Files

Objectives

After studying this section, you should be able to:

- What is a file?
- How are data stored in files?
- How to access data in files?

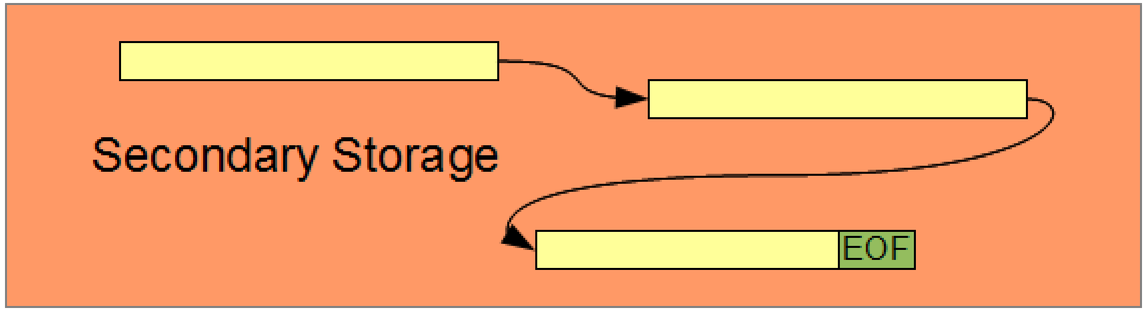
Contents

1. What is a file?
2. Why do we need File Handling in C?
3. Types of Files in C
4. C File Operations: Open, Close, Create, Read, Write, …

1. What is a file?

What is a file?

- A file is a named area of secondary storage.
- The file may be fragmented; that is, it may consist of several parts stored at different non-contiguous locations in secondary memory.
- A file does not necessarily occupy contiguous space on the storage device.

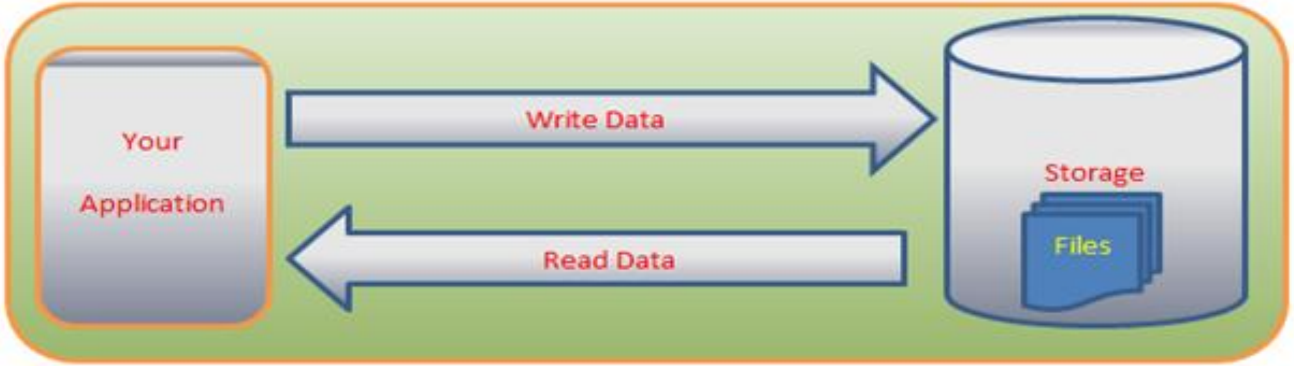


- The byte is the fundamental storage unit of a file. The distinguishing feature of a file is the end-of-file mark. We refer to this mark as **EOF** (**EOF** typically has the value -1)

2. Why do we need File Handling in C?

Why do we need File Handling in C?

- The operations using the C program are done on a prompt/terminal which is not stored anywhere.
- The output is deleted when the program is closed.
- However, in the software industry, most programs are written to store the information fetched from the program. The use of file handling is exactly what the situation calls for.



Why do we need File Handling in C?

- **Reusability**: The data stored in the file can be accessed, updated, and deleted anywhere and anytime providing high reusability.
- **Portability**: Without losing any data, files can be transferred to another in the computer system. The risk of flawed coding is minimized with this feature.
- **Efficient**: A large amount of input may be required for some programs. File handling allows you to easily access a part of a file using few instructions which saves a lot of time and reduces the chance of errors.
- **Storage Capacity**: Files allow you to store a large amount of data without having to worry about storing everything simultaneously in a program.

3. Types of Files in C

Type of Files in C

- A file can be classified into two types based on the way the file stores the data. They are as follows:
  - **Text Files**
  - **Binary Files**

Hello World !!!
file.txt
1001001100010
1001011110010
1001001111110
file.bin

Text Files

- A text file contains data in the **form of ASCII characters** and is generally used to store a stream of characters.

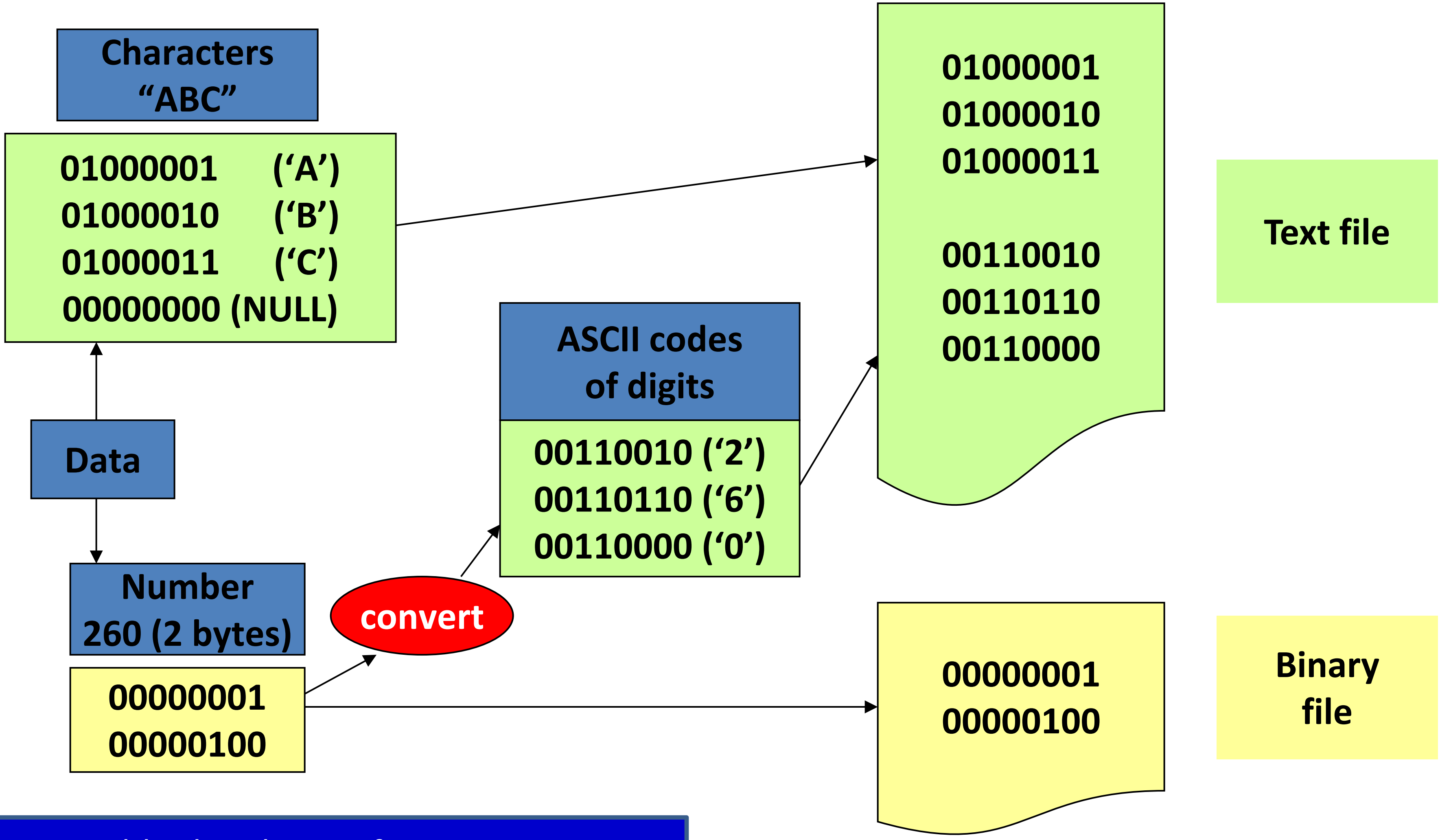
- Each line in a text file ends with a new line character (**‘\n’**).
- It can be read or written by any text editor.
- They are generally stored with **.txt** file extension.
- Text files can also be used to store the source code.

Binary Files

- A binary file contains data in **binary form (i.e. 0’s and 1’s)** instead of ASCII characters. They contain data that is stored in a similar manner to how it is stored in the main memory.

  - The binary files can be created only from within a program and their contents can only be read by a program.
  - More secure as they are not easily readable.
  - They are generally stored with **.bin** file extension.

Type of Files in C (cont.)



