Text Files

COMP 1002/1402

File Types

Two main types:

Binary Files
- Everything stored as 0’s and 1’s

Text Files
- Usually human readable characters
- Each data line ends with newline char

File Storage & Access

Files are stored in auxiliary storage devices

Read and Write access is buffered

A buffer is a temporary storage area

Streams

File Input and Output is element by element

This is essentially a stream of information

In C all File structures are byte streams

FILE

FILE is the type we use to reference files

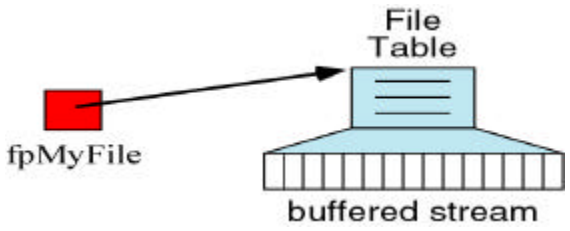
Defined in stdio.h

Usually declare variables as a pointer:

```
FILE *filename;
```

File Table

The details are unnecessary.
FILE is linked to File Tables
File Table :
name of file, location of buffer, current state of the file

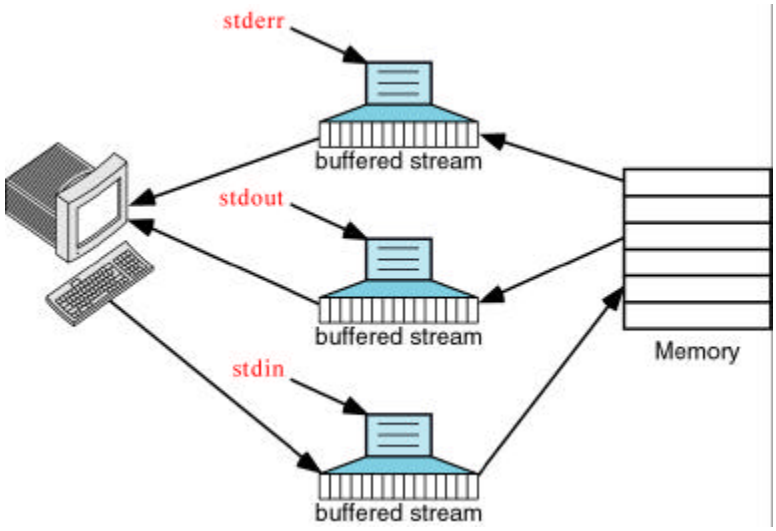


Standard Files

Stdio.h defines 3 standard streams (files)

Standard Input	(stdin)
Standard Output	(stdout)
Standard Error	(stderr)

