File Handling in C

Lecture 10 – ICT1132

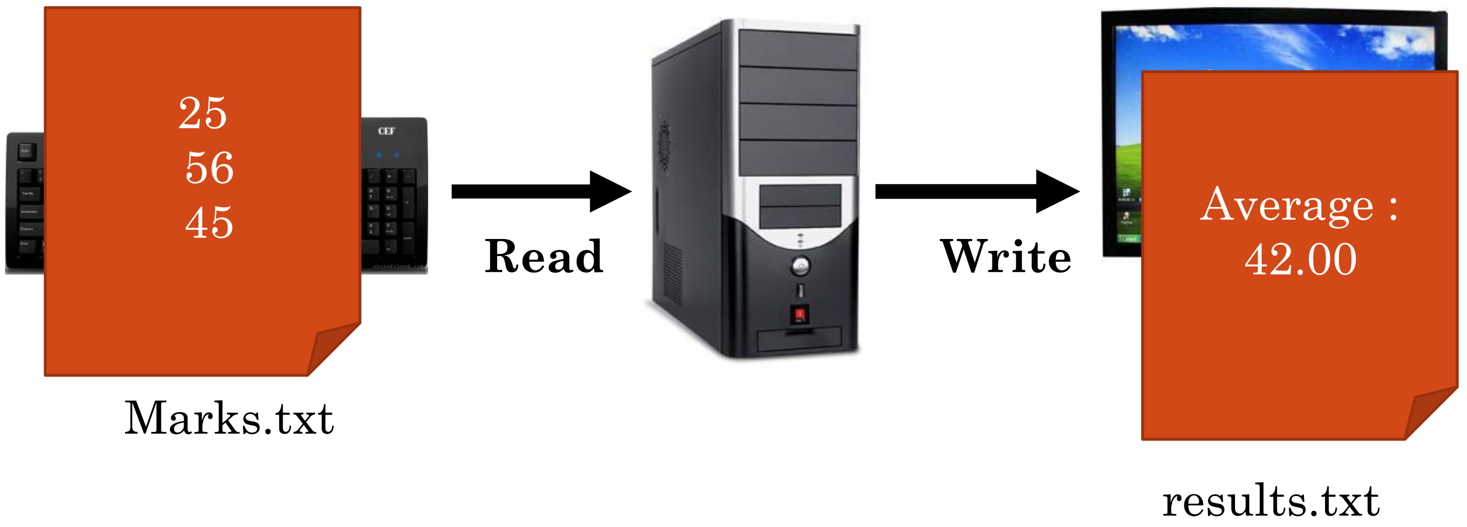
Piyumi Wijerathna
Department of ICT
Faculty of Technology

Overview

- What is a file?
- Files and Streams
- Sequential and Random File Access
- File Processing Operations
- End OF File (EOF)

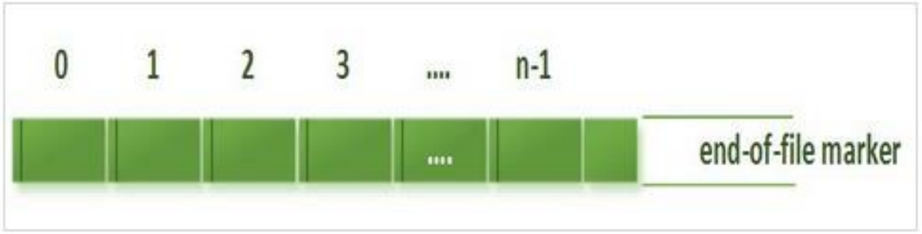
Why files are needed?

- Storage of data in variables and arrays is temporary.
- Data is lost when program terminates.
- Files are used to store data permanently.



Files in Computers

- A **file** is a sequence of bytes storing a group of related data on computer storage.
- Computers store files on secondary storage devices, such as hard drives, CDs, DVDs and flash drives.
- Almost all information stored in a computer must be in a file.
- Each file ends either with an end-of-file (EOF) marker or at a specific byte number recorded in the file system.



- There are two kinds of files in a system,
    - ***Text***
    - ***Binary***

Text Files

- Text files contain ASCII codes of digits, alphabetic and symbols.
- You will see all the contents within the file as plain text.
    - Take minimum effort to maintain.
    - Easily readable.
    - Provide least security.
    - Takes bigger storage space.

Binary Files

- Instead of storing data in plain text, binary files store data (Numbers, Programs, images etc.) in the binary form (0's and 1's).
  - Can hold higher amount of data.
  - Not easily readable.
  - Provides a better security than text files.
- Binary files will not be used in this course.

Inputs and Outputs

- In a typical program, we input data from the keyboard and output data to the monitor.
- We can use data files instead of the keyboard (input) and the monitor (output).
- A computer has only 1 keyboard and 1 monitor, so you don’t need to specify what keyboard, monitor you need to use.

Cont.

- But there are 1000s of files in the Hard Disk.
- When you use files, you need to tell the computer the following,
  - The file name.
  - The pointer associate with the file.
  - Whether you want to read data or store data.

