Introduction to Computing and Programming

File Handling in C

Content

Recap

- User-defined Vs Derived data Types
- Structure
- Union
- Bit Manipulation

What is file?

- A named collection of data, typically stored in a secondary storage (e.g., hard disk).

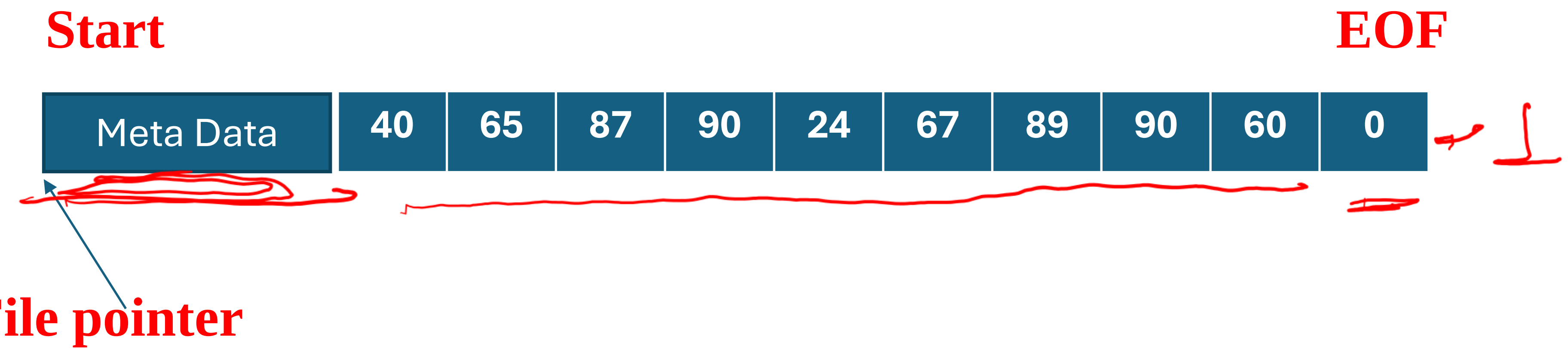
  Examples
  - Records of all employees in an organization
  - Document files created using Microsoft Word
  - Video of a movie
  - Audio of a music

- Non-volatile data storage
  - Can be used when power to computer is off

How is a file stored?

- Stored as sequence of bytes, logically contiguous (may not be physically contiguous on disk).
  - Discrete storage unit for data in the form of a stream of bytes.
  - Every file is characterized with a starting of file, sequence of bytes (actual data), and end of stream (or end of file).
  - Allow only sequential access of data by a pointer performing.
  - Meta-data (information about the file) before the stream of actual data can be maintained to have a knowledge about the data stored in it.

How is a file stored?



- **The last byte of a file contains the end end-of-file character (EOF, with ASCII code 1A (Hex).**
- **While reading a file, the EOF character can be checked to know the end.**

Type of files

- Text files
  - Contain ASCII code only
    - C-programs
- Binary files
  - Contain non-ASCII characters
    - Image, audio, video, executable, etc.

b

Operations on Files

- Open : To open a file to store/retrieve data in it
- Read : The file is used as an input
- Write : The file is used as output
- Close : Preserve the file for a later use
- Access: Random accessing data in a file

File Handling Commands

- Include header file `<stdio.h>` to access all file handling utilities.
- A data type namely FILE is used to create a pointer to a file.

  Syntax
  ```
  FILE * fptr;        // fptr is a pointer to file
  ```
- To open a file, use `fopen()` function

  Syntax
  ```
  FILE * fopen(char *filename, char *mode)
  ```
- To close a file, use `fclose()` function

  Syntax
  ```
  int fclose(FILE *fptr);
  ```

fopen() function

FILE * fopen(char *filename, char *mode)

- The first argument is a string to characters indicating the name of the file to be opened and the convention of file name should follow the convention of giving file name in the operating system.

  Examples:

  xyz12.c　　student.data　　myFile

- The second argument is to specify the mode of file opening. There are five file opening modes in C
  - "r"　: Opens a file for reading
  - "w"　: Creates a file for writing (overwrite, if it contains data)
  - "w+"　: Creates a file for reading and writing (overwrite, if it contains data)
  - "a"　: Opens a file for appending - writing on the end of the file
  - “rb”　: Read a binary file (read as bytes)
  - “wb”　: Write into a binary file (overwrite, if it contains data)
- It returns the special value NULL to indicate that it couldn't open the file.

fopen() function Cont...