Text format is more portable than binary format
But binary format is more efficient than text format

4. C File Operations

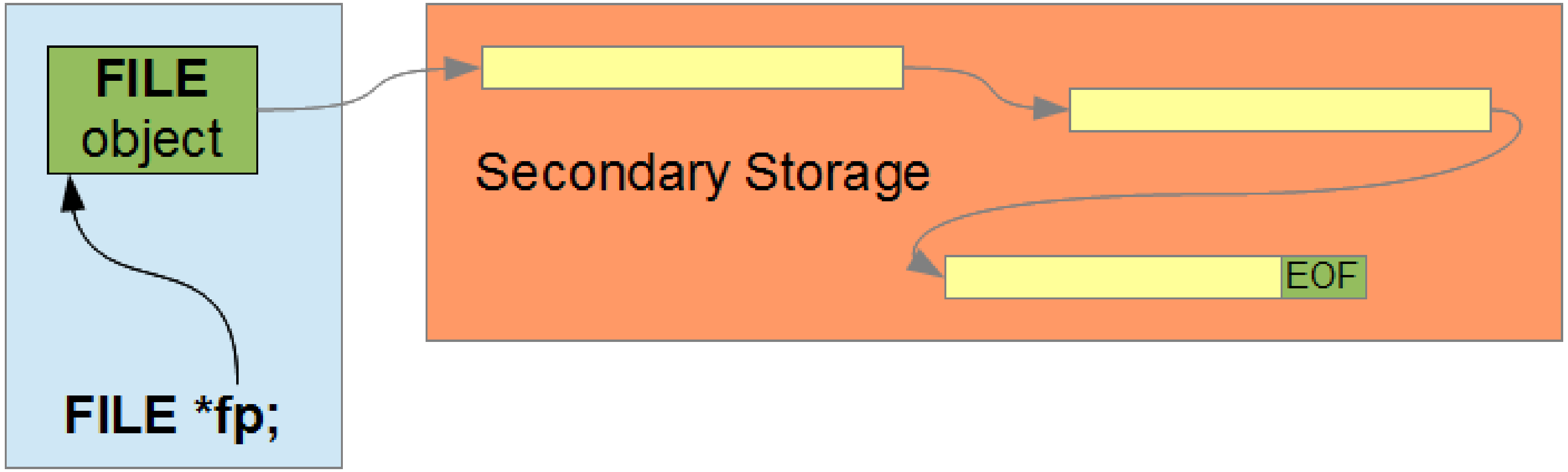
Operations on Files

There are different kinds of file operations in C:

1. Creating a new file
2. Opening an existing file
3. Reading from file
4. Writing to a file
5. Moving to a specific location in a file
6. Closing a file

4.1. Connection to File

- A C program connects to a file through an object of **FILE** type.
- We use a library function to retrieve the address of the file object, store that address in a pointer and subsequently access the file through that pointer called **File Pointer**.

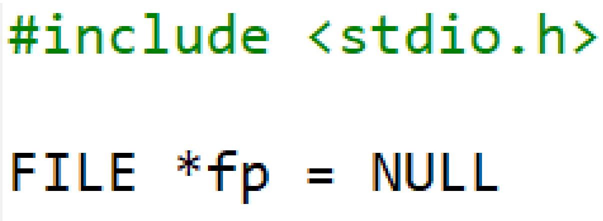


4.1. Connection to File (cont.)

- Allocating a pointer to a FILE object takes the form:

```
FILE *identifier;
```

- Where FILE is the type of the FILE object and identifier is the name of the pointer to the FILE object. We call this pointer a handle to the object.
- The structure type FILE is declared in the **<stdio.h>** header file. To allocate memory for a FILE pointer, we write:



```
#include <stdio.h>

FILE *fp = NULL
```

4.2. Open a File in C

- For opening a file in C, the **fopen()** function is used with the filename or file path along with the required access modes.
- **Syntax**:

```
FILE* fopen(const char *file_name, const char *access_mode);
```

- **Parameters**:
  - file_name: name of the file when present in the same directory as the source file. Otherwise, full path.
  - access_mode: Specifies for what operation the file is being opened.
- **Return Value**:
  - If the file is opened successfully, returns a file pointer to it.
  - If the file is not opened, then returns NULL.

File opening modes in C

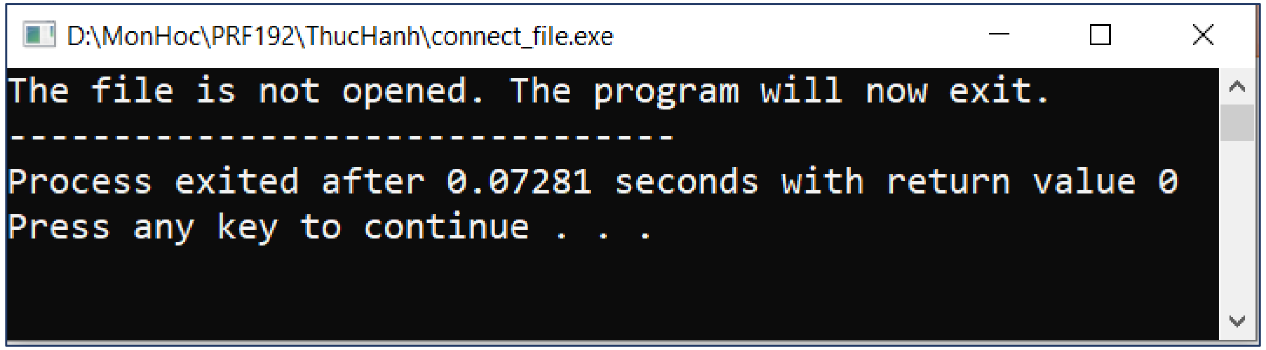
Mode	Description	Example
r	Opens a text file in read-only mode, allowing only reading operations.	fopen("demo.txt", "r")
w	When using the mode "w", fopen() initializes a text file for writing exclusively. If the file already exists, it clears its contents; otherwise, it creates a new file for writing.	fopen("demo.txt", "w")
a	When employing the "a" mode, fopen() enables opening a text file in append mode. This mode permits writing data to the end of the file, preserving existing content.	fopen("demo.txt", "a")
r+	When using the "r+" mode, fopen() facilitates opening a text file for both reading and writing operations. This mode grants the ability to manipulate data at any position within the file.	fopen("demo.txt", "r+")
w+	This mode opens a text file for both reading and writing. If the file with same name already exists, it truncates the file to zero length; otherwise, it creates a new file for both reading and writing operations.	fopen("demo.txt", "w+")

File opening modes in C (cont.)

Mode	Description	Example
a+	This mode opens a text file for both reading and writing, enabling data to be appended to the end of the file without overwriting existing content.	fopen("demo.txt", "a+")
rb	Opens a binary file in read-only mode, allowing reading operations on binary data.	fopen("demo.bin", "rb")
wb	Opens a binary file in write-only mode, truncating the file to zero length if it exists or creating a new file for writing binary data.	fopen("demo.bin", "wb")
ab+	This mode opens a binary file for both reading and writing operations, allowing binary data to be appended to the end of the file without overwriting existing content.	fopen("demo.bin", "ab+")
rb+	Opens a binary file for both reading and writing operations on binary data.	fopen("demo.bin", "rb+")
wb+	Opens a binary file for both reading and writing operations, truncating the file to zero length if it exists or creating a new file for reading and writing binary data.	fopen("demo.bin", "wb+")

Example of Opening a File

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    // file pointer variable to store the value returned by
    // fopen
    FILE* fptr;
    // opening the file in read mode
    fptr = fopen("filename.txt", "r");
    // checking if the file is opened successfully
    if (fptr == NULL) {
        printf("The file is not opened. The program will "
               "now exit.");
        exit(0);
    }
    return 0;
}
```



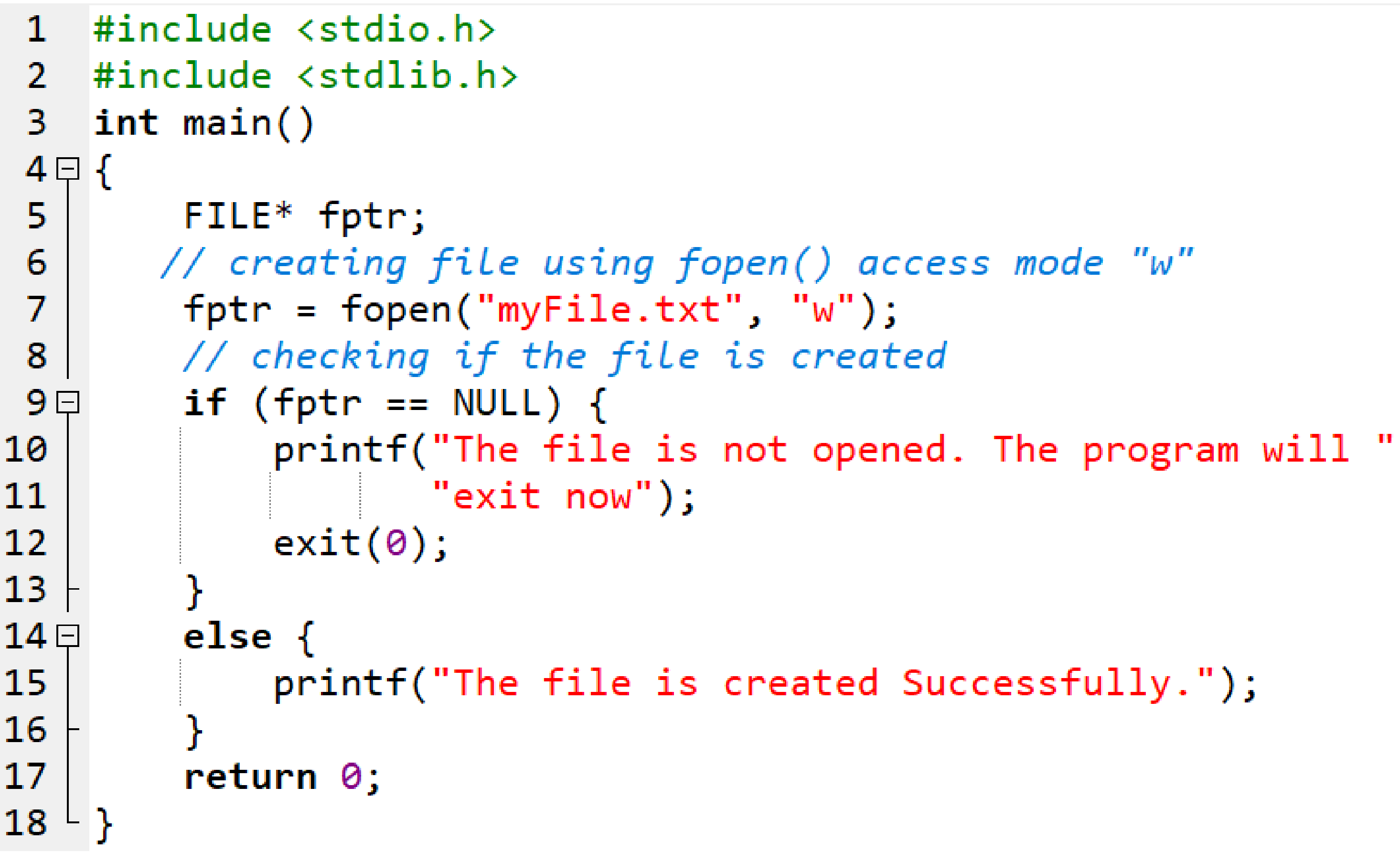
```
The file is not opened. The program will now exit.
--------------------------------
Process exited after 0.07281 seconds with return value 0
Press any key to continue . . .
```

4.3. Create a File in C

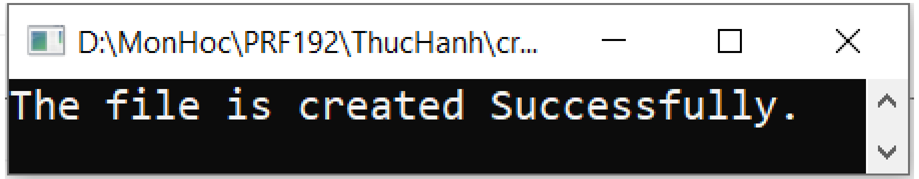
- The **fopen()** function can not only open a file but also can create a file if it does not exist already.
- Use the modes that allow the creation of a file if not found such as **w**, **w+**, **wb**, **wb+**, **a**, **a+**, **ab**, and **ab+**.
- **Syntax**:

```
FILE *fptr;
fptr = fopen("filename.txt", "w");
```

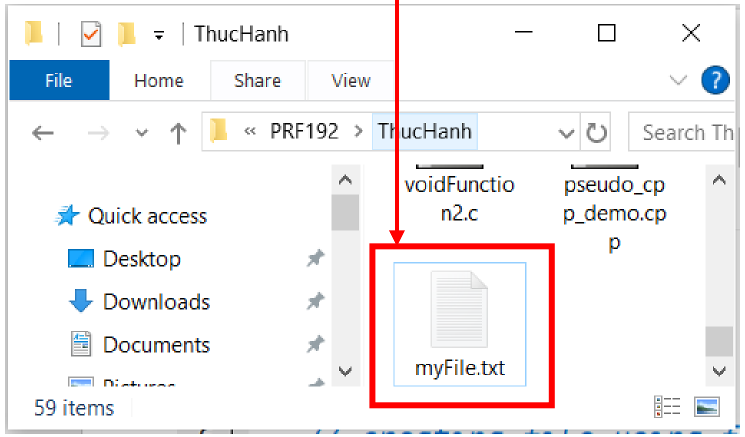
Example of Create a File



```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    FILE* fptr;
   // creating file using fopen() access mode "w"
    fptr = fopen("myFile.txt", "w");
    // checking if the file is created
    if (fptr == NULL) {
        printf("The file is not opened. The program will "
               "exit now");
        exit(0);
    }
    else {
        printf("The file is created Successfully.");
    }
    return 0;
}
```



```
The file is created Successfully.
```



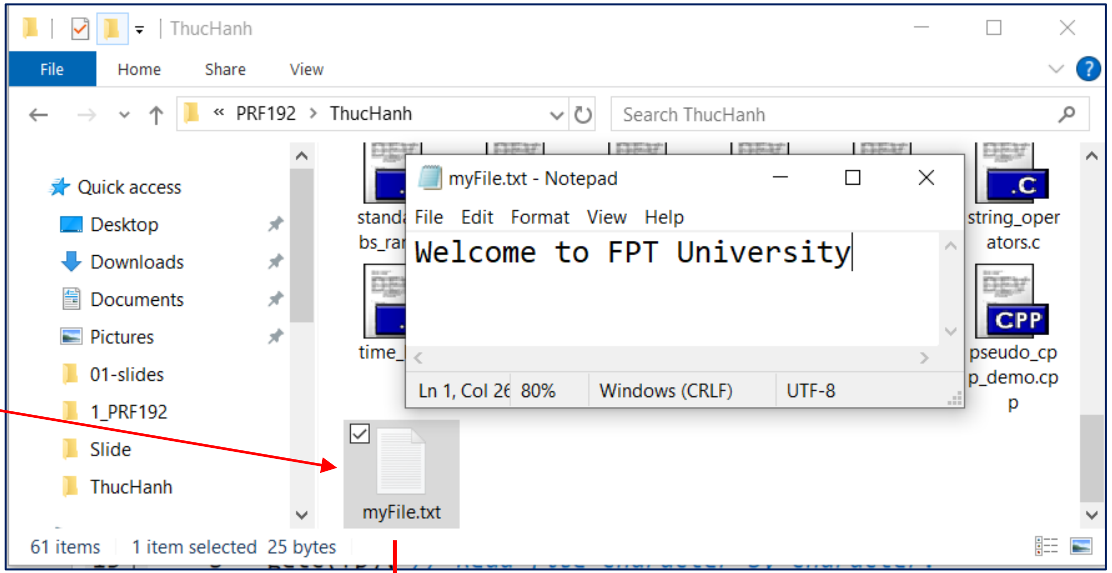
4.4. Reading From a File

- Using the functions to read data from a file

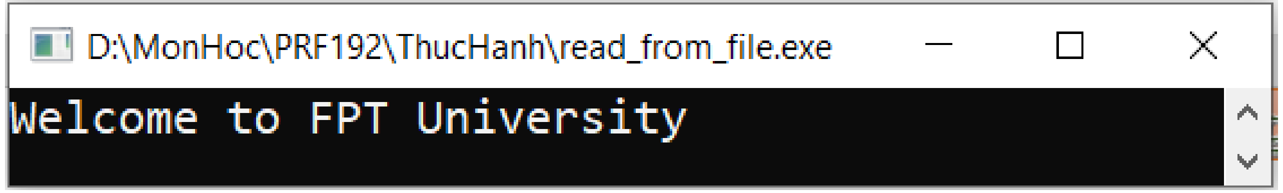
Mode	Description
fscanf()	Retrieves input from a file using a formatted string and variable argument list.
fgets()	Obtains a complete line of text from the file.
fgetc()	Reads a single character from the file.
fgetw()	Reads a numerical value from the file.
fread()	Extracts a specified number of bytes from a binary file.

4.4. Reading From a File: Example

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
  FILE * fp;
  char s;
  fp = fopen("myFile.txt", "r");
  if (fp == NULL) {
    printf("\nCAN NOT OPEN FILE");
    exit(1);
  }
  do {
    s = getc(fp); // Read file character by character.
    printf("%c", s);
  }
  while (s != EOF);
  fclose(fp);

  return 0;
}
```



```
Welcome to FPT University
```



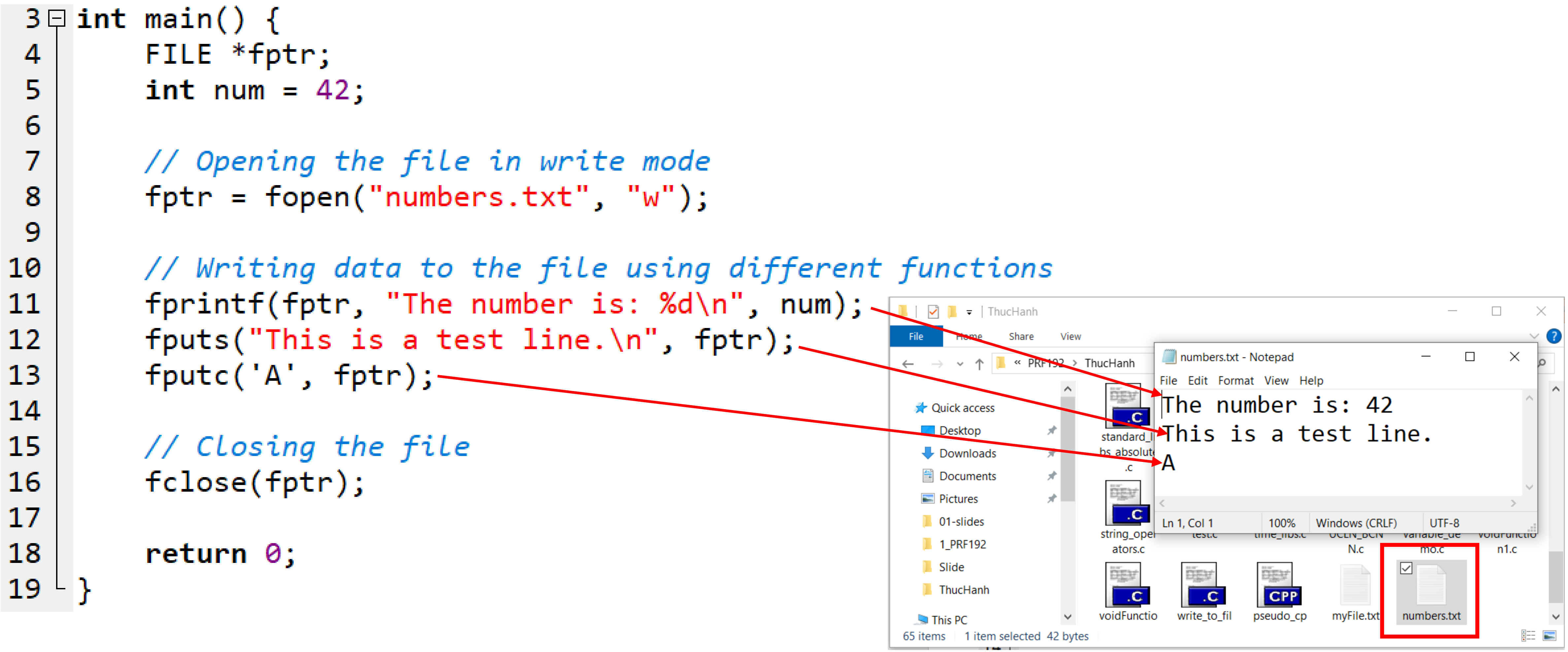
```
Welcome to FPT University
```

4.5. Write to a File

- Using the functions to **write** data to a file

Mode	Description
fprintf()	This function, akin to printf(), uses a formatted string and variable arguments list to print output to the file.
fputs()	Prints an entire line in the file along with a newline character at the end.
fputc()	Writes a single character into the file.
putw()	Writes a number to the file.
fwrite()	Writes the specified number of bytes to the binary file.