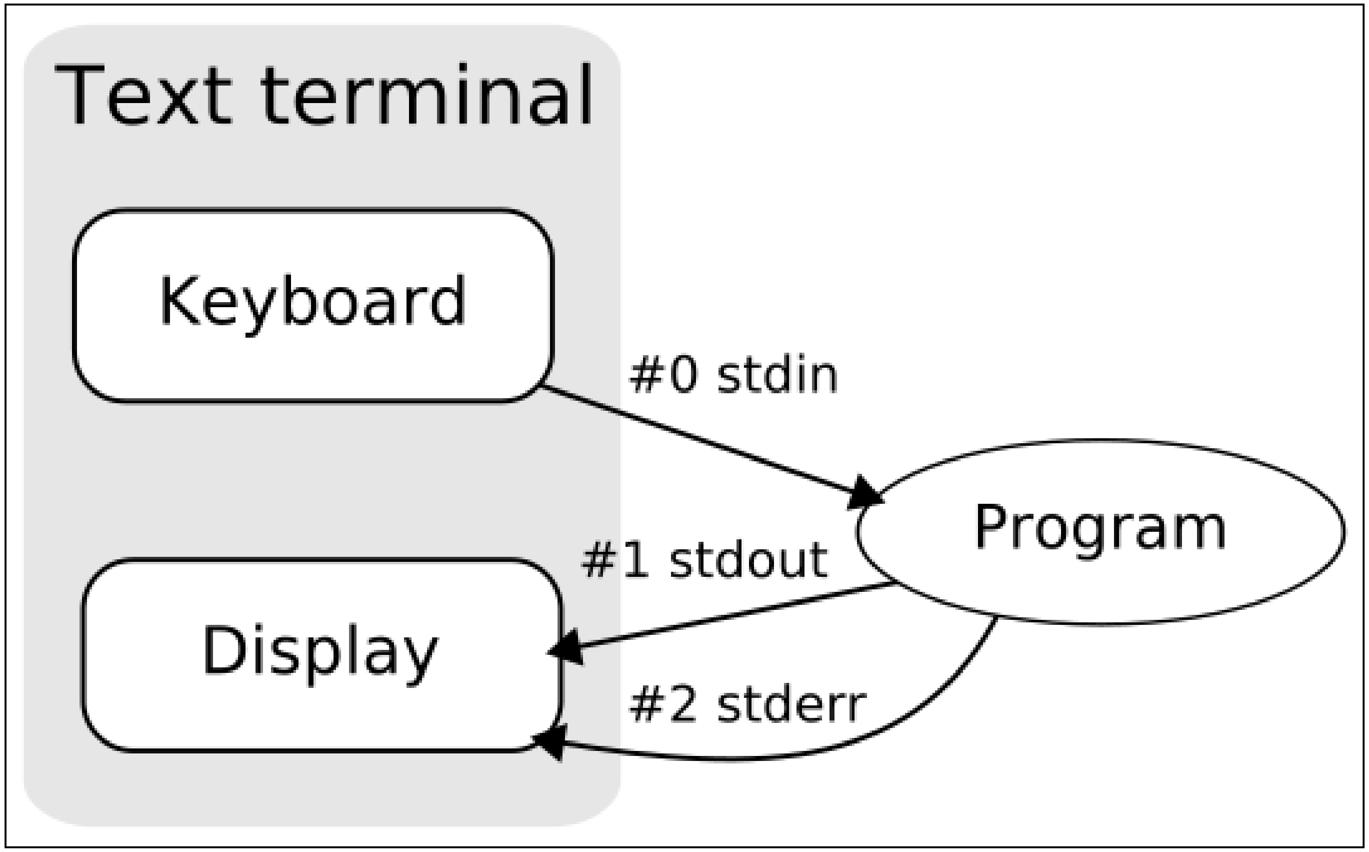
Files and Streams

Streams

- In C all input and output is done with streams.
- Stream is a sequence of bytes of data.
- A sequence of bytes flowing into program is called input stream.
- A sequence of bytes flowing out of the program is called output stream.
- When program execution begins, three associated streams are automatically opened,

  the standard input, the standard output and the standard error.

- **Ex:** The standard input stream enables a program to read data from the keyboard, and the standard output stream enables a program to print data on the screen.



Files and Streams (Cont.)

- C views each file simply as a sequential stream of bytes.
- A stream is associated with the file when it is opened.
- Streams provide communication channels between files and programs.
- Opening a file returns a pointer to a FILE structure (defined in <stdio.h>) that contains information used to process the file.
- The standard input, standard output and standard error are manipulated using pointers **stdin, stdout** and **stderr.**

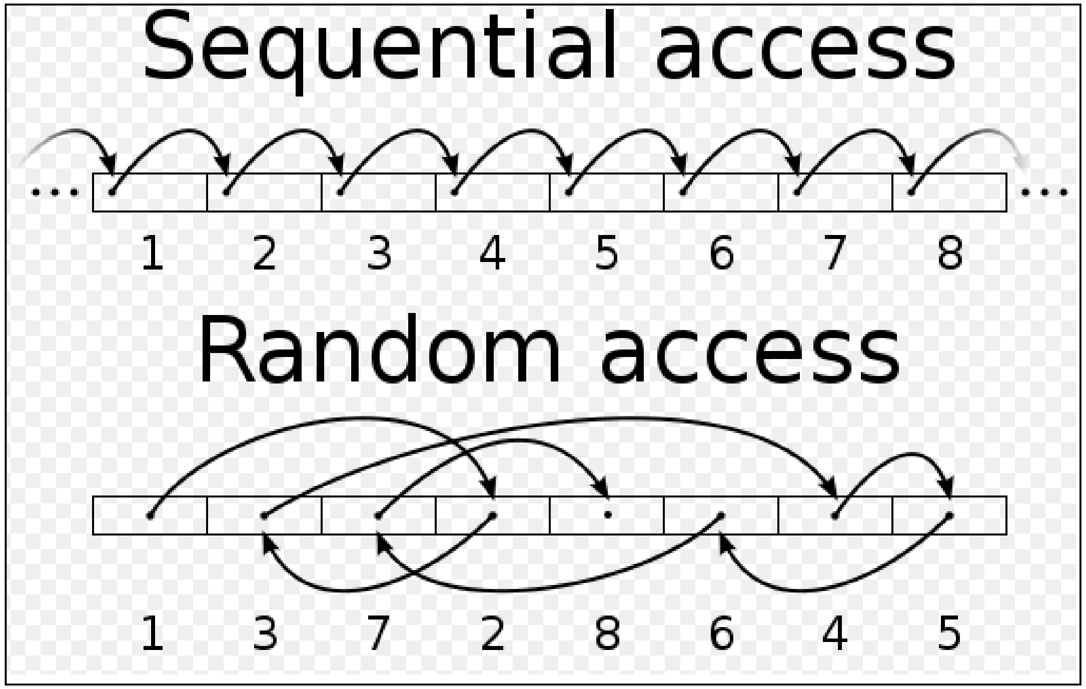
Sequential and Random Access File Handling in C

Sequential and Random Access

- In computer programming, the two main types of file handling are:

–Sequential access.

–Random access.



Sequential Access

- Program processes the data in a sequential fashion.

Random Access

- Only accesses the file at the point at which the data should be read or written, rather than processing it sequentially.

File Processing

File Operations

- Creating a new file
- Opening an existing file
- Reading from a file
- Writing to a file
- Closing a file

Functions for basic file operations

- **fopen()** - create a new file or open a existing file.
- **fclose()** - close an opened file.
- **fscanf()** - read from a file.
- **fprintf()** - write to a file.
- **getc()** - read a single character from file.
- **putc()** - write a character to the file.
- **fgets()** – read a line from the file.
- **fputs()** – write a line to the file.
- **fseek/fsetpos** - move a file pointer to somewhere in a file.
- **ftell/fgetpos** - tell you where the file pointer is located.

Steps of Processing a File

1. Declare a pointer of type FILE (defined in <stdio.h>). This pointer is needed for communication between the file and the program.

```
FILE *p;
```

2. Open the file using fopen (defined in <stdio.h>) ,by associating the stream name with the file name.

```
p = fopen("filename.txt","w");
```

3.Read or write the data.

4. Close the file.

Open Files

File Open

- Opening a file is performed using library function **fopen().**

  Syntax:

  FILE *filepointer;

