Lecture 13
File Handling in C

What is a File?

- A *file* is a collection of related data that a computers treats as a single unit.
- Computers store files to secondary storage so that the contents of files remain intact when a computer shuts down.
- When a computer reads a file, it copies the file from the storage device to memory(RAM);
- When it writes to a file, it transfers data from memory (RAM) to the storage device.

Why files are needed?

1. When a program is terminated, the entire data is lost. Storing in a file will preserve your data even if the program terminates.
2. If you have to enter a large number of data, it will take a lot of time to enter them all.
3. However, if you have a file containing all the data, you can easily access the contents of the file using few commands in C.
4. You can easily move your data from one computer to another without any changes.

Types of Files

When dealing with files, there are two types of files you should know about:

1. Text files

2. Binary files

Text Files

- Text files are the normal .txt files that you can easily create using Notepad or any simple text editors.
- When you open those files, you'll see all the contents within the file as plain text.
- You can easily edit or delete the contents.
- They take minimum effort to maintain,
  - easily readable,
  - provide least security
  - and takes bigger storage space.

Binary Files

- Binary files are mostly the .bin files in your computer.
- Instead of storing data in plain text, they store it in the binary form (0's and 1's).
- They can hold higher amount of data,
  - not readable easily and
  - provides a better security than text files.

File Operations

- In C, you can perform four major operations on the file, either text or binary:
  1. Creating a new file
  2. Opening an existing file
  3. Closing a file
  4. Reading from and writing information to a file

Working with files

- When working with files, you need to declare a pointer of type file.
- This declaration is needed for communication between the file and program.
- C uses a structure called **`FILE`** defined in **`stdio.h`** to store the attributes of a file.
- Example:

```
FILE *fptr;
```

Steps in Processing a File

1) Create the stream via a pointer variable using the **`FILE`** structure:

```
FILE *p;
```

2) Open the file, associating the stream name with the file name.

3) Read or write the data.

4) Close the file.

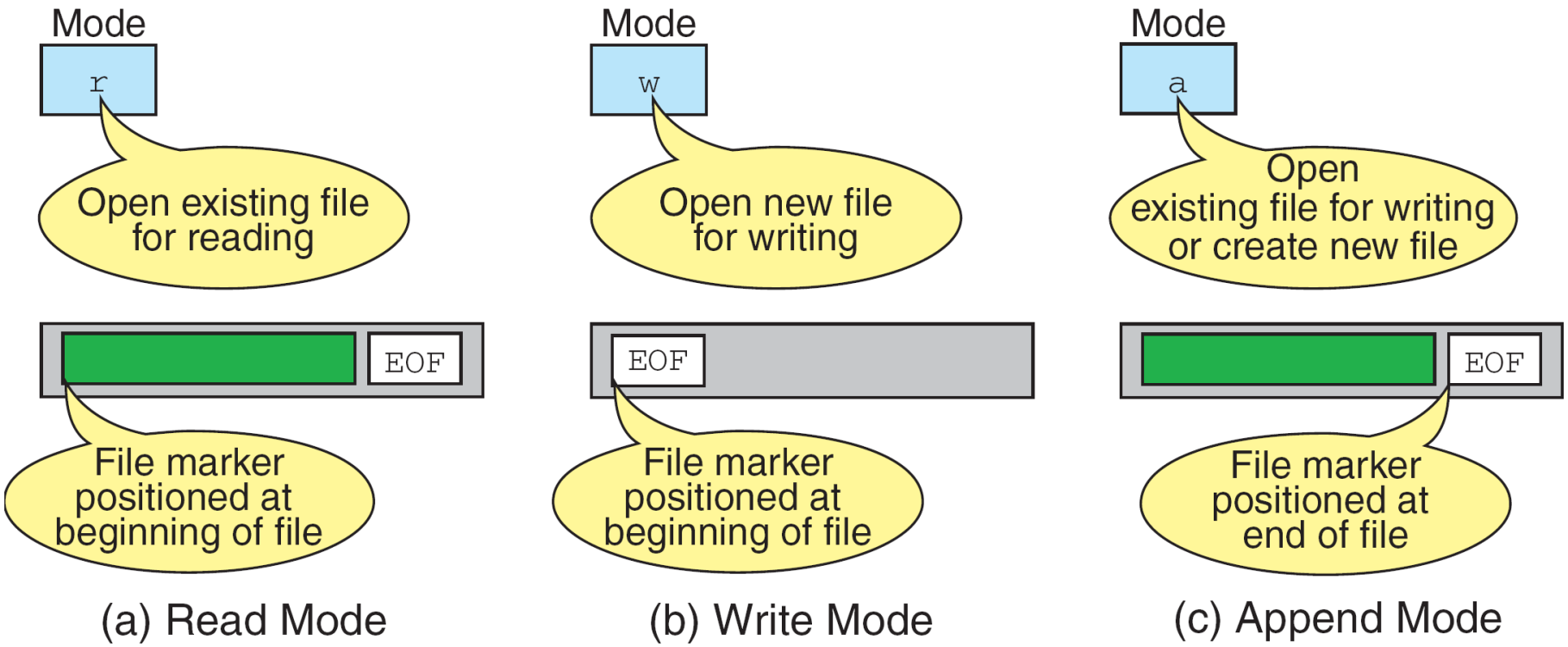
The basic file operations are

- fopen - open a file- specify how its opened (read/write) and type (binary/text)
- fclose - close an opened file
- fread - read from a file
- fwrite - write to a file
- fseek/fsetpos - move a file pointer to somewhere in a file.
- ftell/fgetpos - tell you where the file pointer is located.