- If a file that does not exist is opened for writing or appending, it is created as a new.
- Opening an existing file for writing causes the old contents to be discarded.
- Opening an existing file for appending preserves the old contents, and new contents will be added at the end.
- File opening error
  - Trying to read a file that does not exist.
  - Trying to read a file that doesn’t have permission.
  - If there is an error, fopen() returns NULL

fopen() example

```
#include <stdio.h>
void main() {
  FILE *fptr;          // Declare a pointer to a file
  char filename[]= "file2.dat";
  fptr = fopen(filename,"w");
// Also, alternatively fptr = fopen (“file2.dat”,"w");
  if (fptr == NULL) {
    printf (“Error in creating file”);
    exit(-1);       // Quit the function
  }
  else /* code for doing something */
{     fprintf(fptr, "This is a test file.\n");  // Write to the file
      printf("File created successfully!\n");
fclose(fptr);  // Close the file   }
}
```

<stdlib.h>

Reading from a file

- Following functions in C (defined in stdio.h) are usually used for reading simple data from a file
  - fgetc(…)
  - fscanf(…)
  - fgets(…)
  - getc(…)

Reading from a File: fgetc()

Syntax for fgetc(…)

```
int fgetc(FILE *fptr)
```

- The fgetc() function returns the next character in the stream fptr as an unsigned char (converted to int).
- It returns EOF if end of file or error occurs.

```
FILE *fptr;
int c;
/* Open file and check it is open */
while ((c = fgetc(fptr)) != NULL)
   {
     printf ("%c",c);
   }
```

Reading from a File: fscanf()

Syntax for fscanf(…)

```
int fscanf(FILE *fptr, char *format, ...);
```

- fscanf reads from the stream fptr under control of format and assigns converted values through subsequent assignments, each of which must be a pointer.
  - It returns when format is exhausted.
- fscanf returns EOF if end of file or an error occurs before any conversion.
- it returns the number of input items converted and assigned.

Reading from a File: `fgets(...)`

Syntax for fgets(…)

char *fgets(char *s, int n, FILE *fptr)

s	The array where the characters that are read will be stored.
n	The size of *s*.
fptr	The stream to read.

- fgets() reads at most n-1 characters into the array s, stopping if a newline is encountered.
    - The newline is included in the array, which is terminated by ‘\0’.
- The fgets() function returns s or NULL if EOF or error occurs.

fgets() example

```
FILE *fptr;
char line [1000];
/* Open file and check it is open */

while (fgets(line,1000,fptr) != NULL)
   {
   printf ("Read line %s\n",line);
   }
```

Reading from a File: getc(...)

Syntax for getc(…) ✓

int getc(FILE *fptr)

- getc(…) is equivalent to fgetc(…) except that it is a macro.

Read a text file and then print the content on the screen

```
#include <stdio.h>
#include <stdlib.h>
int main()
{ int ch, fileName[25];
  FILE *fp;
  printf("Enter the name of file you wish to read\n");
  gets(fileName);
  fp = fopen(fileName,"r"); // read mode
  if( fp == NULL )
  { printf("Error while opening the file.\n");
    exit(-1);}
  printf("The contents of %s file are :\n", fileName);
  while( ( ch = getc(fp) ) != EOF )
  printf("%c",ch);
  fclose(fp);
  return 0;
}
```

Output:

```
Enter the name of file you wish to read
test.txt
The contents of test.txt file are :
C programming is fun.
```

Writing into a file

- Following functions in C (defined in `stdio.h`) are usually used for writing simple data into a file
  - `fputc(…)`
  - `fprintf(…)`
  - `fputs(…)`
  - `putc(…)`

Writing into a file: fputc(…)

Syntax for fputc(…)

```
int fputc(int c, FILE *fptr)
```

- The fputc() function writes the character c to file fptr and returns the character written, or EOF if an error occurs.

```
#include  <stdio.h>

   filecopy(File *fpIn, FILE *fpOut)
   {
      int c;
      while ((c = fgetc(fpIn) != EOF)
         fputc(c, fpOut);
   }
```

Writing into a file: fprintf(…)

Syntax for fprintf(…)

```
int fprintf(FILE *fptr, char *format,...)
```

- fprintf() converts and writes output to the steam fptr under the control of format.
- The function is similar to printf() function except the first argument which is a file pointer that specifies the file to be written.
- The fprintf() returns the number of characters written, or negative if an error occur.

