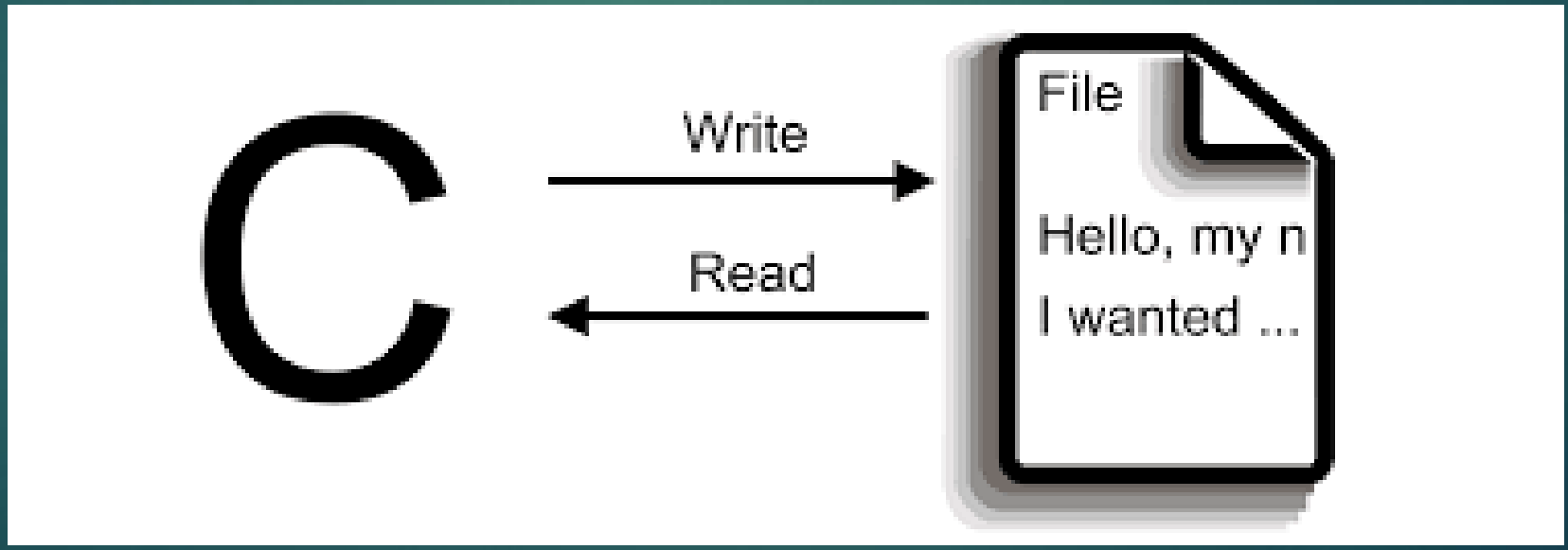
FILE HANDLING



INTRODUCTION TO FILES

- Whenever we need to store a large amount of data, we use arrays or structures.
- Any array is allocated with sequential memory bytes.
- However, once the program finishes execution, all the data will we lost as it is stored in main memory i.e volatile memory.
- The above will not work for real-life applications
- Thus we need files.
- A **file** represents a sequence of bytes on the disk where a group of related data is stored.
- The disk is permanent storage that continues to exist even after the program finishes execution.
- We can also move files from one computer to another.

TYPES OF FILES

- 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 a higher amount of data
  - Are not readable easily
- Text files:
  - Text files are the normal **.txt** files.
  - You can easily create text files using any simple text editors such as Notepad.
  - You can easily edit or delete the contents.
  - They take minimum effort to maintain, are easily readable, and take bigger storage space.

WORKING WITH FILES

- Step 1: Declaring a file pointer
  - When working with files, you need to declare a pointer of type file.
  - This declaration is needed for communication between the file and the program.

```
FILE * ptr_name;
```

  - File pointer is a pointer which is used to handle and keep track on the files being accessed.
  - A new data type called "FILE" is used to declare file pointer. This data type is defined in stdio.h file.
  - The ptr_name will be used throughout the program to access the file.
  - The file pointer points to the address where the file exists in the memory.

WORKING WITH FILES

- Step 2: Opening a file
  - Step 1 ONLY creates the file pointer.
  - In order to make it point to a file, we must OPEN a file
  - Now, there are two possibilities:
  1. The file already exists in the disk
  2. The file does not exist
  - The syntax for opening a file is:

```
ptr = fopen("fileopen","mode");
```

  - For example,

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

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. Data is added to the end of the file.	If the file does not exist, it will be created.
ab	Open for append in binary mode. Data is added to the end of the file.	If the file does not exist, it will be created.
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 exist, it will be created.
ab+	Open for both reading and appending in binary mode.	If the file does not exist, it will be created.