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



from Figure 7-4 in* Forouzan & Gilberg, *p. 401

Opening a file - for creation and edit

Opening a file is performed using the library function in the "stdio.h" header file: fopen().

The syntax for opening a file in standard I/O is:

```
ptr = fopen("filename","mode")
```

For Example:

```
fopen("E:\\cprogram\\newprogram.txt","w");

fopen("E:\\cprogram\\oldprogram.bin","rb");
```

Opening a file - for creation and edit cont.

- Let's assume the file `newprogram.txt` doesn't exist in the location E:\cprogram.
  - The first function creates a new file named `newprogram.txt` and opens it for writing as per the mode 'w'.
  - The writing mode allows you to create and edit (overwrite) the contents of the file.
- If the second binary file `oldprogram.bin` exists in the location E:\cprogram.
  - The second function opens the existing file for reading in binary mode 'rb'.
  - The reading mode only allows you to read the file, you cannot write into the file.

File Mode	Meaning of Mode	During Inexistence of file
r	Open for reading.	If the file does not exist, fopen() returns NULL.
rb	Open for reading in binary mode.	If the file does not exist, fopen() returns NULL.
w	Open for writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
wb	Open for writing in binary mode.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
a	Open for append. i.e, Data is added to end of file.	If the file does not exists, it will be created.
ab	Open for append in binary mode. i.e, Data is added to end of file.	If the file does not exists, it will be created.

File Mode	Meaning of Mode	During Inexistence of file
r+	Open for both reading and writing.	If the file does not exist, fopen() returns NULL.
rb+	Open for both reading and writing in binary mode.	If the file does not exist, fopen() returns NULL.
w+	Open for both reading and writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
wb+	Open for both reading and writing in binary mode.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
a+	Open for both reading and appending.	If the file does not exists, it will be created.
ab+	Open for both reading and appending in binary mode.	If the file does not exists, it will be creat

Closing a File

The file (both text and binary) should be closed after reading/writing.

- Closing a file is performed using library function fclose().

```
fclose(fptr);
```

- `fptr` is the file pointer associated with file to be closed.

Reading and writing to a text file

For reading and writing to a text file, we use the functions

- `fprintf()`
- and `fscanf()`.

They are just the file versions of `printf()` and `scanf()`. The only difference is that, `fprint` and `fscanf` expects a pointer to the structure `FILE`.

Writing to a text file

Example 1: Write to a text file using fprintf()

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

int main()
{
    int num;
    FILE *fptr;
    fptr = fopen("C:\\program.txt","w");

    if(fptr == NULL)
    {
        printf("Error!");
        exit(1);
    }

    printf("Enter num: ");
    scanf("%d",&num);

    fprintf(fptr,"%d",num);
    fclose(fptr);

    return 0;
}
```

Reading from a text file

Example 2: Read from a text file using fscanf()

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

int main()
{
    int num;
    FILE *fptr;

    if ((fptr = fopen("C:\\program.txt","r")) == NULL){
        printf("Error! opening file");

        // Program exits if the file pointer returns NULL.
        exit(1);
    }

    fscanf(fptr,"%d", &num);

    printf("Value of n=%d", num);
    fclose(fptr);

    return 0;
}
```

fgetc()

Read a single character from the input file, pointed to by fp.

Syntax:

identifier = getc (file pointer);

Example:

```
FILE *fp;
fp=fopen("input.txt","r");
char ch;
ch = fgetc (fp);
```

fputc()

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

Example:

```
FILE *fp;
char ch;
fputc (ch,fp);
```

Reading and Writing Files

```
#include <stdio.h>
int main ( )
{
    FILE *outfile, *infile ;
    int b = 5, f ;
    float a = 13.72, c = 6.68, e, g ;
    outfile = fopen ("testdata.txt", "w") ;
    fprintf (outfile, " %f %d %f ", a, b, c) ;
    fclose (outfile) ;
    infile = fopen ("testdata.txt", "r") ;
    fscanf (infile,"%f %d %f", &e, &f, &g) ;
    printf (" %f %d %f \n ", a, b, c) ;
    printf (" %f %d %f \n ", e, f, g) ;
}
```

Example(Character by Character)

```
#include <stdio.h>
void main()
{
 char ch;
 FILE *fp;
 fp=fopen("out.txt","r");
 while(!feof(fp))
  {
  ch=getc(fp);
  printf("\n%c",ch);
 }
}
```

Example(Input Line)

```
#include <stdio.h>
void main()
 {
    char line[500];
       FILE *fp;
       fp=fopen("out.txt","r");
       while(!feof(fp))
       {
           fgets(line,5,fp);
           printf("%s\n",line);
       }
 }
```