Writing into a File: fprintf(…)

```
#include  <stdio.h>
   void main()
   {
      FILE *fptr;
      fptr = fopen(“test.txt”, “w”);
      fprintf(fptr, “Programming in C is really a fun!\n”);
      fprintf(fptr, “Let’s enjoy it\n”);
      fclose(fptr);
      return;
   }
```

Writing into a File: fputs()

Syntax for fputs:

```
int fputs(char *s, FILE *fptr)
```

- The fputs() function writes a string (which need not contain a newline) to a file.
- It returns non-negative, or EOF if an error occurs.

```
#include  <stdio.h>
  void main()
  {
    FILE *fptr;
    fptr = fopen("test.txt", "w");
    fputs("Programming in C is really a fun!",
fptr);
    fputs("\n", fptr);
    fputs("Let's enjoy it \n", fptr);
    fclose(fptr);
    return;
  }
```

Writing into a File: putc(…)

Syntax for putc(…)

```
int putc(FILE *fptr)
```

- The putc() function is same as the putc(…).

```
#include  <stdio.h>

 filecopy(File *fpIn, FILE *fpOut)
 {
    int c;
    while ((c = getc(fpIn) != EOF)
       putc(c, fpOut);
 }
```

Program to write some text reading from the keyboard and writing them into a file and then print the content from the file on the screen.

```
#include  <stdio.h>
  main()
  {
   FILE *f1;
  char c;
  printf("Data Input\n\n");/* Open the file INPUT */
  f1 = fopen("INPUT", "w");
  while((c=getchar()) != EOF) /* Get a character from
keyboard*/
   putc(c,f1); /* Write a character to INPUT*/
   fclose(f1); /* Close the file INPUT*/
   printf("\nData Output\n\n");
   f1 = fopen("INPUT","r"); /* Reopen the file INPUT */
   while((c=getc(f1)) != EOF) /* Read a character from
INPUT*/
   printf("%c",c); /* Display a character on screen */
   fclose(f1); /* Close the file INPUT */
 }
```

Output:

```
Data Input
  This is a program to test the file handling
features on this system
Data Output
  This is a program to test the file handling
features on this system
```

Special streams in C

- When a C program is started, the operating system environment is responsible for opening three files and providing file pointer for them. These files are

• stdin	Standard input. Normally it is connected to keyboard
• stdout	Standard output, In general, it is connected to display screen
• stderr	It is also an output stream and usually assigned to a program in the same way that stdin and stderr are. Output written on stderr normally appears on the screen

Note:

getc(stdin) is same as fgetc (stdin)

Example: Special Streams

```
#include <stdio.h>
main()
{
 int i;
 fprintf(stdout,"Give value of i \n");
 fscanf(stdin,"%d",&i);
 fprintf(stdout,"Value of i=%d \n",i);
 }
```

Output:

```
Give value of i
15
Value of i=15
```

Error Handling : stderr and exit

- What happens if the errors are not shown in the screen instead if it's going into a file or into another program via a pipeline.
- To handle this situation better, a second output stream, called `stderr`, is assigned to a program in the same way that `stdin` and `stdout` are.
- Output written on `stderr` normally appears on the screen even if the standard output is redirected.

Error Handling: Example

```
#include <stdio.h>
main(int argc, char *argv[])
  { FILE *fp;
     void filecopy(FILE *, FILE *);
     char *prog = argv[0];  /* program name for errors */
     if (argc == 1 ) /* no args; copy standard input */
        filecopy(stdin, stdout);
     else
        while (--argc > 0)
```

```
if ((fp = fopen(*++argv, "r")) == NULL) {
   fprintf(stderr, "%s: can't open %s\n", prog, *argv);
      exit(1);
  } else {
        filecopy(fp, stdout);
        fclose(fp);}
 if (ferror(stdout)) {
     fprintf(stderr, "%s: error writing stdout\n", prog);
      exit(2);
  }
      exit(0);
}
```

Structured Input/Output for Files

- Other than the simple data, C language provides the following two functions for storing and retrieving composite data.