4.5. Write to a File: Example

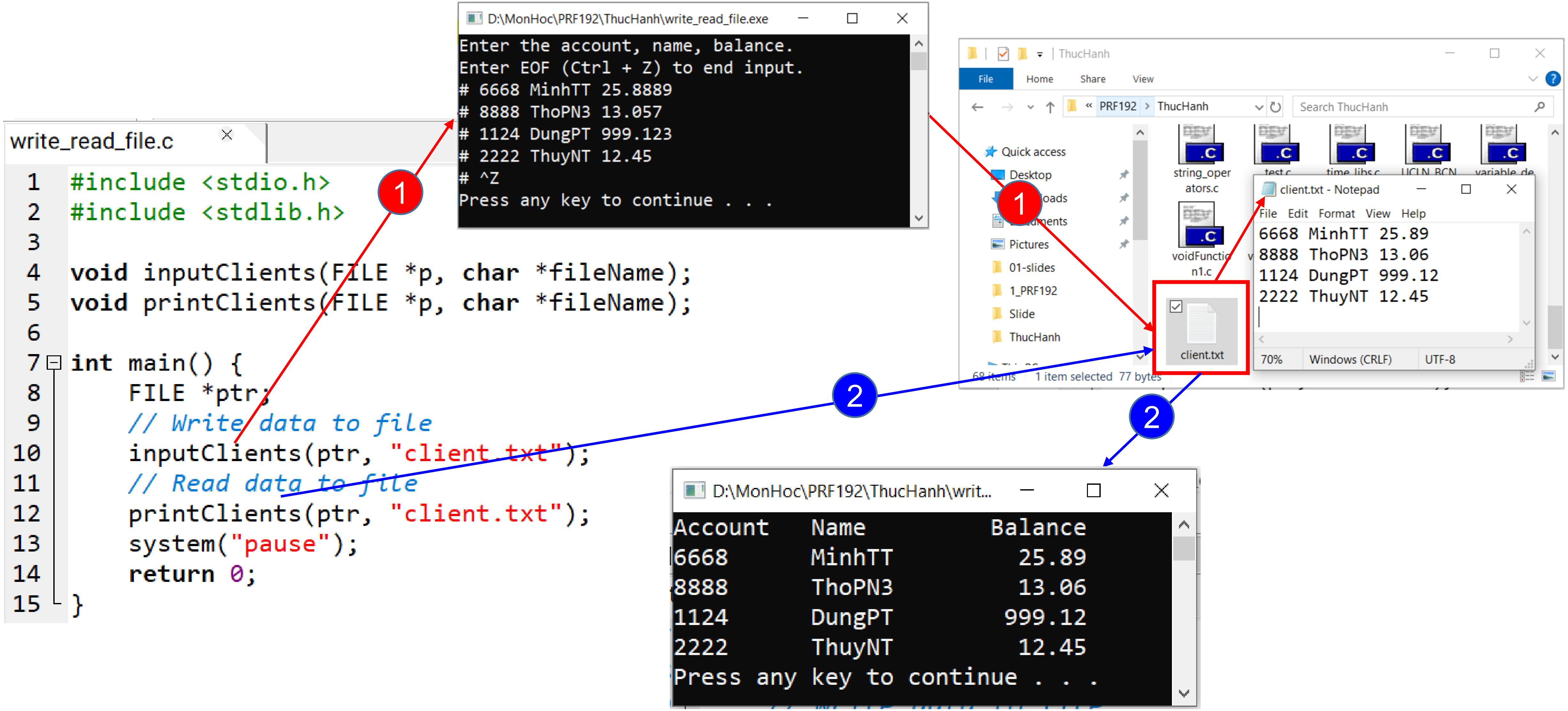


```c
int main() {
    FILE *fptr;
    int num = 42;

    // Opening the file in write mode
    fptr = fopen("numbers.txt", "w");

    // Writing data to the file using different functions
    fprintf(fptr, "The number is: %d\n", num);
    fputs("This is a test line.\n", fptr);
    fputc('A', fptr);

    // Closing the file
    fclose(fptr);

    return 0;
}
```

numbers.txt - Notepad

```
The number is: 42
This is a test line.
A
```

Demo: Program Write and Read text file



write_read_file.c

```c
#include <stdio.h>
#include <stdlib.h>

void inputClients(FILE *p, char *fileName);
void printClients(FILE *p, char *fileName);

int main() {
    FILE *ptr;
    // Write data to file
    inputClients(ptr, "client.txt");
    // Read data to file
    printClients(ptr, "client.txt");
    system("pause");
    return 0;
}
```

D:\MonHoc\PRF192\ThucHanh\write_read_file.exe

```
Enter the account, name, balance.
Enter EOF (Ctrl + Z) to end input.
# 6668 MinhTT 25.8889
# 8888 ThoPN3 13.057
# 1124 DungPT 999.123
# 2222 ThuyNT 12.45
# ^Z
Press any key to continue . . .
```

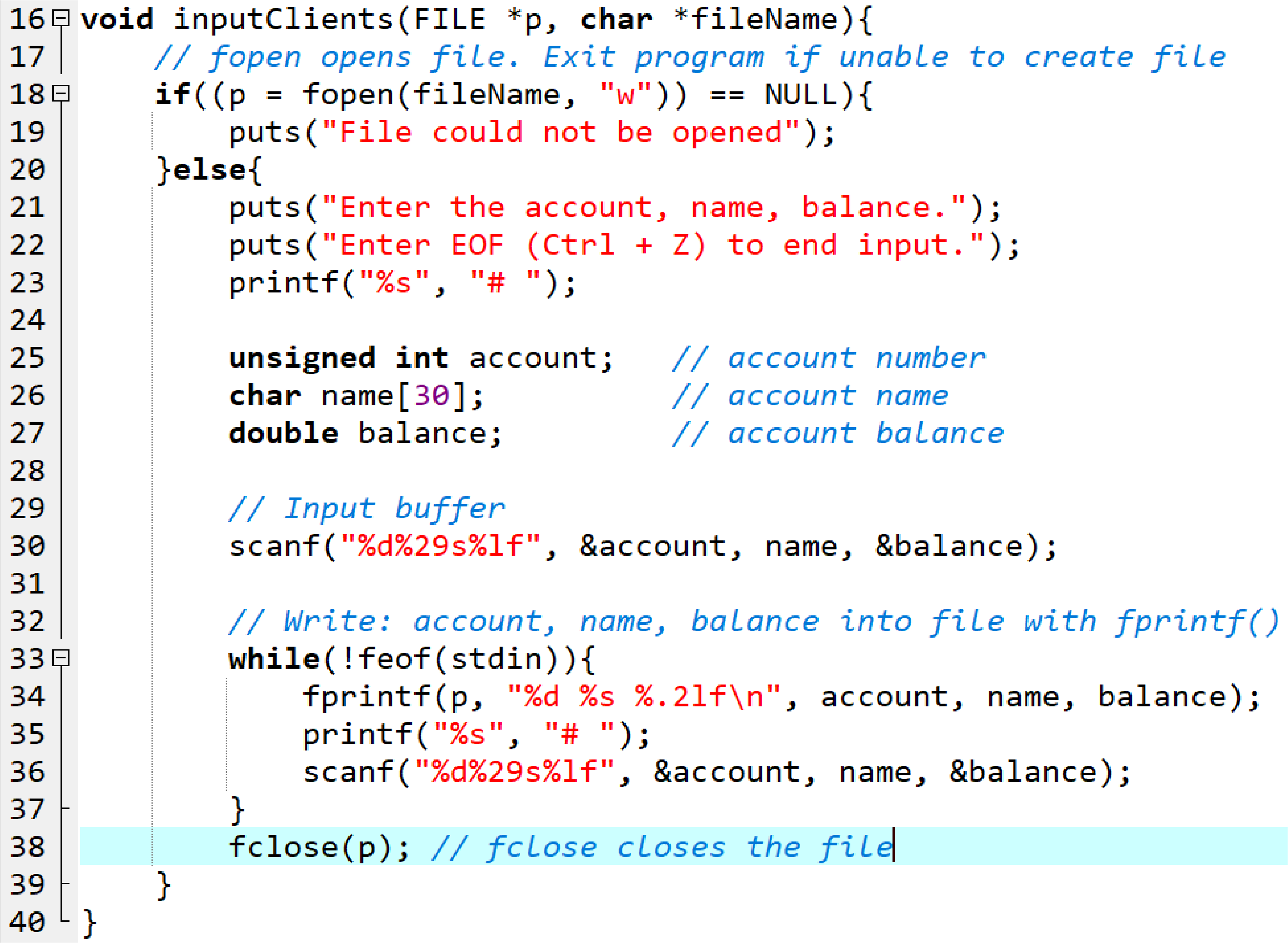
client.txt - Notepad

```
6668 MinhTT 25.89
8888 ThoPN3 13.06
1124 DungPT 999.12
2222 ThuyNT 12.45
```

D:\MonHoc\PRF192\ThucHanh\writ...