  filepointer = `fopen(“filename”, “mode”);`

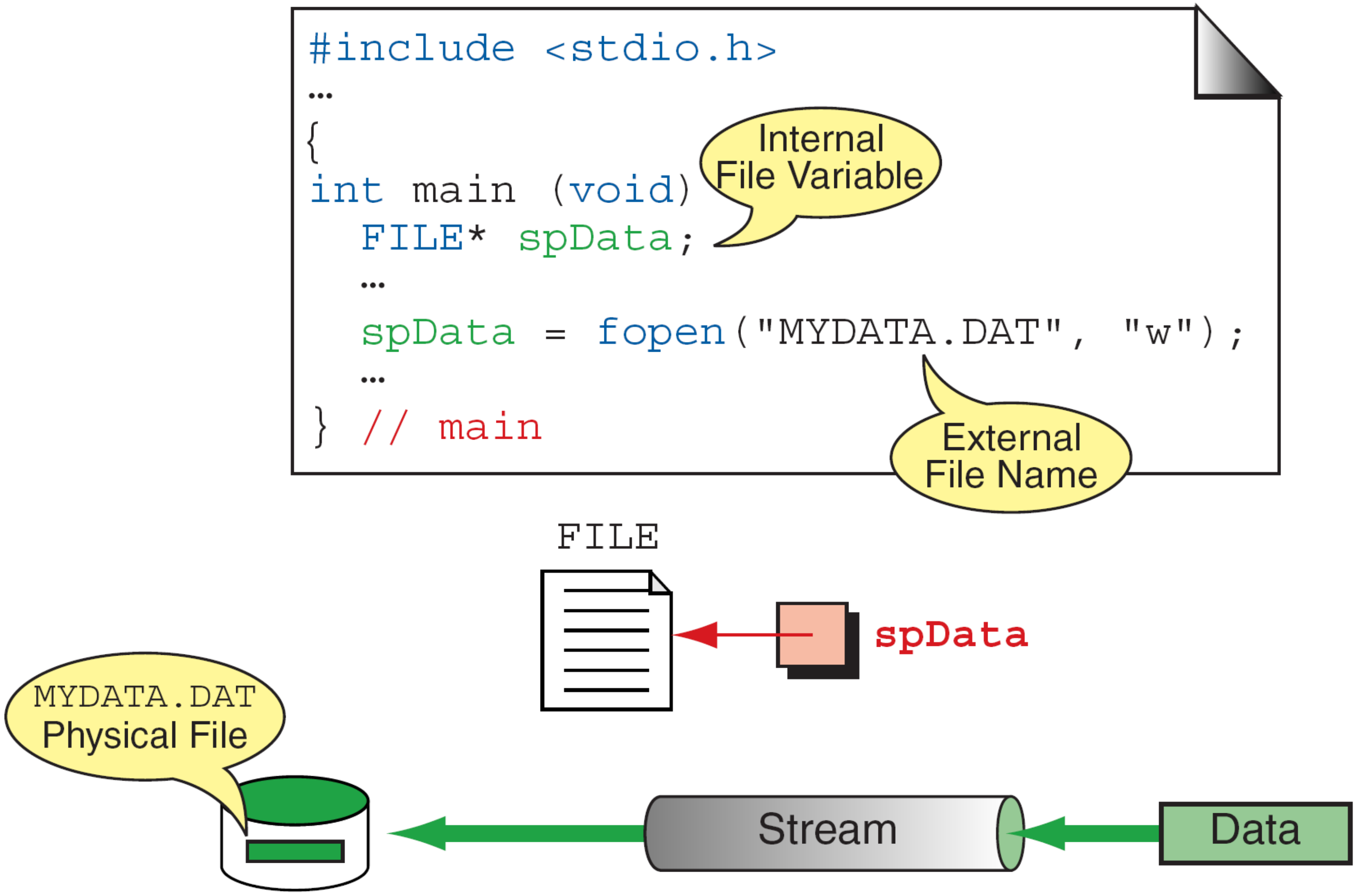
- Function fopen() takes two arguments:
  - The file open mode tells C how the program will use the file.
  - The filename indicates the system name and location for the file.

More on fopen()

- Each open file must have a **separately declared pointer** of type FILE that's used to refer to the file.
- We assign the return value of **fopen** to our pointer variable.

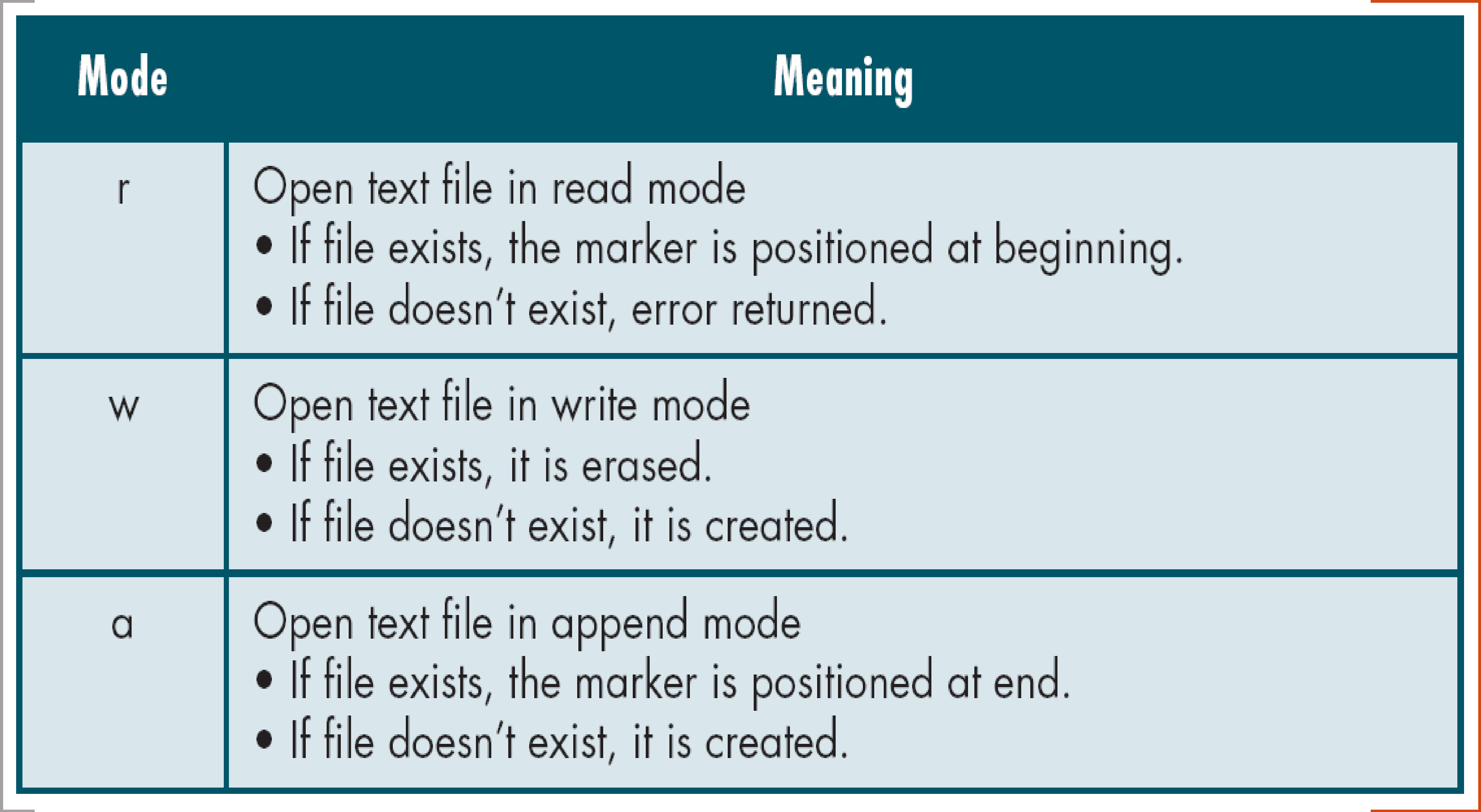
```
spData = fopen("myfile.txt", "w");

fptr = fopen("C:\\MYDATA\file1.dat", "w");
```