- `fwrite()` To write a group of structured data

- `fread()` To read a group of structured data

Writing records: fwrite()

fwrite() writes data from the array pointed to, by ptr to the given stream fptr.
Syntax:

```
int fwrite(void *ptr, int size, int nobj, FILE *fptr);
```

- ptr This is the pointer to a block of memory with a minimum size of size *nobj bytes.
- size This is the size in bytes of each element to be written.
- nobj This is the number of elements, each one with a size of size bytes.
- fptr This is the pointer to a FILE object that specifies an output stream.

Example: fwrite()

```
#include<stdio.h>
      struct Student
      {
            int roll;
            char name[25];
            float marks;
      };
      void main()
      { FILE *fp;
            int ch;
            struct Student Stu;
            fp = fopen("Student.dat","w");   //Statement   1
            if(fp == NULL)
            {printf("\nCan't open file or file doesn't exist.");
              exit(0); }
   do{printf("\nEnter Roll : ");
    scanf("%d",&Stu.roll);
    printf("Enter Name : ");
    scanf("%s",Stu.name);
    printf("Enter Marks : ");
    scanf("%f",&Stu.marks);
 fwrite(&Stu,sizeof(Stu),1,fp);
      printf("\nDo you want to add another data (y/n) : ");
      ch = getchar();}
while(ch=='y' || ch=='Y');
printf("\nData written successfully...");
fclose(fp) }
```

```
Enter Roll : 1
Enter Name : AA
Enter Marks : 78.53
Do you want to add another data (y/n) : y
Enter Roll : 2
Enter Name : BB
Enter Marks : 72.65
Do you want to add another data (y/n) : y
Enter Roll : 3
Enter Name : CC
Enter Marks : 82.65
Do you want to add another data (y/n) : n
Data written successfully...
```

Reading Records: `fread()`

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

Syntax:

```
int fread(void *ptr, int size, int nobj, FILE *fptr);
```

- ptr This is the pointer to a block of memory with a minimum size of size *nobj bytes.
- size This is the size in bytes of each element to be read.
- nobj This is the number of elements, each one with a size of size bytes.
- fptr This is the pointer to a FILE object that specifies an input stream.

Example: `fread()`

```
#include<stdio.h>
struct Student
    {     int roll;
          char name[25];
          float marks;};
    void main()
    {    FILE *fp;
          int ch;
          struct Student Stu;
          fp = fopen("Student.dat","r");  //Statement   1
          if(fp == NULL)
          { printf("\nCan't open file or file doesn't exist.");
            exit(0);}
     printf("\n\tRoll\tName\tMarks\n");
     while(fread(&Stu,sizeof(Stu),1,fp)>0)
     printf("\n\t%d\t%s\t%f",Stu.roll,Stu.name,Stu.marks);
    fclose(fp);}
```

OUTPUT

```
Roll     Name      Marks
1        AA        78.53
2        BB        72.65
3        CC        82.65
```

