Warning: These notes are not complete, it is a Skelton that will be modified/add-to in the class. If you want to us them for studying, either attend the class or get the completed notes from someone who did

EECS2031

Lecture 2
Data types

Modifiers

- signed (unsigned) int long int
- long long int
- int may be omitted
- sizeof()

Data Types

- **int** i=3;   // integer
- **long** l=3;   // long integer
- integer   **unsigned long** ul= 3UL;   //unsigned long
- **int** i=0xA;   //hexadecimal
- **int** i=012;   //octal number
- **float** pi=3.14159   //float
- floating point   **float** pi=3.141F   //float
- **double** pi=3.1415926535897932384L

Characters

- One byte
- Included between 2 single quotes
- char x =‘A’
- Character string “This is a string”
- ‘A’ != “A”

| A |

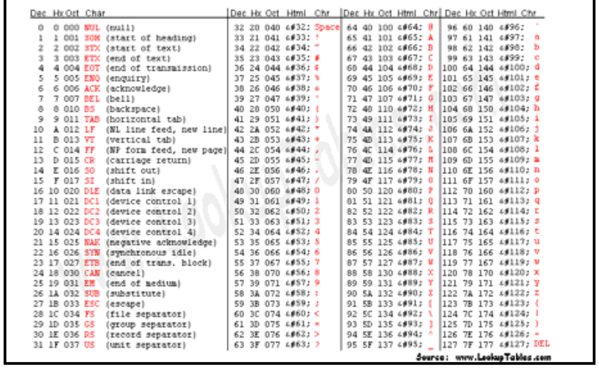
| A | \0 |

- X=‘\012’   newline or 10 decimal

Arrays

- int a[14];
- char s[10];

Characters



Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH (start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX (start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX (end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT (end of transmission)	36	24	044	&#36;	$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ (enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK (acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL (bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS (backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB (horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF (NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT (vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF (NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR (carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO (shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI (shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE (data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1 (device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3 (device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4 (device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK (negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN (synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB (end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN (cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	EM (end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB (substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC (escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS (file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	|
29	1D	035	GS (group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	RS (record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US (unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Boolean Expressions

- Relational operators
- ==, !=, <, <=, >, >=
- Logical operators
- &&, ||, !

I/O

- Every program has a standard input and output (stdin, stdout and stderr)
- Usually, keyboard and monitor
- Can use > and < for redirection
- printf(“This is a test %d \n”,x)
- scanf(“%x%d”,&x,&y)

%d	%s	%c	%f	%lf
integer	string	character	float	double precision

I/O

- `int getchar`
  - Returns the next character on standard input or EOF if there are no characters left.
- `int putchar(int c);`
  - Writes the character c on the standard output
- `int printf(char *format,…)`
- `printf(“The result is %f \n”,x);`

C Basics

- Expressions
- abc= x+y*z
- J=a%i
- ++x vs. x++
- X += 5;

  // x = x + 5;
- Y /= z;

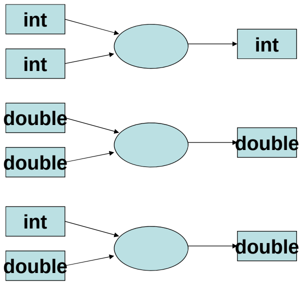
  // Y = Y / z

What is x *= y+1 ?

C Basics

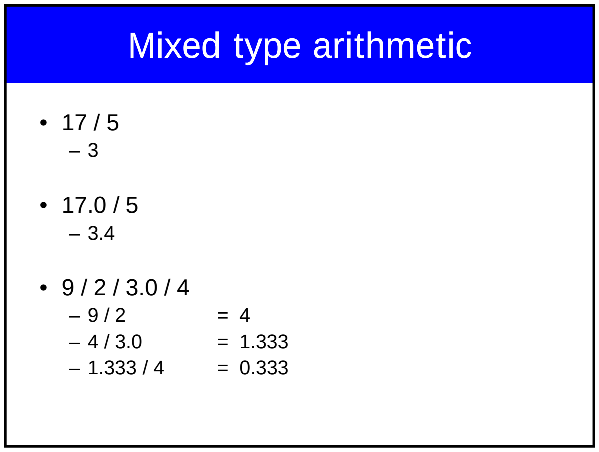
- Decimal numbers 123487
- Octal: starts with 0 0654
- Hexadecimal starts with 0x or 0X ox4Ab2
- 7L for long int =7
- 8U for unsigned
- For floats 24, 23.45, 123.45e-8, 3.4F, 2.15L