More on fopen



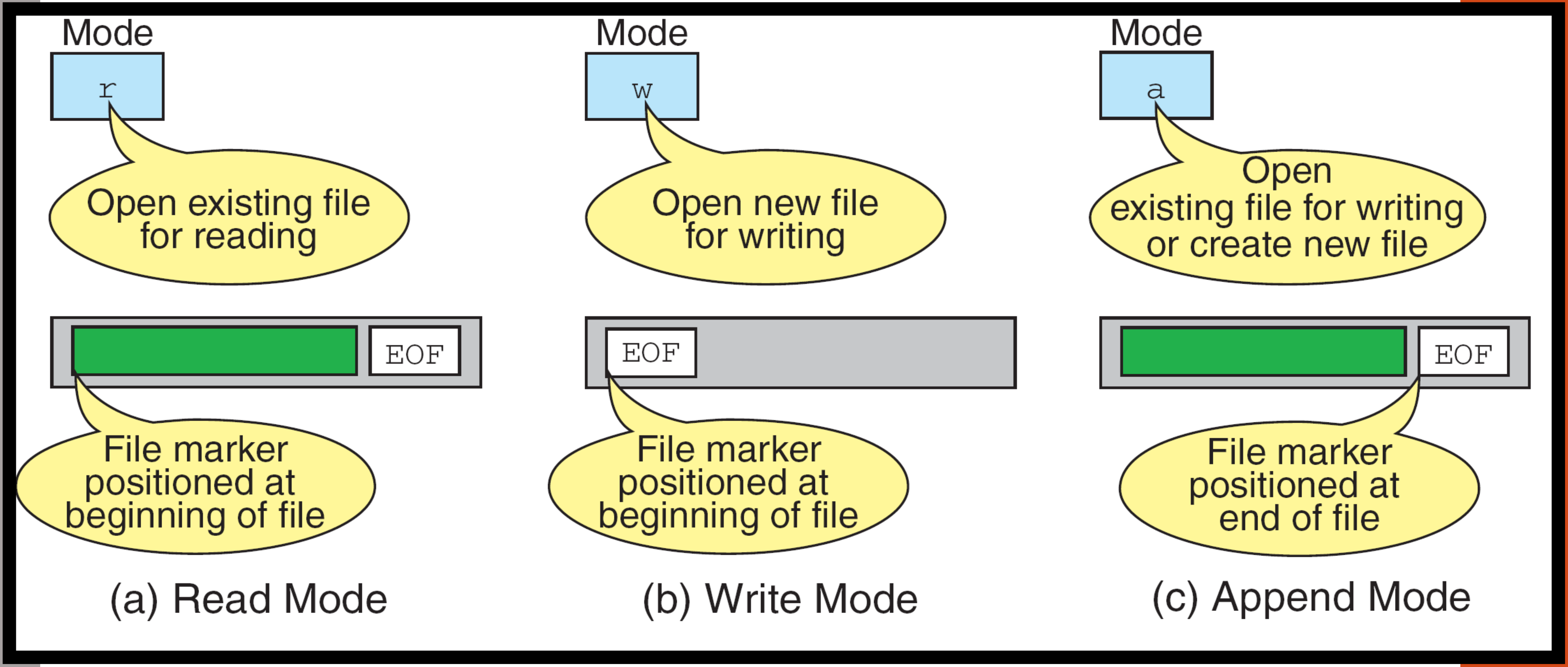
```
#include <stdio.h>
…
{
int main (void)
   FILE* spData;
   …
   spData = fopen("MYDATA.DAT", "w");
   …
} // main
```

File Open Modes



Mode	Meaning
r	Open text file in read mode • If file exists, the marker is positioned at beginning. • If file doesn't exist, error returned.
w	Open text file in write mode • If file exists, it is erased. • If file doesn't exist, it is created.
a	Open text file in append mode • If file exists, the marker is positioned at end. • If file doesn't exist, it is created.

More on File Open Modes


Mode
r
Open existing file for reading
EOF
File marker positioned at beginning of file
(a) Read Mode
Mode
w
Open new file for writing
EOF
File marker positioned at beginning of file
(b) Write Mode
Mode
a
Open existing file for writing or create new file
EOF
File marker positioned at end of file
(c) Append Mode

Additionally,

- **r+** - open for reading and writing, start at beginning
- **w+** - open for reading and writing (overwrite file)
- **a+** - open for reading and writing (append if file exists)

Difference between Append and Write Mode

- Both are used to write in a file. In both the modes, new file is created if it doesn't exists already.
- Open a file in the **write mode**, the file is reset, resulting in deletion of any data already present in the file.
- **Append mode** is used to append or add data to the existing data of file (if any).

Read Data from Files

Reading Data

- The stored data in files will be read by the program for processing when needed.
- Function ***fscanf()*** is used to read data and it receives a file pointer for the file being read.

fscanf()

- fscanf() is similar to scanf().
- However fscanf() require one additional argument as the first argument which is the pointer to the file.

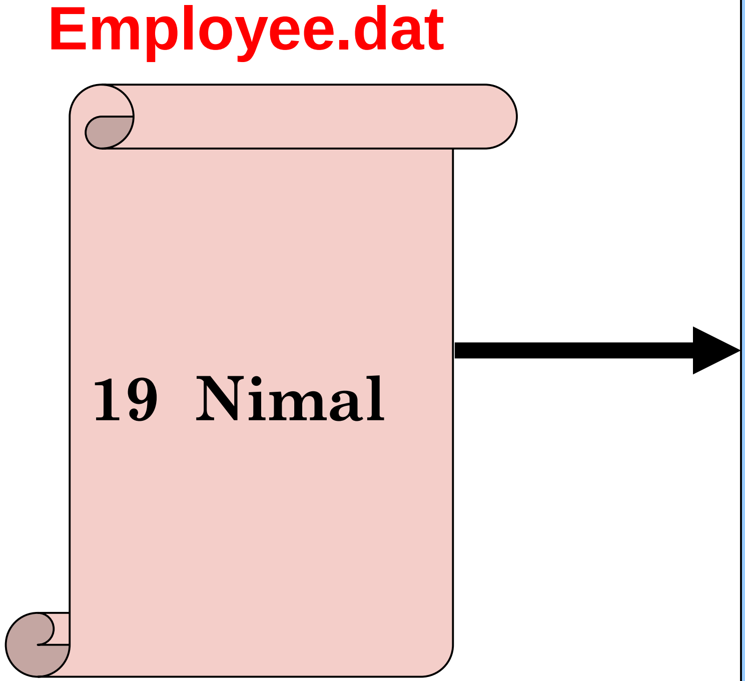
Syntax:

```
        fscanf (fp, “formats”, identifiers);
```

Example:

```
    FILE *fp;
    fp=fopen(“input.txt”,”r”);
    int i;
    fscanf (fp, “%d", &i);
```

Read Files for Inputting Data



```
int main() {
  FILE  *fp;
   int empno;
   char name[20];
  fp = fopen(“Employee.txt”,”r”);

  fscanf (fp, “%d”, &empno);
  fscanf (fp, “%s”, name);
  ….
}
```

This is very similar to inputting data from the keyboard.

Error Handling in Files

The fail method

- The fail method returns true when the file is in a fail state.
- Input files will fail if they **do not exist**, or **can not be located**.
- Output files will fail if the new **file cannot be created**, if the **disk drive is not operational** or the **disk is write protected or full.**
- C programming does not provide direct support for error handling.