Standard Files



User Files

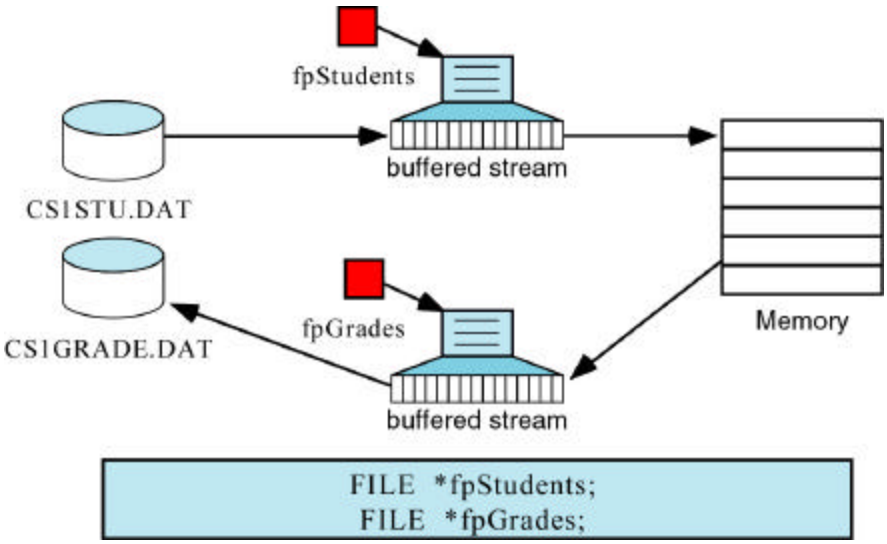
Standard Files are opened automatically

User files must be opened by the user

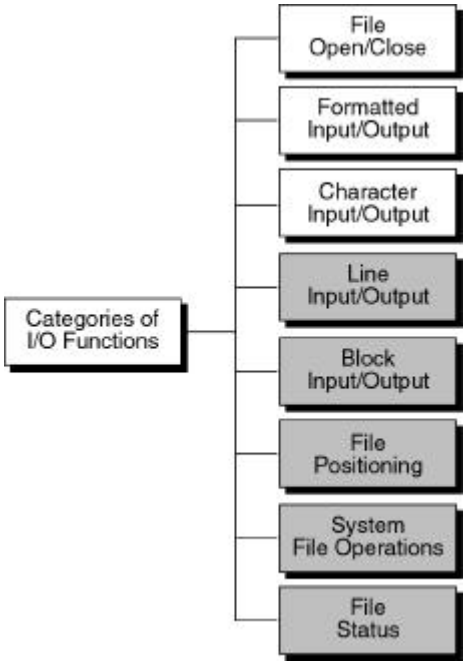
Can be opened for input or output

One file one stream (or vice versa)

User Files



Types of Input & Output



Opening a File

```
fopen("filename","mode");
```

Returns a pointer to **`FILE`**

Automatically creates buffer

Mode	Meaning
r	Open file for reading •If file exists, the marker is positioned at beginning •If file doesn't exist, it is created
w	Open text file for writing •If file exists, it is emptied. •If file doesn't exist, it is created.
a	Open text file for append •If file exists, the marker is positioned at end. •If file doesn't exist, it is created.

Open Examples

```
FILE *fpTemp;
fpTemp = fopen("TEMPS.DAT","w");
fpTemp = fopen("A:\\TEMPS.DAT","w");
```

Returns **NULL** if there is a problem

```
if((fpTemp=fopen("T.DAT","w"))==NULL){
```

…

Closing a file

```
fclose( fpTemp);
```

Returns **EOF** (end of file constant)
if an error occurs

```
if(fclose(fpTemp) == EOF) { …
```

Formatting Input/Output

Format Strings (printf, scanf,…) are important

Format Strings specify:

White space,
Text characters, and
Field Specifiers (**%…**)

Whitespace

Input

One or more whitespace chars:

Discard one or more whitespace chars

Output

Whitespace copied to output

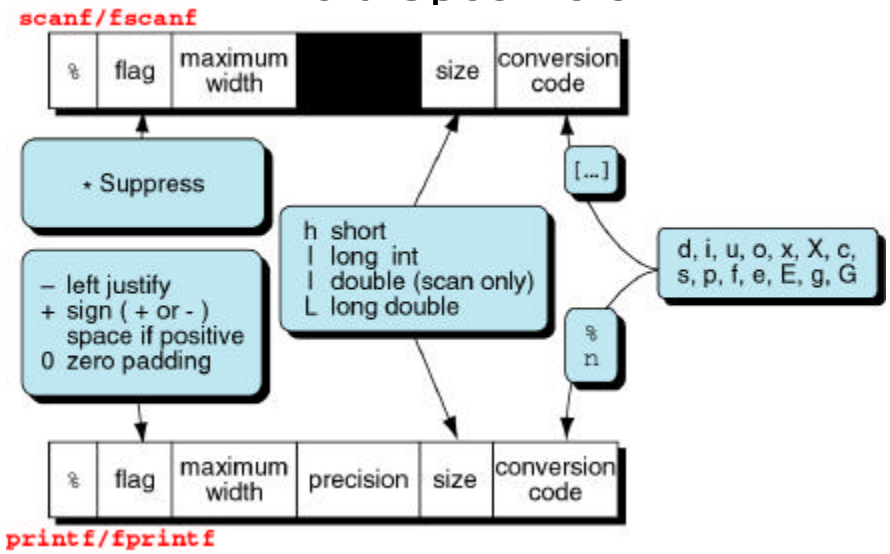
Text Characters

Input and Output:

Exact Matching (Input)

or Copy to Output

Field Specifiers



scanf				printf		
Flag	Size	Codes	Usage	Flag	Size	Codes
+-	h	d	short int	+-	h	d
			int	0		
	l		long int	sp	l	
+-	h	u	unsigned short int	+-	h	u
			unsigned int	0		
	l		unsigned long int	Sp	l	
+-	h	o	short int – octal	+-	h	o
			int – octal	0		
	l		long int – octal	sp	l	

scanf			Usage	printf		
Flag	Size	Codes		Flag	Size	Codes
+-	h	x X	short int – hex	+-	h	x X
			int – hex	0		
	l		long int – hex	sp	l	
		f	float			f
	l		double		l	
	L		long double		L	
		e, E	float - scientific			e, E
	l	g, G	double – scientific		l	g, G
	L		long double – scientific		L	

scanf			Usage	printf		
Flag	Size	Codes		Flag	Size	Codes
		c	char			c
		s	string			s
		p	pointer (address)			p
	h	n	short – I/O count			n
			int – I/O count			
	l		long – I/O count		l	
		[…]				

`scanf` and **`fscanf`**

```
scanf("format string", addresses);

FILE *fp;
fscanf(fp,"format string",addresses);
```

Each function:

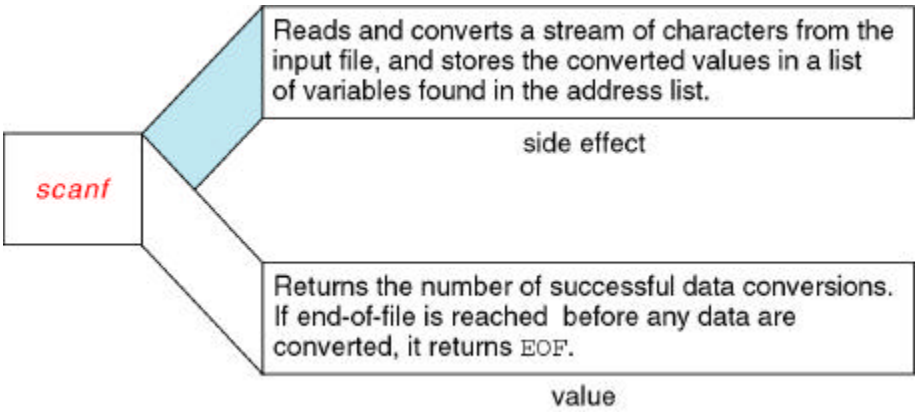
- Side effects
- Return value

Input Data Formatting

Conversion processes characters until:

1. End-of-file is reached
2. An inappropriate character is found
3. The number of characters read is equal to maximum field width.