Mixed type arithmetic



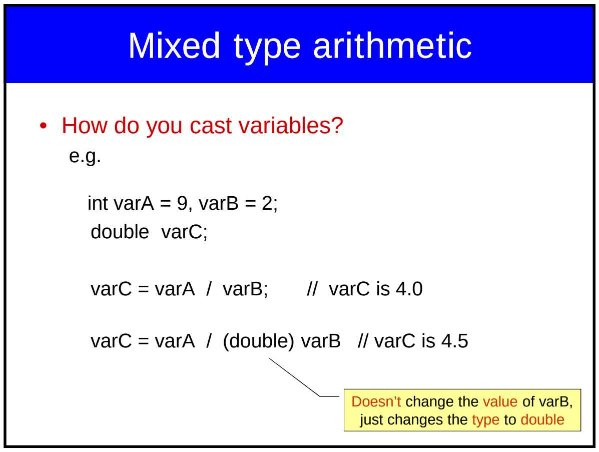
```
int x=5, y=2, w;
double z, q = 2;

z = x/y;
          // z = 2.0
w = x/y;
          // w = 2
z = x/q;
         // z = 2.5
w = x/q;
         // w = 2
```



Mixed type arithmetic

- 17 / 5
  - 3
- 17.0 / 5
  - 3.4
- 9 / 2 / 3.0 / 4
  - 9 / 2 = 4
  - 4 / 3.0 = 1.333
  - 1.333 / 4 = 0.333



Mixed type arithmetic

- How do you cast variables?

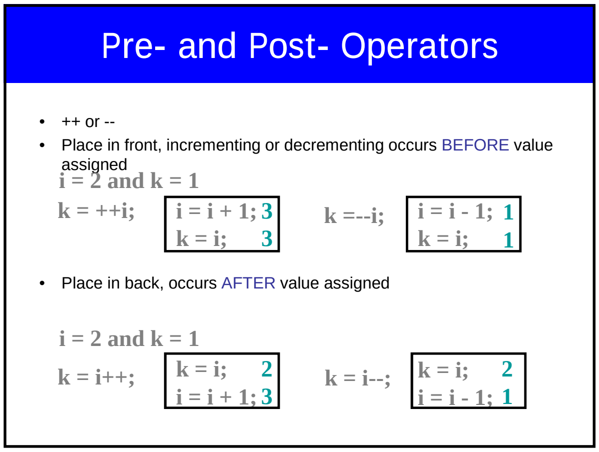
e.g.

```
  int varA = 9, varB = 2;
   double  varC;

   varC = varA  /  varB;       //  varC is 4.0

   varC = varA  /  (double) varB   // varC is 4.5
```

Doesn't change the value of varB, just changes the type to double



Pre- and Post- Operators

- ++ or --
- Place in front, incrementing or decrementing occurs BEFORE value assigned

```
i = 2 and k = 1
k = ++i;    i = i + 1; 3      k =--i;    i = i - 1;  1
            k = i;     3                 k = i;      1
```

- Place in back, occurs AFTER value assigned

```
i = 2 and k = 1
k = i++;    k = i;     2      k = i--;   k = i;      2
            i = i + 1; 3                 i = i - 1;  1
```