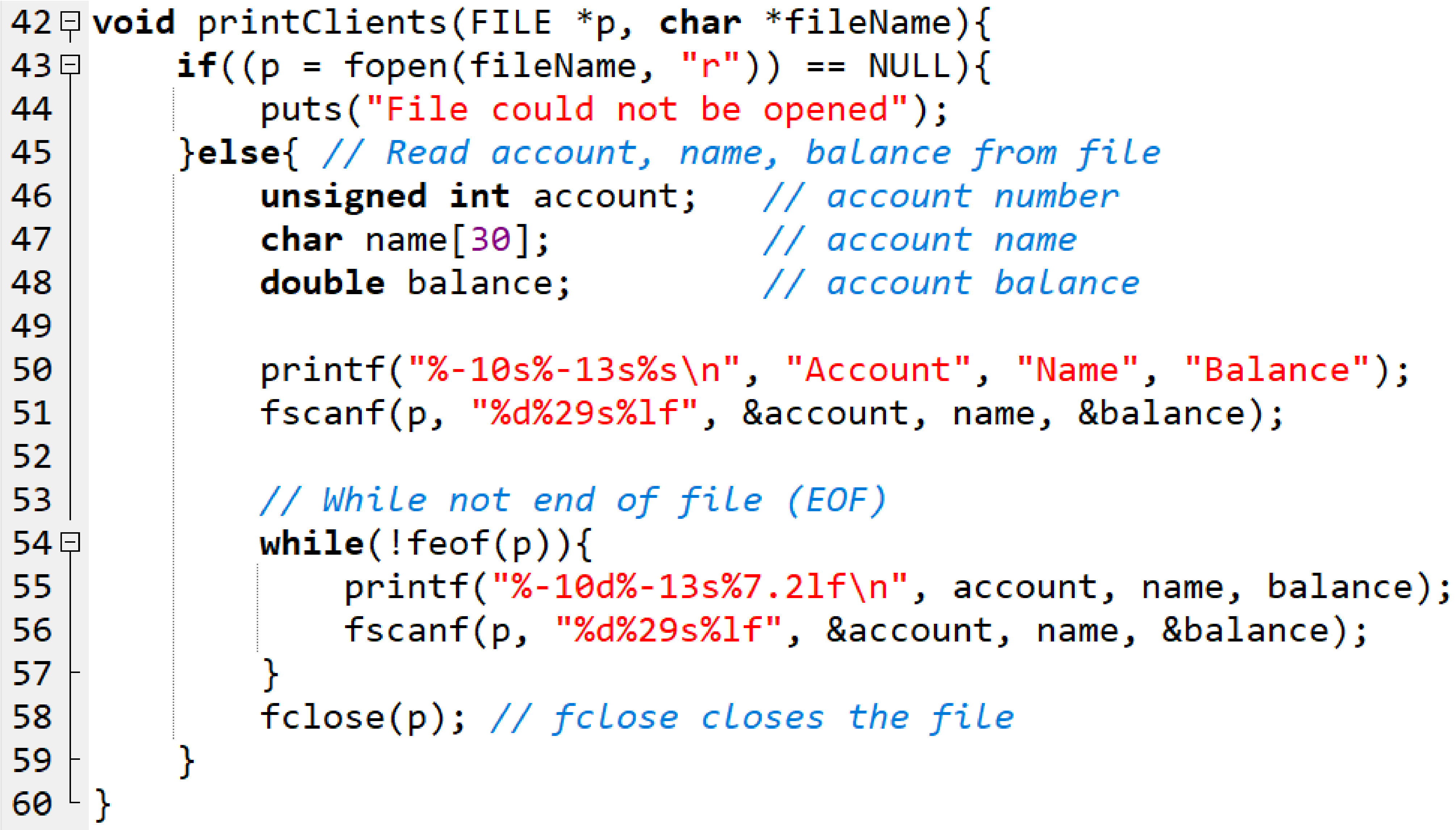
```
Account     Name            Balance
6668        MinhTT            25.89
8888        ThoPN3            13.06
1124        DungPT           999.12
2222        ThuyNT            12.45
Press any key to continue . . .
```

Demo (cont.)



```
void inputClients(FILE *p, char *fileName){
    // fopen opens file. Exit program if unable to create file
    if((p = fopen(fileName, "w")) == NULL){
        puts("File could not be opened");
    }else{
        puts("Enter the account, name, balance.");
        puts("Enter EOF (Ctrl + Z) to end input.");
        printf("%s", "# ");

        unsigned int account;   // account number
        char name[30];          // account name
        double balance;         // account balance

        // Input buffer
        scanf("%d%29s%lf", &account, name, &balance);

        // Write: account, name, balance into file with fprintf()
        while(!feof(stdin)){
            fprintf(p, "%d %s %.2lf\n", account, name, balance);
            printf("%s", "# ");
            scanf("%d%29s%lf", &account, name, &balance);
        }
        fclose(p); // fclose closes the file
    }
}
```

Demo (cont.)



```
void printClients(FILE *p, char *fileName){
    if((p = fopen(fileName, "r")) == NULL){
        puts("File could not be opened");
    }else{ // Read account, name, balance from file
        unsigned int account;   // account number
        char name[30];          // account name
        double balance;         // account balance

        printf("%-10s%-13s%s\n", "Account", "Name", "Balance");
        fscanf(p, "%d%29s%lf", &account, name, &balance);

        // While not end of file (EOF)
        while(!feof(p)){
            printf("%-10d%-13s%7.2lf\n", account, name, balance);
            fscanf(p, "%d%29s%lf", &account, name, &balance);
        }
        fclose(p); // fclose closes the file
    }
}
```

4.6. Write to a Binary File

- When writing data to a binary file, use the fwrite() function. This function allows us to store information in binary form, comprising sequences of bits (0s and 1s).
- **Syntax**:

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

- **Parameters:**
  - ptr: A reference to the memory block holding the data intended for writing.
  - size: The byte size of each element to be written.
  - nmemb: The count of elements to be written.
  - file_pointer: The FILE pointer associated with the output file stream.
- **Return Value**: returns the number of objects successfully written

4.7. Reading from Binary File

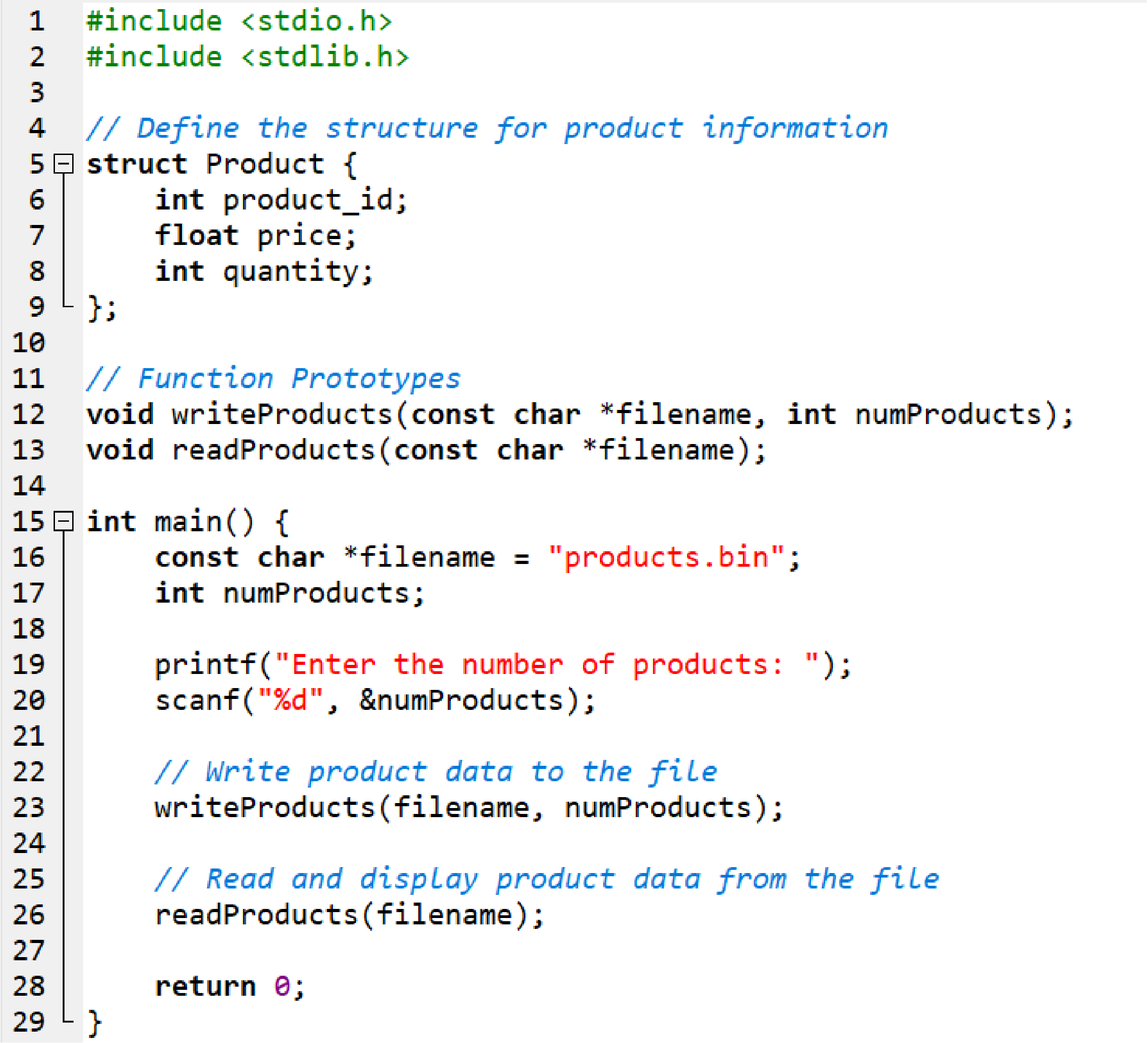
- When retrieving data from a binary file, use the fread() function. This function facilitates the extraction of data from the file in its binary representation.
- **Syntax**:

```
size_t fread(void *memory_ptr, size_t size, size_t nmemb, FILE *file_pointer);
```

- **Parameters**:
  - memory_ptr: A pointer to the memory block where the data will be stored.
  - size: The size of each element to be read (in bytes).
  - nmemb: The number of elements to be read.
  - file_pointer: The FILE pointer pointing to the input file stream.
- **Return Value**: The function returns the number of objects successfully read.