Ex:-

```
#include <stdio.h>
#include <stdlib.h>
void main ()
{
   FILE *fp;
   fp = fopen("data.txt", "r");

   if (fp == NULL) {
      printf("File does not exist, please check!\n");
      exit (1);
   }
   fclose(fp);
}
```

Note: Use of exit(1) :- Terminate running the program.

exit(1) is defined in **`<stdlib.h>`** *header file.*

Write Data into Files

Writing Data

- The processed data in programs will be written to the files when needed.
- Function ***fprintf()*** is used to write data and it receives a file pointer for the file being written.

fprintf()

- fprintf() is similar to printf().
- However fprintf() require one additional argument as the first argument which is the pointer to the file.

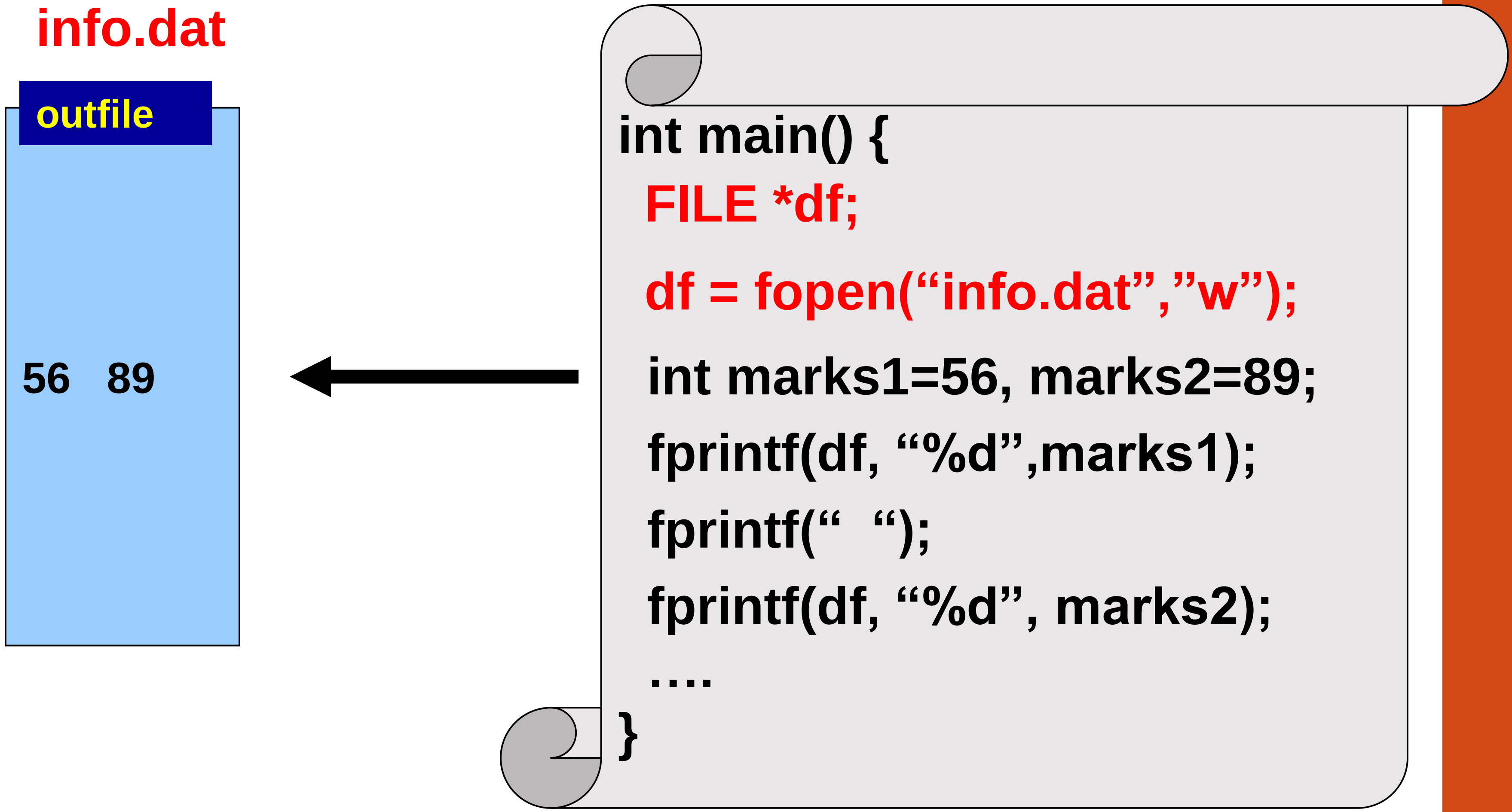
Syntax:

```
        fprintf (fp, format, variables);
```

Example:

```
    int i = 12;
    float x = 2.356;
    char ch = 's';

    FILE *fp;
    fp=fopen(“out.txt”,”w”);
    fprintf (fp, "%d %f %c", i, x, ch);
```

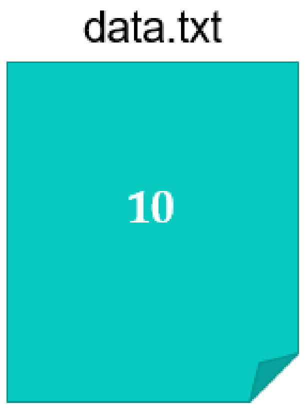
Using Files for Storing Data



```
int main() {
  FILE *df;
  df = fopen(“info.dat”,”w”);
  int marks1=56, marks2=89;
  fprintf(df, “%d”,marks1);
  fprintf(“  “);
  fprintf(df, “%d”, marks2);
  ….
}
```

Writing Data to A Sequential - Access File

```
#include <stdio.h>
int main(void)
{
    int number = 10;

    FILE *cfPtr;
    cfPtr = fopen(“data.txt”, “w”);

    if ( cfPtr == NULL)
            printf(“Cannot create file\n”);
    else
            fprintf(cfPtr, “%d\n”, number);

    fclose(cfPtr);
    return 0;
}
```



Close each file as it is no longer needed.

