

The Arrow Operator (`->`):

- `ptr->member` is equivalent to `(*ptr).member`
- Much cleaner and more readable
- Commonly used when passing structures to functions

2.9 Structures and Functions

Passing by Value:

```
void printStudent(struct Student s) {
    printf("ID: %d\n", s.id);
    printf("Name: %s\n", s.name);
    printf("GPA: %.2f\n", s.gpa);
}

// Usage
```

```
struct Student s1 = {12345, "Alice", 3.75};
printStudent(s1);  // Entire structure is copied
```

Passing by Reference (Pointer):

```
void updateGPA(struct Student *s, float newGPA) {
    s->gpa = newGPA;
}

// Usage
struct Student s1 = {12345, "Alice", 3.75};
updateGPA(&s1, 3.85);  // Pass address of structure
printf("Updated GPA: %.2f\n", s1.gpa);  // Output: 3.85
```

Returning Structures from Functions:

```
struct Student createStudent(int id, char *name, float gpa) {
    struct Student s;
    s.id = id;
    strcpy(s.name, name);
    s.gpa = gpa;
    return s;
}

// Usage
struct Student s1 = createStudent(12345, "Alice", 3.75);
```

2.10 Practical Example: Student Database

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

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

// Function to input student data
```

```c
void inputStudent(struct Student *s) {
    printf("Enter 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);
}

// Function to display student data
void displayStudent(struct Student s) {
    printf("\n--- Student Information ---\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);
}

int main() {
    struct Student students[3];

    // Input data for 3 students
    for (int i = 0; i < 3; i++) {
        printf("\nEnter details for student %d:\n", i + 1);
        inputStudent(&students[i]);
    }

    // Display all students
    printf("\n\n=== All Students ===\n");
    for (int i = 0; i < 3; i++) {
        displayStudent(students[i]);
    }

    return 0;
}
```

3. Enumerations (enum)

3.1 What is an Enumeration?

An **enumeration** is a user-defined data type consisting of a set of named integer constants. Enums make code more readable by replacing "magic numbers" with meaningful names.

Python vs C Comparison:

Python	C
No built-in enum (uses constants)	Has `enum` keyword
`RED = 0; GREEN = 1; BLUE = 2`	`enum Color {RED, GREEN, BLUE};`

3.2 Declaring an Enum

Basic Syntax:

```
enum enum_name {
    constant1,
    constant2,
    constant3
};
```

Example:

```
enum Day {
    SUNDAY,     // 0
    MONDAY,     // 1
    TUESDAY,    // 2
    WEDNESDAY,  // 3
    THURSDAY,   // 4
    FRIDAY,     // 5
    SATURDAY    // 6
};
```

Key Points:

- By default, the first constant is assigned value 0

- Each subsequent constant is incremented by 1
- You can explicitly assign values

3.3 Custom Values in Enums

```
enum Status {
    SUCCESS = 1,
    FAILURE = 0,
    PENDING = -1
};

enum Month {
    JANUARY = 1,
    FEBRUARY,    // 2
    MARCH,       // 3
    APRIL,       // 4
    MAY,         // 5
    JUNE,        // 6
    JULY,        // 7
    AUGUST,      // 8
    SEPTEMBER,   // 9
    OCTOBER,     // 10
    NOVEMBER,    // 11
    DECEMBER     // 12
};
```

3.4 Using Enums

```
#include <stdio.h>

enum Day {
    SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
    THURSDAY, FRIDAY, SATURDAY
};

int main() {
    enum Day today = WEDNESDAY;

    printf("Today is day number: %d\n", today);  // Output: 3
```

```
    if (today == WEDNESDAY) {
        printf("It's the middle of the week!\n");
    }

    // Using enum in switch statement
    switch (today) {
        case MONDAY:
            printf("Start of work week\n");
            break;
        case FRIDAY:
            printf("TGIF!\n");
            break;
        case SATURDAY:
        case SUNDAY:
            printf("Weekend!\n");
            break;
        default:
            printf("Regular work day\n");
    }

    return 0;
}
```

3.5 Enums with Structures

Combining enums with structures creates powerful data models:

```
enum Grade {
    GRADE_A = 90,
    GRADE_B = 80,
    GRADE_C = 70,
    GRADE_D = 60,
    GRADE_F = 0
};

enum StudentStatus {
    ACTIVE,
    GRADUATED,
```

```
    SUSPENDED,
    WITHDRAWN
};

struct Student {
    int id;
    char name[50];
    enum Grade grade;
    enum StudentStatus status;
};

int main() {
    struct Student s1 = {
        .id = 12345,
        .name = "Alice",
        .grade = GRADE_A,
        .status = ACTIVE
    };

    printf("Student: %s\n", s1.name);

    if (s1.status == ACTIVE) {
        printf("Status: Active Student\n");
    }

    if (s1.grade >= GRADE_B) {
        printf("Good performance!\n");
    }

    return 0;
}
```

3.6 Practical Example: Traffic Light System

```
#include <stdio.h>

enum TrafficLight {
    RED,
    YELLOW,
```

```
    GREEN
};

void displayLightAction(enum TrafficLight light) {
    switch (light) {
        case RED:
            printf("STOP! Red light is on.\n");
            break;
        case YELLOW:
            printf("CAUTION! Yellow light is on.\n");
            break;
        case GREEN:
            printf("GO! Green light is on.\n");
            break;
        default:
            printf("Invalid light state.\n");
    }
}

int main() {
    enum TrafficLight currentLight = RED;

    printf("Traffic Light Simulation:\n\n");

    for (int i = 0; i < 3; i++) {
        displayLightAction(currentLight);