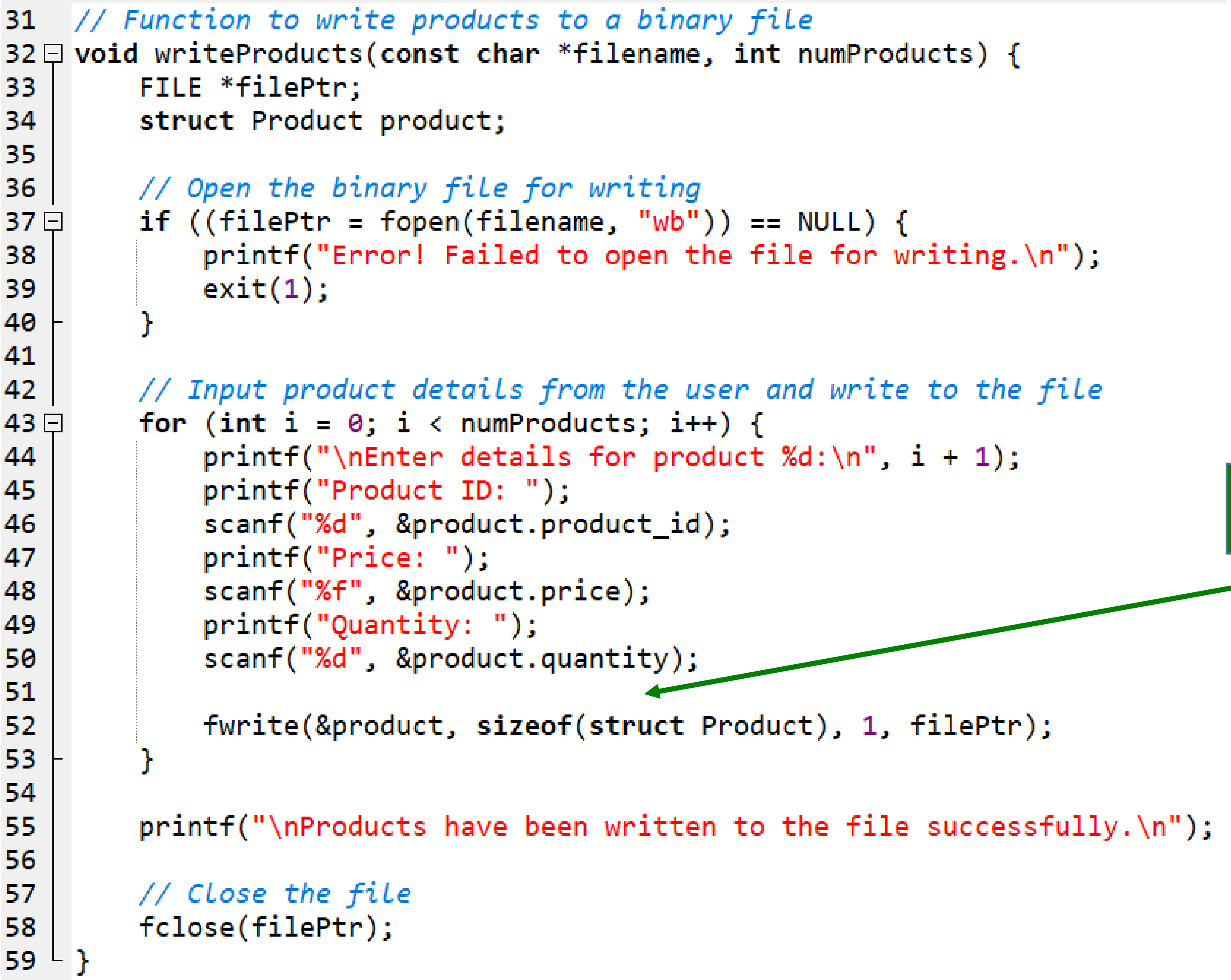
Write and Read Binary File: Demo

- Development of a program to manage ‘n’ product. Each product includes the following information: Product code, Price and quantity.
- **Requirements:**
  - Enter product information from the keyboard and save it to a binary file
  - Read data from the binary file and Print out a list of products



```c
#include <stdio.h>
#include <stdlib.h>

// Define the structure for product information
struct Product {
    int product_id;
    float price;
    int quantity;
};

// Function Prototypes
void writeProducts(const char *filename, int numProducts);
void readProducts(const char *filename);

int main() {
    const char *filename = "products.bin";
    int numProducts;

    printf("Enter the number of products: ");
    scanf("%d", &numProducts);

    // Write product data to the file
    writeProducts(filename, numProducts);

    // Read and display product data from the file
    readProducts(filename);

    return 0;
}
```

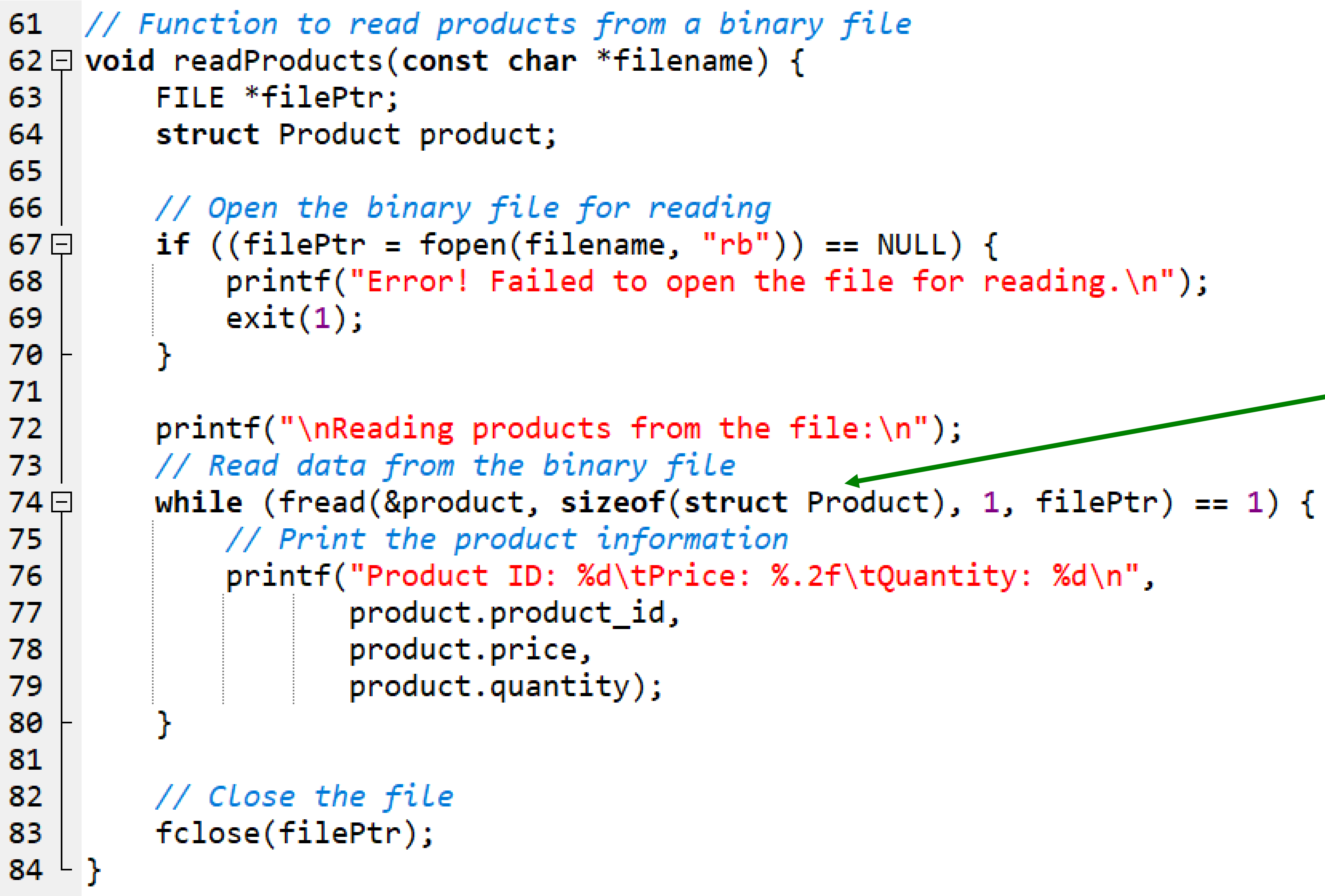
Write and Read Binary File: Demo



```c
// Function to write products to a binary file
void writeProducts(const char *filename, int numProducts) {
    FILE *filePtr;
    struct Product product;

    // Open the binary file for writing
    if ((filePtr = fopen(filename, "wb")) == NULL) {
        printf("Error! Failed to open the file for writing.\n");
        exit(1);
    }

    // Input product details from the user and write to the file
    for (int i = 0; i < numProducts; i++) {
        printf("\nEnter details for product %d:\n", i + 1);
        printf("Product ID: ");
        scanf("%d", &product.product_id);
        printf("Price: ");
        scanf("%f", &product.price);
        printf("Quantity: ");
        scanf("%d", &product.quantity);

        fwrite(&product, sizeof(struct Product), 1, filePtr);
    }

    printf("\nProducts have been written to the file successfully.\n");

    // Close the file
    fclose(filePtr);
}
```