Close Files

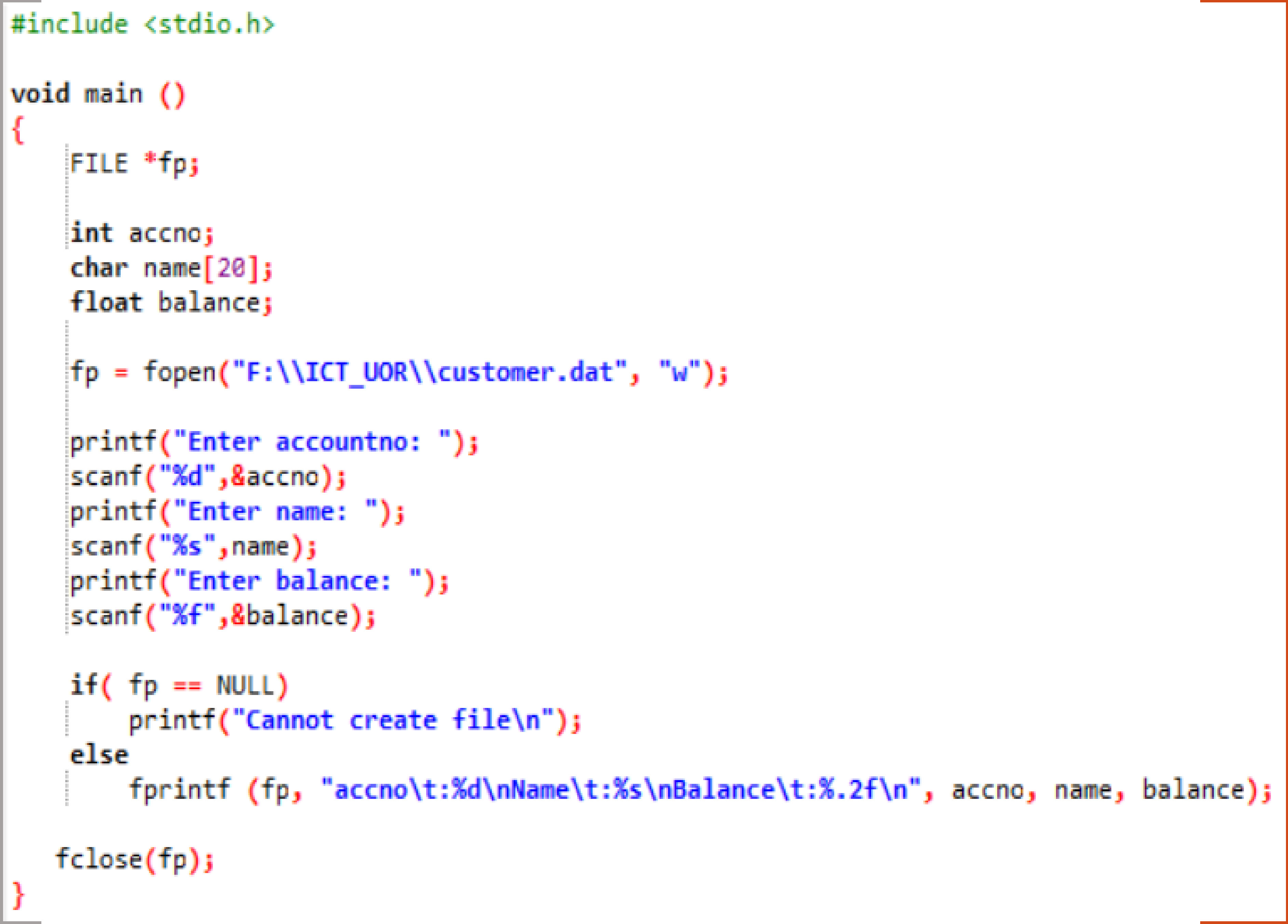
Closing a File

- When we finish with a mode, we need to close the file before ending the program or beginning another mode with that same file.
- To close a file, we use **`fclose()`** and the pointer variable.

```
    fclose(spData);
```

Exercise

- Write a program to input the account number, name and account balance of a bank customer from the keyboard and write the data to “customers.dat” file.



```
#include <stdio.h>

void main ()
{
    FILE *fp;

    int accno;
    char name[20];
    float balance;

    fp = fopen("F:\\ICT_UOR\\customer.dat", "w");

    printf("Enter accountno: ");
    scanf("%d",&accno);
    printf("Enter name: ");
    scanf("%s",name);
    printf("Enter balance: ");
    scanf("%f",&balance);

    if( fp == NULL)
        printf("Cannot create file\n");
    else
        fprintf (fp, "accno\t:%d\nName\t:%s\nBalance\t:%.2f\n", accno, name, balance);

   fclose(fp);
}
```

Data File with Multiple Values

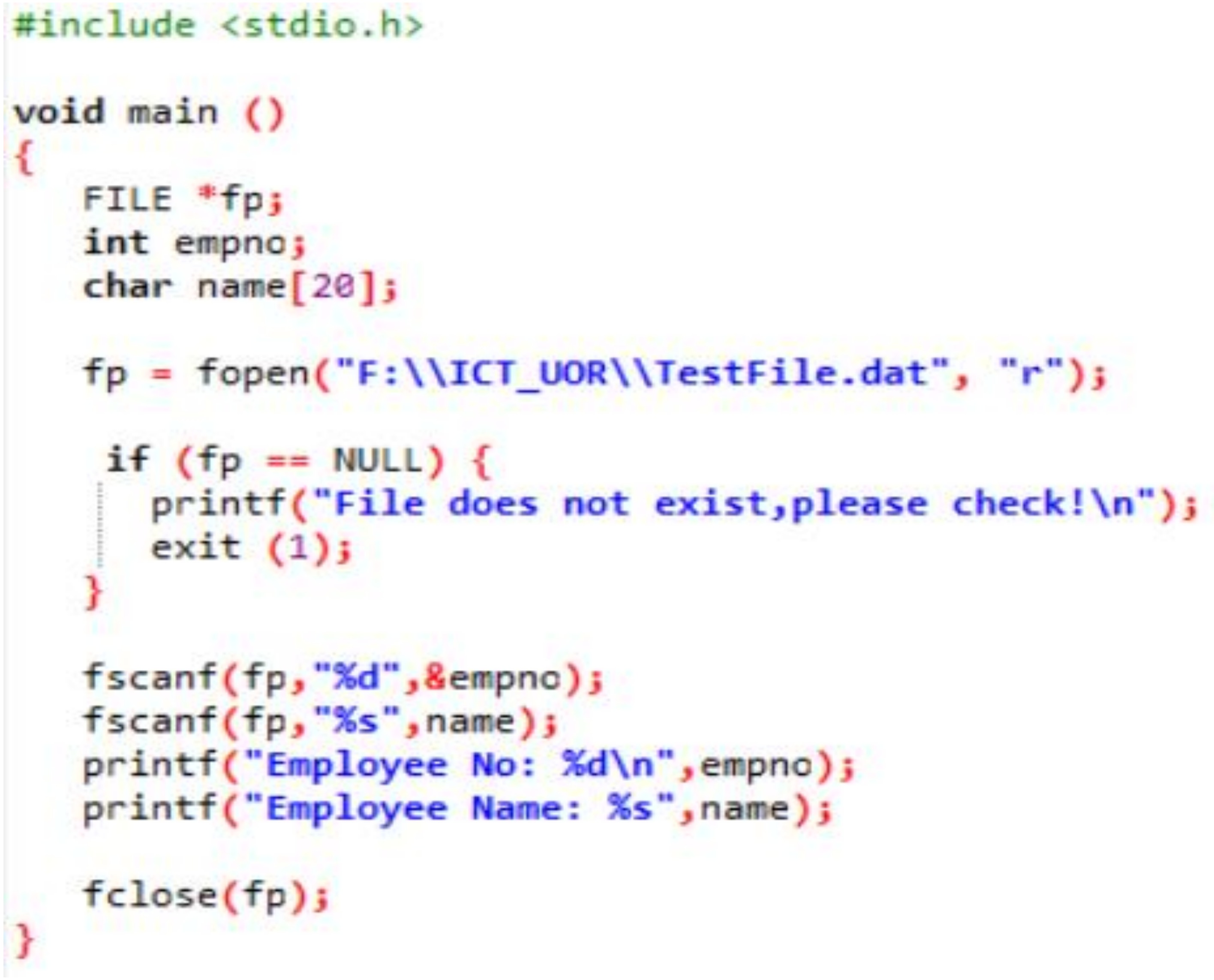
- Whitespaces (spaces, tabs, newline) should be used to separate multiple values stored in an input file.