POINTS TO BE NOTED: FILE MODES

- The “r+”, “w+” and “a+” modes exist to avoid closing and re-opening the file for different operations.
- “r+” mode will not create a new file if file is not present. “w+” mode creates a new file if not present.
- If you write in “w” mode, the entire old content gets overwritten. In “r+” mode,the older content remains as it is.
- In “r+” mode, pointer points to the beginning of the file.
- So, If you write in “r+” mode, it will overwrite from the beginning but remaining content remains as it is.
- If you write to the file in the “r” mode, no changes will be made to the file.
- If you try to read from a file in the “w” or the “a” mode, no content will be read.
- The “a” mode and “w” mode are exactly the same if no file exists.

WORKING WITH FILES

- Step 3: Reading and writing to a file
  - 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().
  - printf() and scanf() read and write on the Console/Terminal.
  - The only difference is that fprintf() and fscanf() expect a pointer to the structure FILE.
  - Syntax of fprintf()

    fprintf(file_ptr,"String"); OR fprintf(file_ptr,"format specifiers", variables);
  - Syntax of fscanf()

    fscanf(file_ptr, "format specifiers", variables);

WORKING WITH FILES

- Step 4: Closing a file
  - A file must always be closed in the end because:
    - An open file that is no longer required consumes space and wastes memory resources.
    - Any changes written/appended to the file might be made to a buffer copy.
    - To save changes to the file permanently contents need to be flushed from buffer to the disk.
    - Flushing happens when a file is closed.
    - When we want to open a file in different modes, (Example first write, then read) it must first be closed.
    - Syntax:

fclose(fp);

AN EXAMPLE….

```
#include <stdio.h>
#include<stdlib.h>
int main() {
  FILE *fp;
  char ch1[20];
  fp = fopen("file1.txt","w");
  fprintf(fp,"Hello");
  fclose(fp);
  fp = fopen("file1.txt","r");
  if(fp == NULL)
  {
  printf("\nCan't open file or file doesn't exist.");
  exit(0);
  }
  fscanf(fp,"%s",ch1);
  printf("The file contents are: %s",ch1);
  fclose(fp);
}
```

Output:

The file contents are: Hello

QUICK EXERCISE

Assume the contents of the file is:

Hello to all.How are you?

What will be the contents of the file?

```
#include <stdio.h>
int main()
{
    FILE *fp; char c;
    fp = fopen("file1.txt", "r+");
    fprintf(fp,"Good Morning.");
    fclose(fp);
    return 0;
}
```

Output:
Good Morning.How are you?

PROGRAM IT

Create a structure Employee with employee_name, ID and salary. Create an Employee variable, input the values from the user on the console and write it to a file.

WORKING WITH FILES: SOME MORE FUNCTIONS

- EOF
  - In C, some read functions return EOF when end of file is reached.
  - It is a pre-defined constant with type int and a negative value.
  - **EOF** is defined in stdio.h (and is usually -1).
- fgetc()
  - fgetc() is used to obtain input from a file single character at a time.
  - It returns the character present at position indicated by file pointer.
  - After reading the character, the file pointer is advanced to next character.
  - If pointer is at end of file or if an error occurs EOF file is returned by this function.
- fputc()
  - fputc() is used to write a single character at a time to a given file.
  - It writes the given character at the position denoted by the file pointer and then advances the file pointer.

WORKING WITH FILES: SOME MORE FUNCTIONS

- fseek()
  - fseek() is used to move file pointer associated with a given file to a specific position.
  - Position defines the point with respect to which the file pointer needs to be moved. It has three values:
  - SEEK_END : It denotes end of the file.
  - SEEK_SET : It denotes starting of the file.
  - SEEK_CUR : It denotes file pointer's current position.
  - Syntax:

```
fseek(file_ptr,offset to be added to position, position);
```

WORKING WITH FILES: SOME MORE FUNCTIONS

- ftell()
  - ftell() in C is used to find out the position of file pointer in the file with respect to starting of the file.
  - Syntax:

```
long int var_n = ftell(file_ptr);
```

- rewind()
  - The rewind() function is used to move the cursor at the begining of the file.
- HOMEWORK:
  - Reading and writing from a binary file.
  - fread() and fwrite()

QUICK EXERCISE

Assume the contents of the file are:

Hello to all.

What will be the output?

```
#include <stdio.h>
int main()
{
    FILE *fp;
    fp = fopen("file1.txt", "r");
    fseek(fp, 0, SEEK_END);
    printf("%ld", ftell(fp));
    return 0;
}
```

Output:
12

QUICK EXERCISE

```
#include <stdio.h>
int main()
{
    FILE *fp; char c;
    fp = fopen("file1.txt", "w");
    fprintf(fp,"Hello to all");
    rewind(fp);
    fseek(fp,6,SEEK_CUR);
    fprintf(fp,"Good Morning.");
    fclose(fp);
    return 0;
}
```

Output:
Hello Good Morning.

PROGRAM IT

Read the contents of a file and display it on the console using fgetc().

Program:

```
FILE *fp; char c;
fp = fopen("file1.txt", "r");
while((c=fgetc(fp))!=EOF)
{
printf("%c",c);
}
```