# File Handling in C

# What is a File?

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

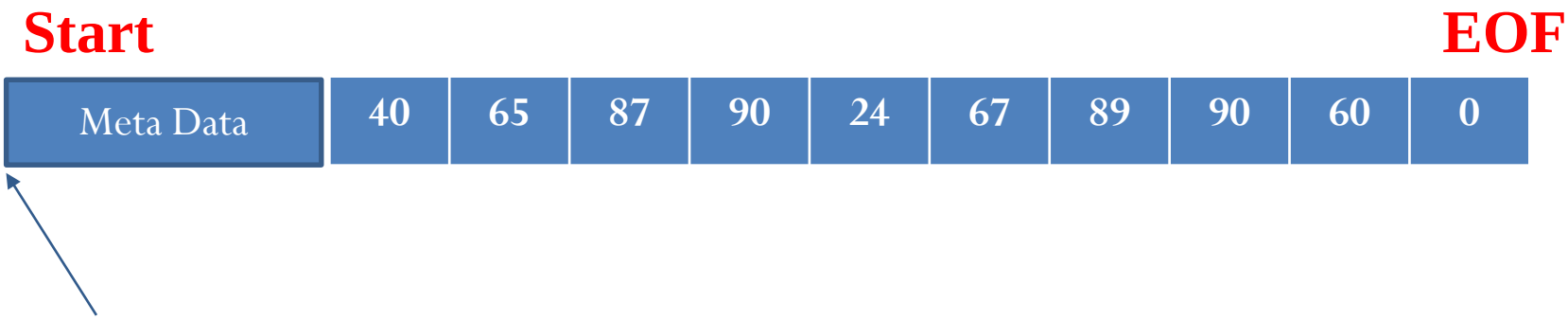
  Examples
  - Records of all employees in an organization
  - Document files created using Microsoft Word
  - Video of a movie
  - Audio of a music

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

# How a File is 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 (or beginning of file -BOF), sequence of bytes (actual data), and end of stream (or end of file-EOF).
  - Allows only sequential access of data by a pointer performing.

# How a File is Stored?




**Note:**

- 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.
- 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.

What type of file a .docx file produced by MS-Word?

# Operations on Files

- Typical operations on a file are
  - 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

# Opening and Closing a File

# File Handling Commands

- Include header file `<stdio.h>` to access all file handling utilities.
- A data type namely FILE is there 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 of characters indicating the name of the file to be opened.

  Examples: `xyz12.c; student.data`
- 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)
  - "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

- 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.

# Example: `fopen()`

```
#include <stdio.h>
void main()
{
    FILE *fptr;                  // Declare a pointer to a file

    char filename[]= "file2.dat";

    fptr = fopen(filename,"w");

//     fptr = fopen ("file2.dat","w"); // alternatively

    if (fptr == NULL) {
       printf ("Error in creating file");
       exit(-1);                 // Quit the function
    }
    else /* code for doing something */

fclose(fptr);//
}
```

# Reading from a File

# 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(…)`
  - `ungetc(…)`

# 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.

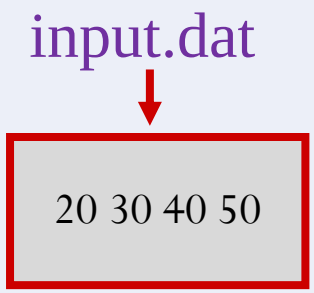
# Example: Using `fscanf(...)`

```
FILE *fptr;

fptr= fopen ("input.dat","r");
int n;
/* Check it's open */
if (fptr == NULL)
   {
    printf("Error in opening file \n");
   }


  n = fscanf(fptr,"%d %d",&x,&y);

  ...
```




x = 20
y = 30

# 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.

# Example: Using `fgets(...)`

```
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 a File: `getc(...)`

## Syntax for `getc(...)`

```
int getc(FILE *fptr)
```

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

# Example: Using `getc(...)`

C program to 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.
```

# Undo a File Reading: `ungetc()`

`ungetc()`: Push a character back onto an input stream.

**Syntax:**

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

Arguments:

| | |
|---|---|
| `c` | The character that you want to push back. |
| `fptr` | The stream you want to push the character back on. |

- Only one character of pushback is guaranteed per file.
- `ungetc` may be used with any of the input functions like `scanf`, `getc`, or `getchar`.