e.g.

```
Nimal  90  34 22
Kamal  22  33 99
```

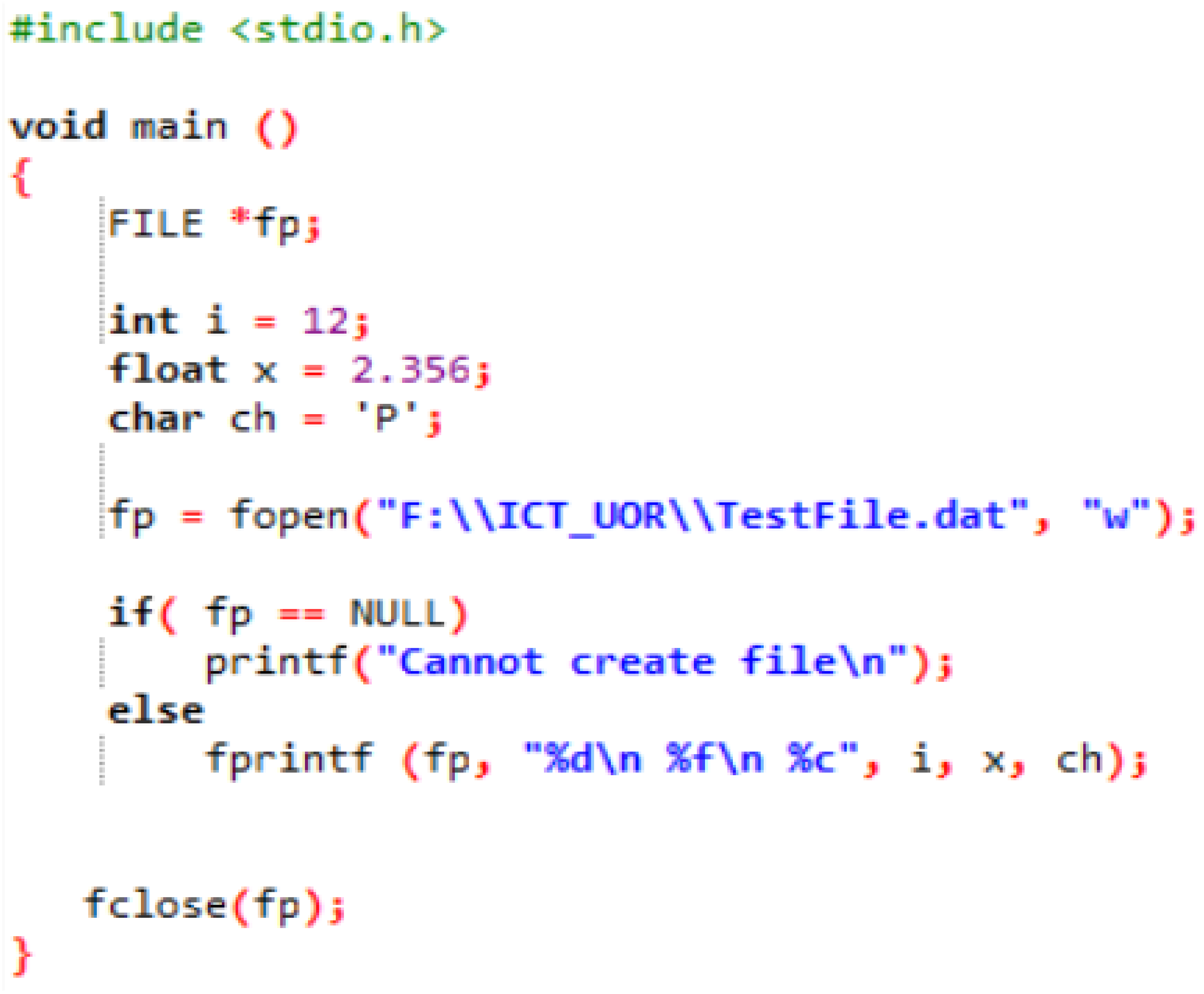
Examples

Reading from a File



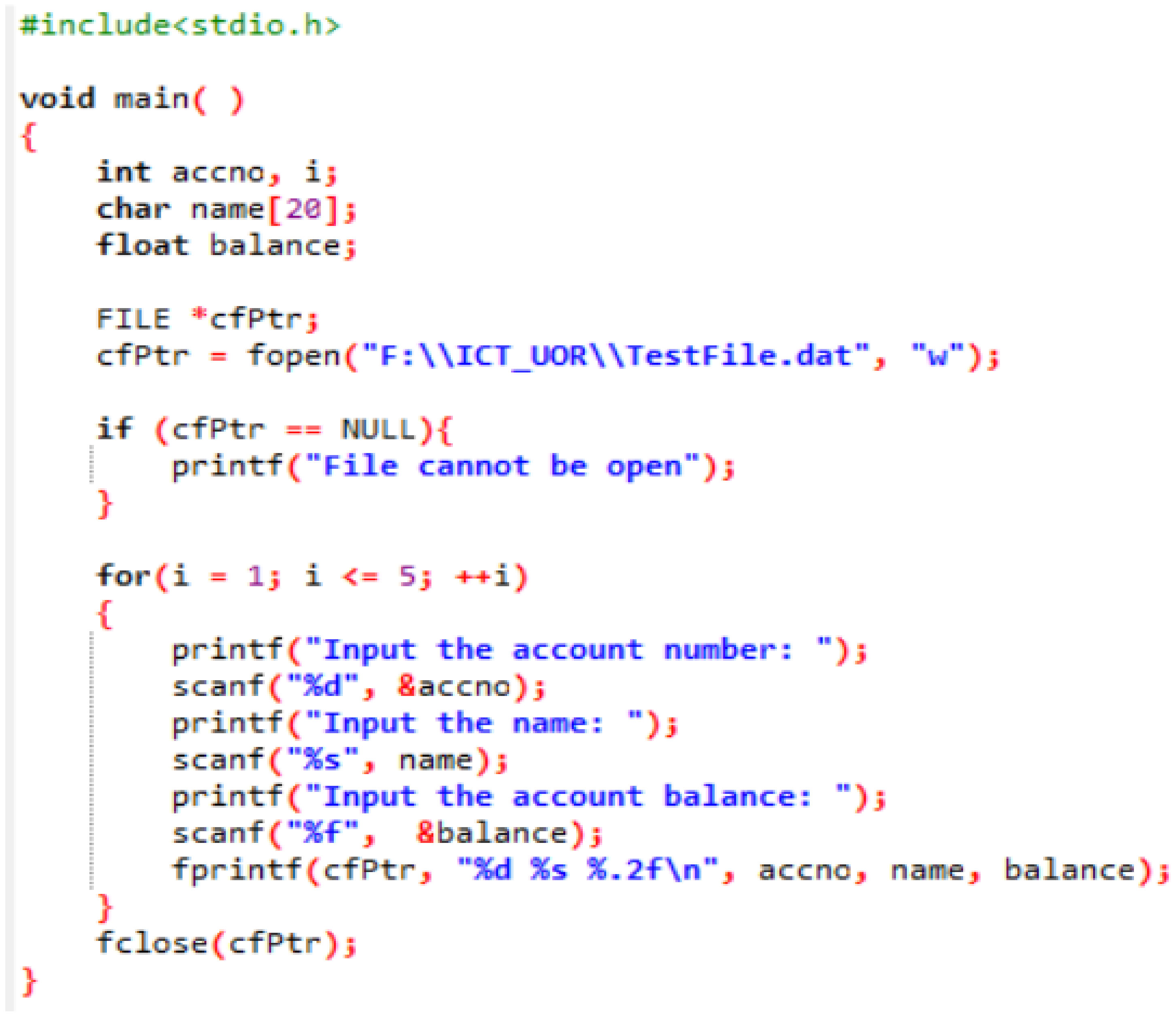
```
#include <stdio.h>

void main ()
{
    FILE *fp;
    int empno;
    char name[20];

    fp = fopen("F:\\ICT_UOR\\TestFile.dat", "r");

     if (fp == NULL) {
       printf("File does not exist,please check!\n");
       exit (1);
    }

    fscanf(fp,"%d",&empno);
    fscanf(fp,"%s",name);
    printf("Employee No: %d\n",empno);
    printf("Employee Name: %s",name);

    fclose(fp);
}
```