        // Cycle through lights
        if (currentLight == RED) {
            currentLight = GREEN;
        } else if (currentLight == GREEN) {
            currentLight = YELLOW;
        } else {
            currentLight = RED;
        }

        printf("Waiting...\n\n");
    }

    return 0;
```

```
}
```

4. Type Definitions (typedef)

4.1 What is typedef?

`typedef` creates aliases (alternative names) for existing data types. It makes code more readable and easier to maintain.

Basic Syntax:

```
typedef existing_type new_name;
```

4.2 typedef with Basic Types

```
// Create aliases for basic types
typedef int Integer;
typedef float Real;
typedef char Character;

// Usage
Integer age = 25;
Real temperature = 36.5;
Character grade = 'A';
```

4.3 typedef with Structures

Without typedef:

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

// Must always use "struct Student"
struct Student s1;
struct Student students[100];
```

With typedef - Method 1:

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

typedef struct Student Student;

// Now can use just "Student"
Student s1;
Student students[100];
```

With typedef - Method 2 (Combined):

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

// Usage
Student s1;
Student students[100];
```

With typedef - Method 3 (Anonymous struct):

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

// Usage
Student s1;
```

4.4 typedef with Enums

```
typedef enum {
    MONDAY,
    TUESDAY,
    WEDNESDAY,
    THURSDAY,
    FRIDAY,
    SATURDAY,
    SUNDAY
} Day;

// Usage
Day today = WEDNESDAY;
Day weekend[2] = {SATURDAY, SUNDAY};
```

4.5 typedef with Pointers

```
typedef int* IntPtr;
typedef struct Student* StudentPtr;

// Usage
IntPtr p1, p2;  // Both are int pointers
StudentPtr sptr;  // Student pointer
```

4.6 typedef with Arrays

```
typedef int IntArray[10];
typedef char String[100];

// Usage
IntArray numbers;  // Same as: int numbers[10];
String name;       // Same as: char name[100];
```

4.7 Practical Example: Complex Data Types

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

// Define enumeration for course grades
```

```
typedef enum {
    GRADE_A = 4,
    GRADE_B = 3,
    GRADE_C = 2,
    GRADE_D = 1,
    GRADE_F = 0
} Grade;

// Define structure for a course
typedef struct {
    char code[10];
    char name[50];
    int credits;
    Grade grade;
} Course;

// Define structure for a student
typedef struct {
    int id;
    char name[50];
    Course courses[5];
    int numCourses;
    float gpa;
} Student;

// Function to calculate GPA
float calculateGPA(Student *s) {
    int totalCredits = 0;
    float totalPoints = 0.0;

    for (int i = 0; i < s->numCourses; i++) {
        totalCredits += s->courses[i].credits;
        totalPoints += s->courses[i].credits * s->courses[i].grade;
    }

    if (totalCredits == 0) return 0.0;
    return totalPoints / totalCredits;
}

int main() {
```

```
    Student student = {
        .id = 12345,
        .name = "Alice Johnson",
        .numCourses = 3
    };

    // Add courses
    strcpy(student.courses[0].code, "EE101");
    strcpy(student.courses[0].name, "Circuit Analysis");
    student.courses[0].credits = 3;
    student.courses[0].grade = GRADE_A;

    strcpy(student.courses[1].code, "MATH201");
    strcpy(student.courses[1].name, "Calculus II");
    student.courses[1].credits = 4;
    student.courses[1].grade = GRADE_B;

    strcpy(student.courses[2].code, "CS101");
    strcpy(student.courses[2].name, "Programming");
    student.courses[2].credits = 3;
    student.courses[2].grade = GRADE_A;

    // Calculate and display GPA
    student.gpa = calculateGPA(&student);

    printf("Student: %s (ID: %d)\n", student.name, student.id);
    printf("GPA: %.2f\n\n", student.gpa);

    printf("Courses:\n");
    for (int i = 0; i < student.numCourses; i++) {
        printf("  %s - %s (%d credits): Grade %d\n",
               student.courses[i].code,
               student.courses[i].name,
               student.courses[i].credits,
               student.courses[i].grade);
    }

    return 0;
}
```

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
`"w"`	Write only	Overwrites content	Creates new file

Mode	Description	If File Exists	If File Doesn't Exist
`"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

```c
#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;
}
```