# Example: `ungetc()`

```
#include <stdio.h>
int main(void)
{
    int ch;

    while ((ch = getchar()) != '1') // reads characters from the stdin
        putchar(ch);                // and show them on stdout until en

    ungetc(ch, stdin); // ungetc() returns '1' previously read back to stdin

    ch = getchar(); // getchar() attempts to read next character from stdin
                    // and reads character '1' returned back to the stdin by unge

    putchar(ch);    // putchar() displays character
    puts("");

    printf("Thank you!\n");
    return 0;
}
```

**OUTPUT**

```
a
a
v
v
c
c
u
u
1
1
Thank you!
```

# Writing into a File

# 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.

# Example: `fputs(...)`

```
#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);
   }
```

# Writing into a File: Example

- A sample C 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");
```

*Contd…*

# Writing into a File

```
    while((c=getchar()) != EOF) /* Get a character from keyboard*/
    putc(c,f1); /* Write a character to IN

    fclose(f1); /* Close the file INPUT*/
    printf("\nData Output\n\n")

    f1 = fopen("INPUT","r"); /* R

    while((c=getc(f1)) != EOF)
    printf("%c",c); /* Display a chara

    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

# Special Streams

- 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)`

# Special Streams

```
fprintf (stdout,"Hello World!\n");

printf(""Hello World!\n");
```

The above two statements are same!

# 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.

# Example: Error Handling

```
#include <stdio.h>

   /* cat:  concatenate files */
   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)
```

*Contd…*

# Example: Error Handling

```
  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);
}
```

# Direct Input and Output

# 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);
		}
```

*Contd...*

# Example: fwrite()

```
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);
      }
```

*Contd…*

# Example: `fwrite()`

**OUTPUT**

```
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);
                }
```

*Contd…*

# Example: `fread()`

```
 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 |

# Random Accessing Files

# File Positioning Functions in C

- When doing reads and writes to a file, the OS keeps track of where you are in the file using a counter generically known as the file pointer.

- So long we have learnt about the sequential access in a file.
- The following are the functions to access file at random
- `ftell()` Tell the current position of the file pointer
- `fseek()` To position a file pointer at a desired place within the file
- `rewind()` Is equivalent to `fseek()`

# Random Accessing a File: `ftell()`

```
long ftell(FILE *fptr);
```

- `ftell()` takes a file pointer `fptr` and returns in a number of type `long`, that corresponds to the current position.
- It returns `-1L` on error.

Example

```
long n;
n = ftell(fptr);
```

Note:

In this case, `n` gives the relative offset (in bytes) of the current position. This means that `n` bytes have already been read (or written).

# Random Accessing a File: `fseek()`

```
int fseek(FILE *fptr, long offset, int whence);
```

- `fseek()` function is used to move the file position to a desired location within the file.
- The first argument is the file in question. `offset` argument is the position that you want to seek to, and `whence` is what that offset is relative to.
- You can set the value of `whence` to one of the three things:

| | |
|---|---|
| SEEK_SET | offset is relative to the beginning of the file. |
| SEEK_CUR | offset is relative to the current file pointer position. |
| SEEK_END | offset is relative to the end of the file. |

# Example: `fseek()`

- You can set the value of `whence` to one of the three things:

```
fseek(fp, 0L, SEEK_SET);    // go to the beginning
fseek(fp, 0L, SEEK_CUR);    // Stay at the current position
fseek(fp, 0L, SEEK_END);    // go to the end of the file, i.e., past
                               the last character of the file
fseek(fp, 0L, SEEK_SET);    // go to the beginning

fseek(fp, m, SEEK_SET);    // Move to (m+1)th  byte in the file
fseek(fp, m, SEEK_CUR);    // Go forward by m bytes
fseek(fp, -m, SEEK_CUR);    // Go backward by m bytes from the
                               current position
fseek(fp, -m, SEEK_END);    // Go back by m bytes from the end
```

# Random Accessing a File: `rewind()`

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

- `rewind()`: It repositions the file pointer at the beginning of the file

Example

```
rewind (fptr);                  // Set the file pointer at the beginning
fseek(fptr, 0L, SEEK_SET);      // same as the rewind()
```

# `fseek()` vs. `rewind()`

**Return value**

- For `fseek()`, on success zero is returned; `-1L` is returned on failure.
- The call to `rewind()` never fails.