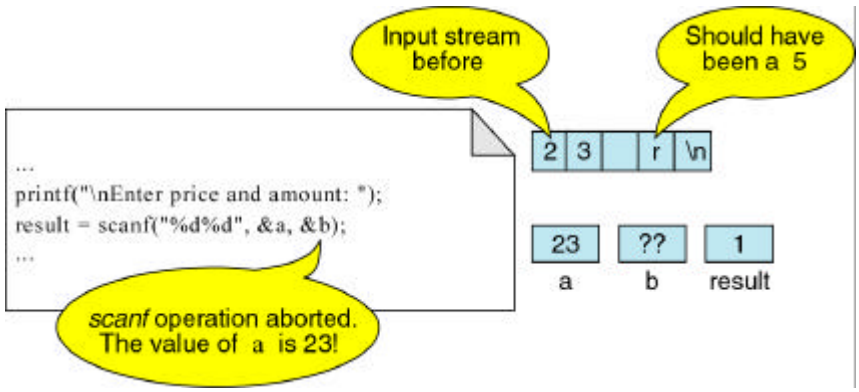
scanf & fscanf



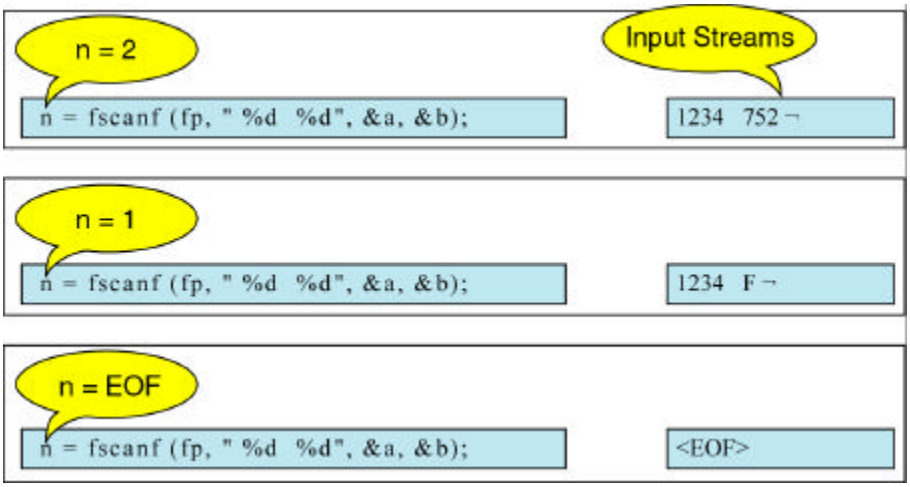
Return Value

EOF if End-of-File is reached

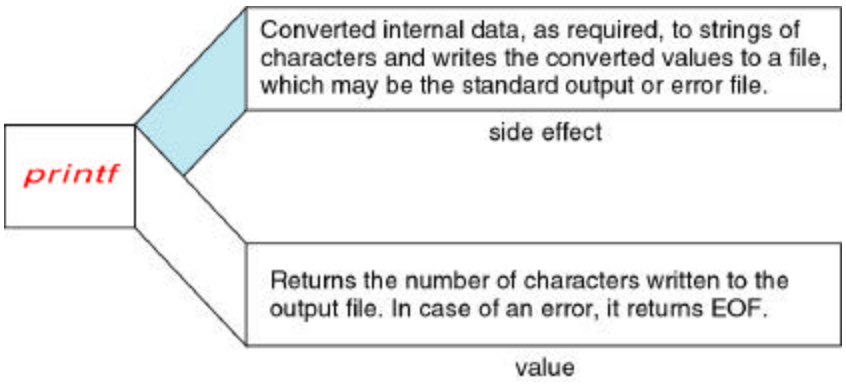
or Number of successful conversions



fscanf examples




```
printf("format string", variables);
FILE *fptemp;
fprintf(fptemp,"format",variables);
```



Return Value

Be certain to check for **EOF**

```
if (fprintf(fp, "%s", str)==EOF)
  {…}
```

The error check is important

Character Input/Output

```
int getchar(void);
/* think scanf with %c */

a = getchar();
/* buffered input same as %c */

int putchar(int out_char);
/* think printf %c */
```

Always returns the outputted character

Character Input/Output

```
int getc(FILE *fpIn);
int fgetc(FILE *fpIn);
```

Each get a character from the FILE *fpIn

Returning it in integer form (EOF) on errors

Character Input/Output

```
int putc(int oneChar, FILE *fpOut);
int fputc(int oneChar, FILE *fpOut);
```

Output a single character

Return the character if successful else EOF

Character Input/Output

```
int ungetc(int oneChar, FILE *stream);
```

Returns **`oneChar`** if successful

Attempts to put **`oneChar`** back into Stream