Precedence

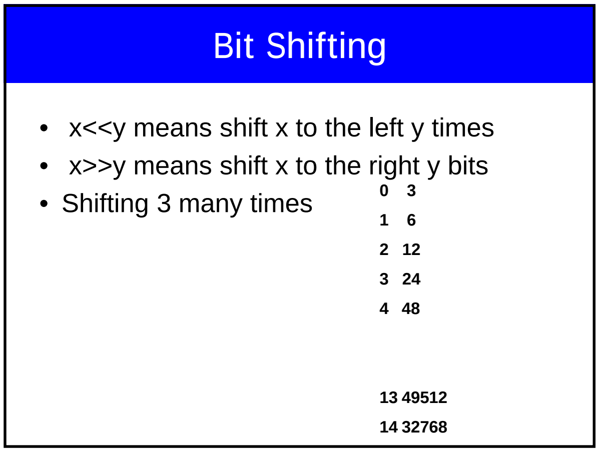
( )	Parentheses	L to R	1
++, - -	Postincrement	L to R	2
++, - -	Preincrement	R to L	3
+, -	Positive, negative	L to R	3
*, /, %	Multiplication, division	L to R	4
+, -	Addition, subtraction	L to R	5
<=, >=, >, <	Relational operator	L to R	6
==, !=	Relational operator	L to R	7
&&	Logical AND	L to R	8
||	Logical OR	L to R	9
+=, -+, *=, /=, %=	Compound assignment	R to L	10
=	Assignment	R to L	10

Examples

- int a=2, b=3; c=5, d=7, e=11, f=3;
- f +=a/b/c; 3
- d -=7+c*--d/e; -3 d=6; 5*6/11 =2; 2+7=9; d=d-9=-3
- d= 2*a%b+c+1; 7
- a +=b +=c +=1+2; 13

Bitwise Operators

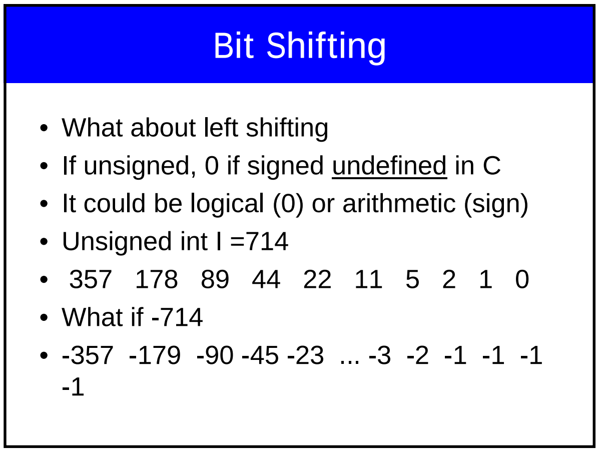
- Works on the individual bits
- &, |, ^, ~
- short int i=5, j=8;
- k=i&j;
- k=i|j;
- k=~j;



Bit Shifting

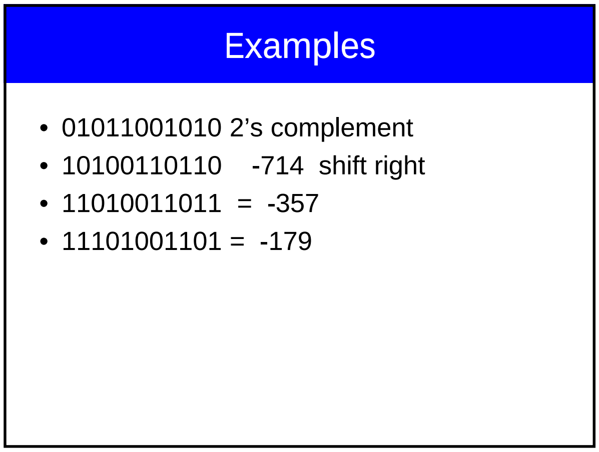
- x<<y means shift x to the left y times
- x>>y means shift x to the right y bits
- Shifting 3 many times

0	3
1	6
2	12
3	24
4	48
13	49512
14	32768



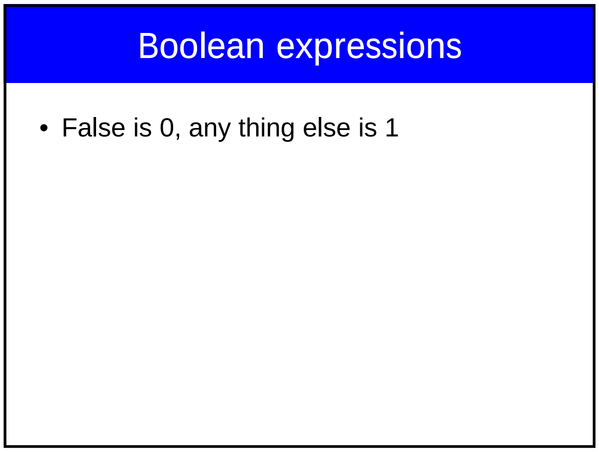
Bit Shifting

- What about left shifting
- If unsigned, 0 if signed undefined in C
- It could be logical (0) or arithmetic (sign)
- Unsigned int I =714
- 357 178 89 44 22 11 5 2 1 0
- What if -714
- -357 -179 -90 -45 -23 ... -3 -2 -1 -1 -1 -1