Write to binary file

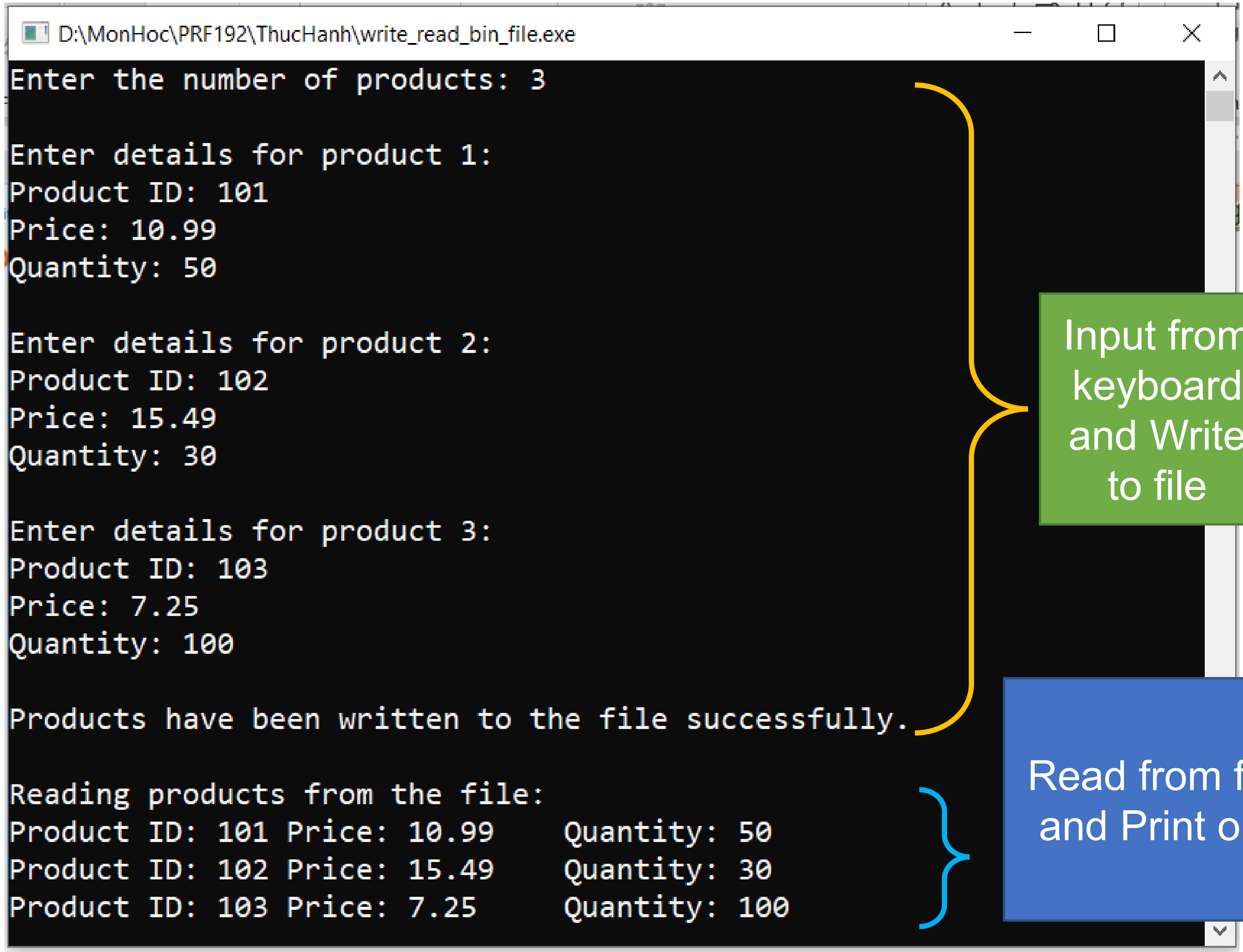
Write and Read Binary File: Demo



```c
// Function to read products from a binary file
void readProducts(const char *filename) {
    FILE *filePtr;
    struct Product product;

    // Open the binary file for reading
    if ((filePtr = fopen(filename, "rb")) == NULL) {
        printf("Error! Failed to open the file for reading.\n");
        exit(1);
    }

    printf("\nReading products from the file:\n");
    // Read data from the binary file
    while (fread(&product, sizeof(struct Product), 1, filePtr) == 1) {
        // Print the product information
        printf("Product ID: %d\tPrice: %.2f\tQuantity: %d\n",
               product.product_id,
               product.price,
               product.quantity);
    }

    // Close the file
    fclose(filePtr);
}
```

Read to binary file

Write and Read Binary File: Demo

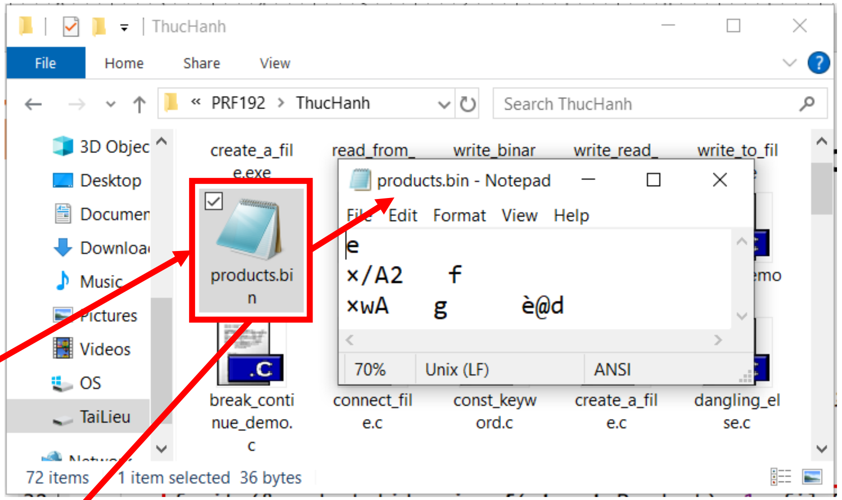


D:\MonHoc\PRF192\ThucHanh\write_read_bin_file.exe

```
Enter the number of products: 3

Enter details for product 1:
Product ID: 101
Price: 10.99
Quantity: 50

Enter details for product 2:
Product ID: 102
Price: 15.49
Quantity: 30

Enter details for product 3:
Product ID: 103
Price: 7.25
Quantity: 100

Products have been written to the file successfully.

Reading products from the file:
Product ID: 101 Price: 10.99    Quantity: 50
Product ID: 102 Price: 15.49    Quantity: 30
Product ID: 103 Price: 7.25     Quantity: 100
```

Input from keyboard and Write to file

Read from file and Print out



```
e
x/A2   f
xwA   g      è@d
```

4.8. Moving to a specific location in a file: fseek()

- The **fseek()** function in C is used to move the file pointer to a specific location in a file.
- It allows you to read or write from any position in a file, making it a powerful tool for random-access file operations
- **Syntax**:

```
int fseek(FILE *file_ptr, long int offset, int pos);
```

- **Return Value**:
  - Returns 0 on success.
  - Returns a nonzero value on failure.

fseek()

- **Parameters:**
  - file_ptr: A pointer to the file object (returned by **fopen()**).
  - offset: The number of bytes to move the file pointer.
  - pos: Determines the position from where the offset is applied. It can be:
    - **SEEK_SET**: Beginning of the file.
    - **SEEK_CUR**: Current position of the file pointer.
    - **SEEK_END**: End of the file.
- **Common Use Cases**:
  - Jump to a specific location in a file.
  - Skip over sections of a file.
  - Read or write from a specific position in a file.