**Examples:**

```
fseek(fp, 100, SEEK_SET); // seek to the 100th byte of the file
fseek(fp, -30, SEEK_CUR); // seek backward 30 bytes from the current position
fseek(fp, -10, SEEK_END); // seek to the 10th byte before the end of file

fseek(fp, 0, SEEK_SET);   // seek to the beginning of the file
rewind(fp);               // seek to the beginning of the file
```

# Examples

# File Handling : Example 1

A program to copy a text file to another file.

```
#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);
   }
```

*Contd…*

# File Handling : Example 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;
}
```

# File Handling : Example 2

A program to copy a text file to another file. Read the file names through command line.

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

int main(int argc, char *argv[])
{
   char ch;
   FILE *source, *target;

   source = fopen(argv[1], "r");

   if( source == NULL )
   {
      printf("Input file error. Program abort...\n");
      exit(-1);
   }
```

*Contd…*

# File Handling : Example 2

```
    target = fopen(argv[2], "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;
}
```

# File Handling : Example 3

A program to concatenate a file (say A) to another file (say B) so that the resultant file is A = A+B. Read the file names for A and B through command line.

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

int main(int argc, char *argv[])
{
   int ch;
   FILE *fpA, *fpB;

   fpB = fopen(argv[2], "r");    //Open the file B

   if( fpB == NULL )
   {
      printf("Input file error. Program abort...\n");
      exit(-1);
   }
```

*Contd…*

# File Handling : Example 3

```
    fpA = fopen(argv[1], "a");   //Open the file A in append mode

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

    while( (ch = fgetc(fpA) ) != EOF )
       fputc(ch, fpA);

    printf("Files are concatenated successfully.\n");

    fclose(fpA);
    fclose(fpB);

    return 0;
}
```

# File Handling : Example 4

A program to encrypt a text file. Read the file names through command line.

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

int main(int argc, char *argv[])
{
   char ch;
   FILE *source, *target;

   source = fopen(argv[1], "r");

   if( source == NULL )
   {
      printf("Input file error. Program abort...\n");
      exit(-1);
   }
```

*Contd…*

# File Handling : Example 4

```
    target = fopen(argv[2], "w");

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

    while( (ch = fgetc(source) ) != EOF )
        fputc(ch+10, target);       //Change the character...

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

    fclose(source);
    fclose(target);

    return 0;
}
```

# File Handling : Example 5

A program to display a file on the screen. Read the file name through command line.

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

int main(int argc, char *argv[])
{
   char ch;
   FILE *source, *target;

   source = fopen(argv[1], "r");

   if( source == NULL )
   {
      printf("Input file error. Program abort...\n");
      exit(-1);
   }

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

   fclose(source);

   return 0;
}
```

# File Handling : Example 6

A program to store a record in file. Read the file and store all records in an array.

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

struct Student {
     int rollNo;
     char name[20];
     float marks;
};

int main(int argc, char *argv[])
{
   int choice = 1;
   struct Student *data;
   FILE *outfile, *infile;

   outfile = fopen(argv[1], "w");

   if( outfile == NULL )
   {
      printf("Input file error. Program abort...\n");
      exit(-1);
   }
```

*Contd…*

# File Handling : Example 6

A program to store a record in file. Read the file and store all records in an array.

```
    while (choice) {
        data = (struct *)malloc(sizeof(struct Student));
        if (data != NULL) {
           printf(\nEnter Roll No: "); scanf("%d",&data->rollNo;);
           printf(\nEnter Name: "); scanf("%s",data->name;);

           fwrite (data, sizeof(struct Student), 1, outfile);

           printf("\nDo you want to add more record (Type 0 for NO)?");
           scanf("%d", &choice);
        }
     }

     fclose(outfile);


    return 0;
}
```

*Contd…*

# File Handling : Example 6

```
    infile = fopen(argv[1], "r");

    struct Student data[100];

    if( infile == NULL )
    {
      printf("File error. Program abort...\n");
      exit(-1);
    }
     choice = 0;

     while (fread (&data, sizeof(struct Student), 1, infile))
         data[choice++] = data;
    }

   return 0;
}
```