Writing to a File



```
#include <stdio.h>

void main ()
{
    FILE *fp;

    int i = 12;
    float x = 2.356;
    char ch = 'P';

    fp = fopen("F:\\ICT_UOR\\TestFile.dat", "w");

    if( fp == NULL)
        printf("Cannot create file\n");
    else
        fprintf (fp, "%d\n %f\n %c", i, x, ch);


   fclose(fp);
}
```

Writing Multiple Records to a File



```
#include<stdio.h>

void main( )
{
    int accno, i;
    char name[20];
    float balance;

    FILE *cfPtr;
    cfPtr = fopen("F:\\ICT_UOR\\TestFile.dat", "w");

    if (cfPtr == NULL){
        printf("File cannot be open");
    }

    for(i = 1; i <= 5; ++i)
    {
        printf("Input the account number: ");
        scanf("%d", &accno);
        printf("Input the name: ");
        scanf("%s", name);
        printf("Input the account balance: ");
        scanf("%f",  &balance);
        fprintf(cfPtr, "%d %s %.2f\n", accno, name, balance);
    }
    fclose(cfPtr);
}
```

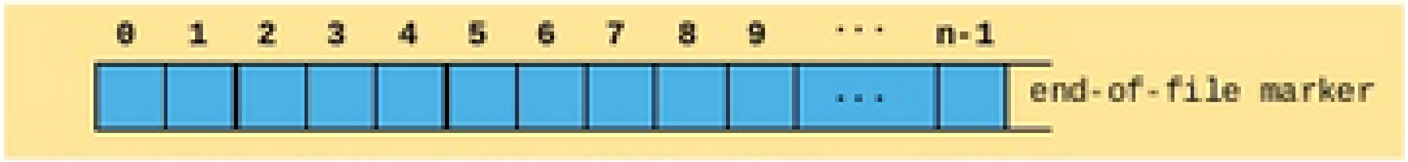
End Of File (EOF)

The end-of-file marker (EOF)

- EOF is not a character, but a signal which indicates that there are no more characters available.
- The end-of-file marker is placed on a file when it is closed.
- Input(File Reading) will continued normally until the end-of-file marker is detected.
- In C, EOF = -1. It is not a character. Because no ASCII value for -1.
- The end-of-file indicator informs the program that there's no more data to be processed.

Printing Value of EOF:

```
void main()
{
		printf("%d", EOF);
}
```



The end-of-file marker (EOF)

- C provides feof() which returns non-zero value only if end of file has passed, otherwise it returns 0.

```
if( feof(fptr) != 0 )                    // as if(1) is TRUE
        printf("End of File");
```

In while Loop

```
while(! feof(fptr)){
--- - --
--- - --
}
```

Do- while Loop

```
do{
--- - --
--- - --
} while(! feof(fptr))
```

Most suitable way

Character Handling in a File

getc()

- Reads a single character from input streams – from a file or standard input.
- depends on the argument pointer.

Syntax:

```
identifier = getc (file pointer);
identifier = getc(stdin);
```

Example:

```
FILE *fp;
fp=fopen(“input.txt”,”r”);
char ch;
ch = getc (fp);
```

putc()

- Write a single character to the output, pointed to by fp.

(or to stdout/screen, pointed by stdout)

Syntax:

putc (character variable, file pointer);

Example:

```
FILE *fp;
char ch = ‘A’;
putc (ch, fp);
```

Example: putc() and getc()

```
FILE *fp;
char ch;
fp = fopen("one.txt", "w");
if(fp==NULL){
        printf("File does not created!!!");
        exit(0);        /*exit   from program*/
}
printf("Enter data");
while((ch = getchar()) != EOF) {
        putc(ch,fp);
}
fclose(fp);
fp = fopen("one.txt", "r");
while( (ch = getc(fp)) != EOF)
        printf("%c",ch);
fclose(fp);
```

if we successfully get a character and assign to C , returned status code is 0, failed is -1. EOF is defined as -1. Therefore when condition -1 == -1 occurs, loops stops

Press *ctrl+c* together to stop entering data

fread()

Declaration:

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

Remarks:

- fread reads a specified number of equal-sized data items from an input stream **into a block**.

(reads data from the given **stream** into the array pointed to, by **ptr**.)

ptr	= Points to a block into which data is read
size	= Length of each item read, in bytes
n	= Number of items read
stream	= file pointer

Example

```
#include <stdio.h>
int main()
{
  FILE *f;
  char buffer[11];
  if (f = fopen(“readme.txt", “r”))
  {
       fread(buffer, 1, 10, f);

       fclose(f);
       printf("first 10 characters of the file:\n%s\n",
  buffer);
   }
 return 0;
}
```

Equals to (f != NULL)

fwrite()

Declaration:

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

Remarks:

- fwrite **writes** specified number of equal-sized data items to an output file.
- writes data from the array pointed by ptr to the given stream.

ptr = Pointer to any object; the data written begins at ptr

size = Length of each item of data

n = Number of data items to be appended

stream = file pointer

Example

```
#include <stdio.h>
int main()
{
   char a[10]={'1','2','3','4','5','6','7','8','9','a'};
   FILE *fs;
   fs=fopen(“readme.txt","w");
   fwrite(a,1,10,fs);
   fclose(fs);
   return 0;
}
```

getw() and putw() functions

getw()	To read an integer from a file.	getw(fp)
putw()	To write an integer into a file.	putw(integer, fp)

Example

```
FILE *fp;
int num;
char ch='n';
fp = fopen("file.txt","w");
if(fp==NULL){
        printf("Can not open file or file does not exist");
        return -1;
}
do{
        printf("\nEnter any number: ");
        scanf("%d",&num);

        //fprintf(fp,"%d ",num);
        putw(num,fp);

        printf("\nDo you want to get another number? ");
        ch = getche();
}while(ch=='y'||ch=='Y');

printf("\nData written successfully");
fclose(fp);
```

```
Output :
   Enter any number : 78
   Do you want to another number : y
   Enter any number : 45
   Do you want to another number : y
   Enter any number : 63
   Do you want to another number : n
   Data written successfully...
```

fgets() and fputs() functions

fgets()	Reads string from a file, one line at a time.	fgets(arr, n, fp)
fputs()	writes a string (a line) into a file pointed by fp	fputs(arr, fp)

- **arr** – buffer to put the data in (a char array)
- **n** – size of the buffer (max number of characters can store in the array)
- **fp** – file pointer

Any
Question

THANK YOU… !