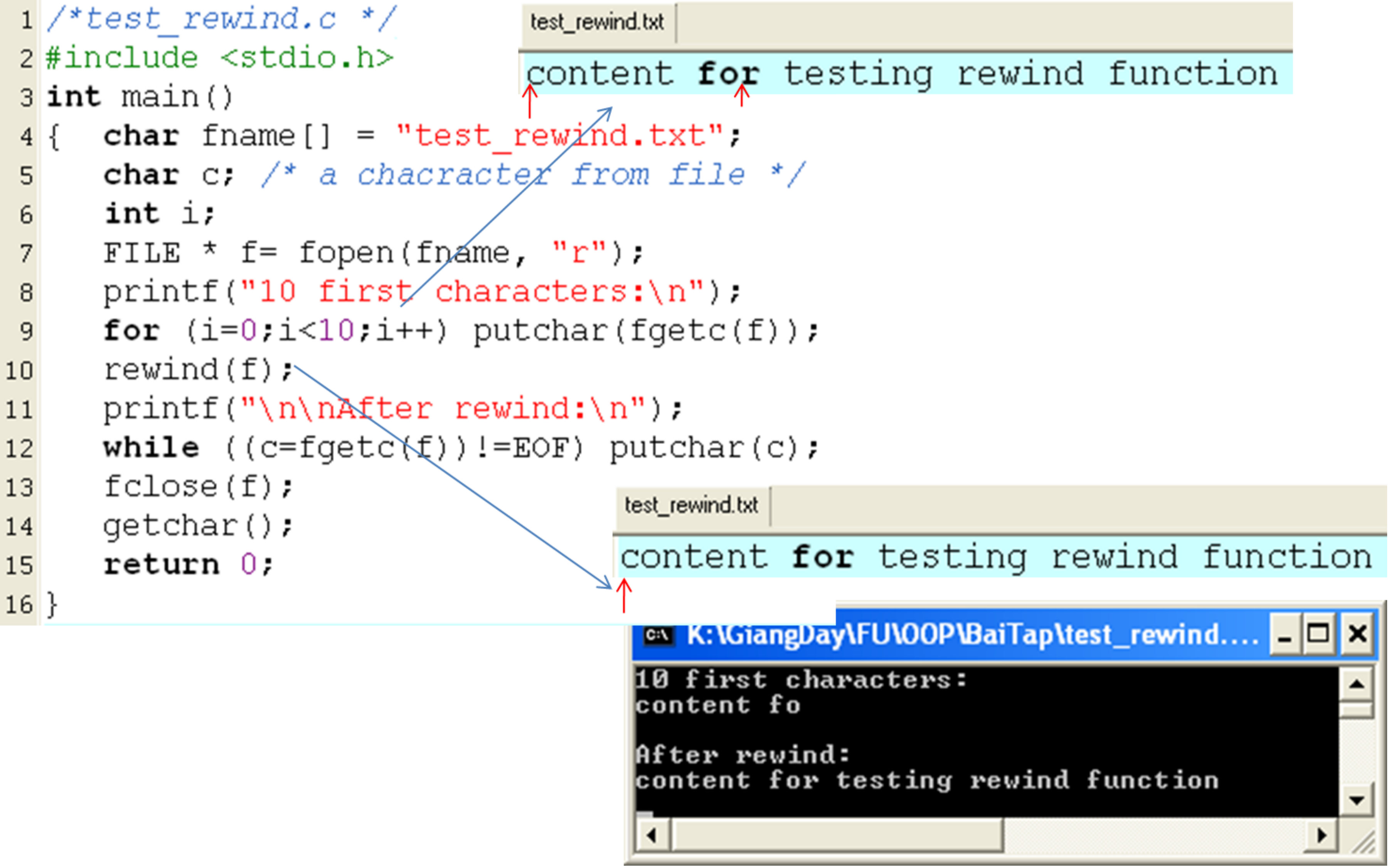
rewind() function

- The **rewind()** function in C is used to move the file pointer back to the beginning of a file.
- Unlike **fseek(),** which requires parameters, **rewind()** is a simple way to reset the file pointer to the start of a file.
- **Syntax**:

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

- **Parameter**:
  - file_pointer: A pointer to the file object, which must have been opened by **fopen()**
- **Return Value**: does not return a value

Demo: rewind(FILE*)



```c
/*test_rewind.c */
#include <stdio.h>
int main()
{   char fname[] = "test_rewind.txt";
    char c; /* a chacracter from file */
    int i;
    FILE * f= fopen(fname, "r");
    printf("10 first characters:\n");
    for (i=0;i<10;i++) putchar(fgetc(f));
    rewind(f);
    printf("\n\nAfter rewind:\n");
    while ((c=fgetc(f))!=EOF) putchar(c);
    fclose(f);
    getchar();
    return 0;
}
```

test_rewind.txt

```
content for testing rewind function
```

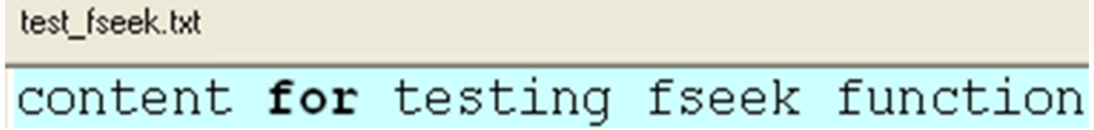
test_rewind.txt

```
content for testing rewind function
```

K:\GiangDay\FU\OOP\BaiTap\test_rewind....

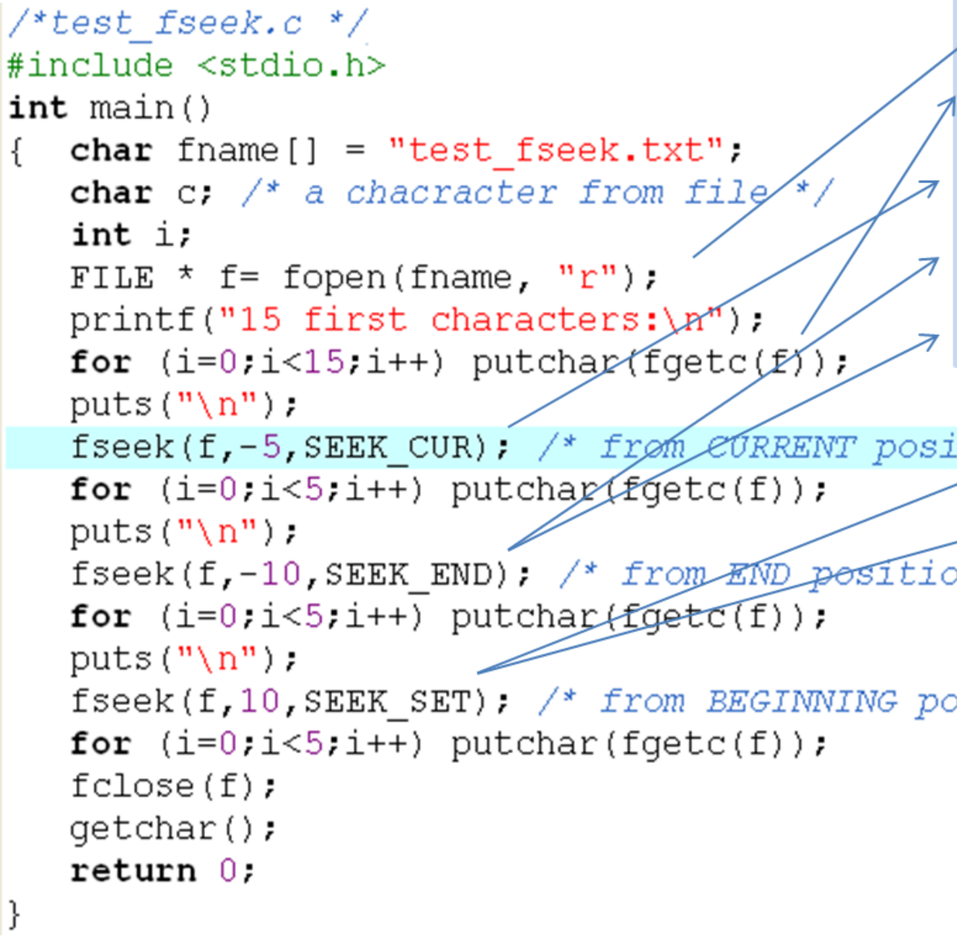
```
10 first characters:
content fo

After rewind:
content for testing rewind function
```

Demo: fseek(…)

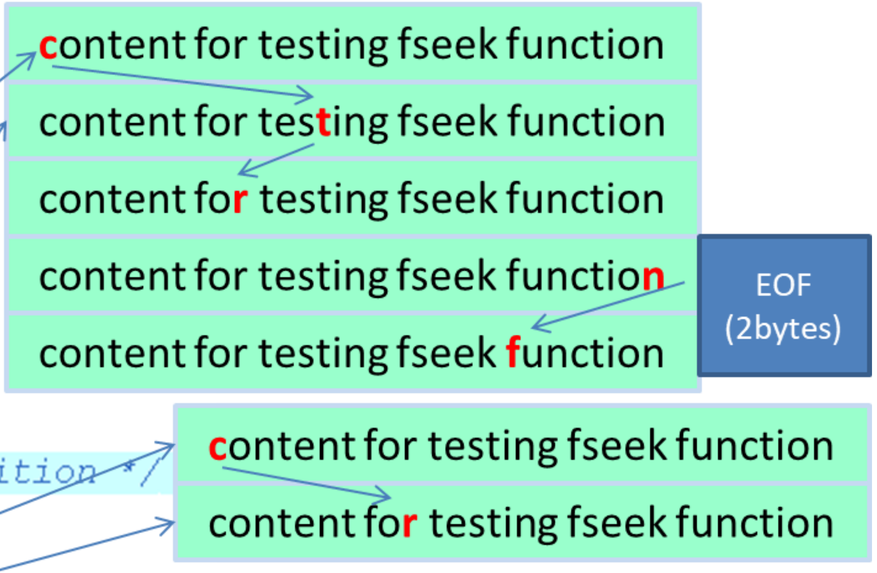


test_fseek.txt

```
content for testing fseek function
```



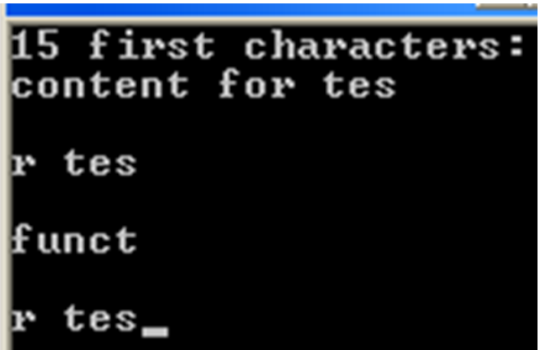
```
/*test_fseek.c */
#include <stdio.h>
int main()
{   char fname[] = "test_fseek.txt";
    char c; /* a chacracter from file */
    int i;
    FILE * f= fopen(fname, "r");
    printf("15 first characters:\n");
    for (i=0;i<15;i++) putchar(fgetc(f));
    puts("\n");
    fseek(f,-5,SEEK_CUR); /* from CURRENT position */
    for (i=0;i<5;i++) putchar(fgetc(f));
    puts("\n");
    fseek(f,-10,SEEK_END); /* from END position */
    for (i=0;i<5;i++) putchar(fgetc(f));
    puts("\n");
    fseek(f,10,SEEK_SET); /* from BEGINNING position */
    for (i=0;i<5;i++) putchar(fgetc(f));
    fclose(f);
    getchar();
    return 0;
}
```



content for testing fseek function
content for testing fseek function
content for testing fseek function
content for testing fseek function
content for testing fseek function

EOF
(2bytes)

content for testing fseek function
content for testing fseek function



```
15 first characters:
content for tes

r tes

funct

r tes
```

Summary

- Understand what a file is? The role of files in storing data on secondary memory.
- Types of Files in C
- Using C language to work with Files
- Open/Close files
- Read and write data to/from text files and binary files
- Move the pointer to work with data in files