File handling: Example

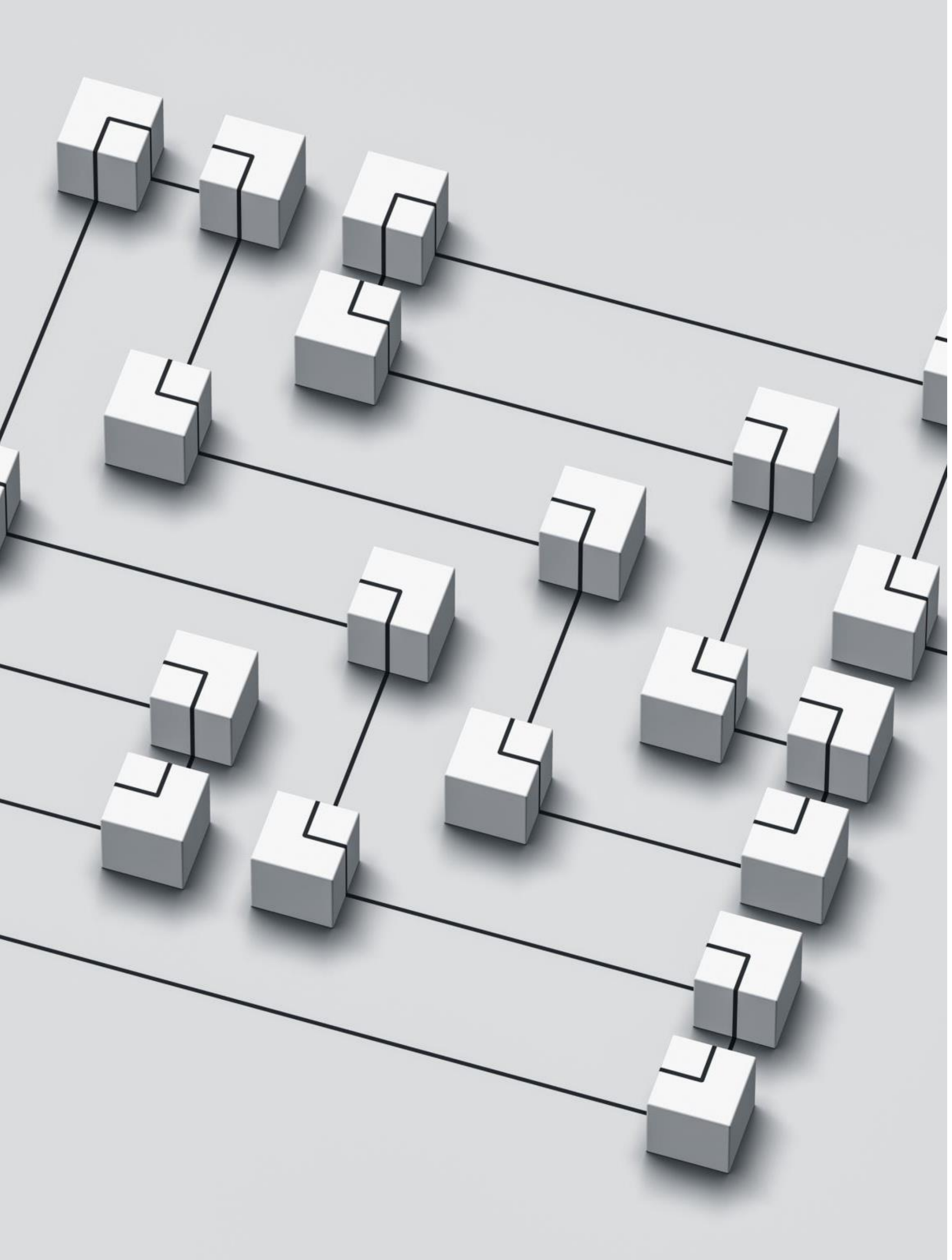
```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   char ch, sourceFile[20], targetFile[20];
   FILE *source, *target;
   printf("Enter name of file to copy\n");
   gets(sourceFile);
   source = fopen(sourceFile, "r");
   if( source == NULL )
   {
      printf("Input file error. Program
abort...\n");
      exit(-1);
   }
printf("Enter name of target file\n");
   gets(target_file);

   target = fopen(targetFile, "w");

   if( target == NULL )
   {
      fclose(source);
      printf("Output File Error! File copy fails...\n");
      exit(-1);
   }

   while( (ch = fgetc(source) ) != EOF )
      fputc(ch, target);

   printf("File copied successfully.\n");

   fclose(source);
   fclose(target);

   return 0;
}
```

Announcement

- Maximum marks in Mid-sem: 40
- I will be in my office on Wednesday 2 to 4 and Friday 1 to 3 for any marks related issue or project clarifications
- I will upload the question bank on File streaming

Report Format for Project: Doc file

- What your project is doing in 1 or 2 paragraph?
- What logic or features have you used from ICP concepts; Explain in 1 paragraph.
- How efficient you have made your code?
- Dynamic memory allocation, file handling, etc.
- Write some special feature or logic that you have used.
- Comment your code properly



Upcoming Lecture

- Revision: Pointers, Pointers Arithmetic, basics and some practice questions
- End-sem Exam pattern