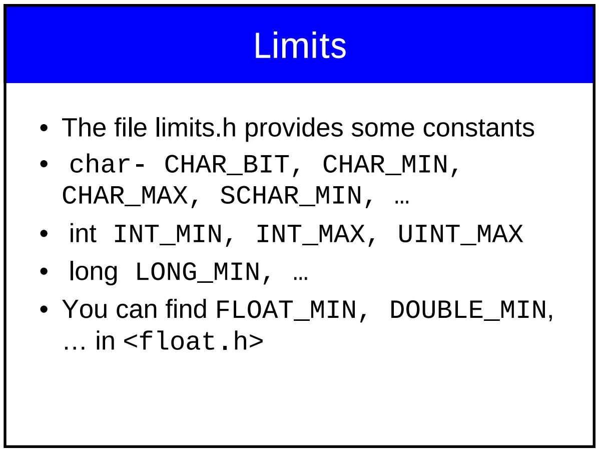
Examples

- 01011001010 2's complement
- 10100110110 -714 shift right
- 11010011011 = -357
- 11101001101 = -179



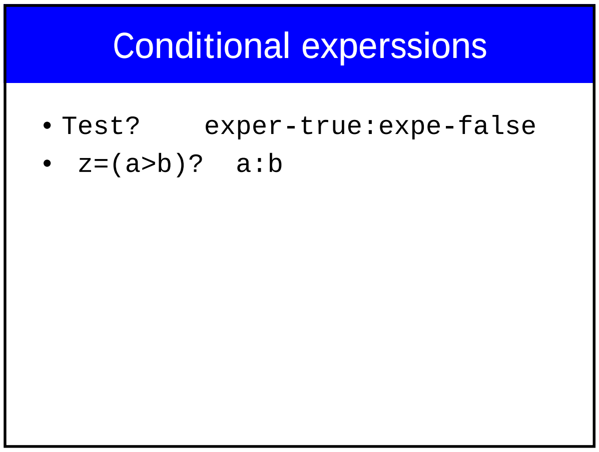
Boolean expressions

- False is 0, any thing else is 1



Limits

- The file limits.h provides some constants
- `char- CHAR_BIT, CHAR_MIN, CHAR_MAX, SCHAR_MIN, …`
- int `INT_MIN, INT_MAX, UINT_MAX`
- long `LONG_MIN, …`
- You can find `FLOAT_MIN, DOUBLE_MIN,` … in `<float.h>`



Conditional experssions

- `Test?    exper-true:expe-false`
- `z=(a>b)?  a:b`

Streams and Files

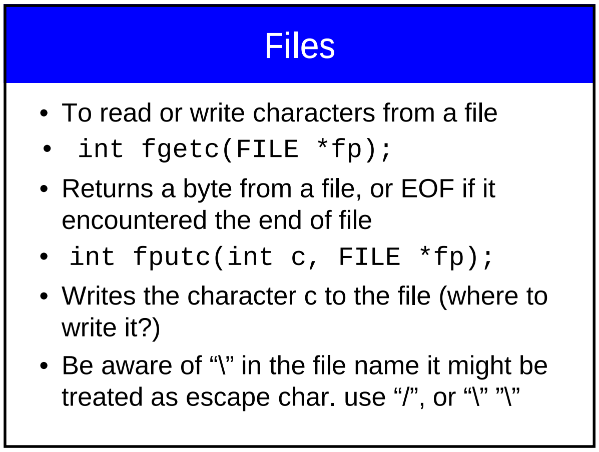
- **Stream**: any source of input or any destination for output.
- Files, but could be also devices such as printers or network ports.
- Accessing streams is done via *file pointer* that is of type `FILE *`.
- Standard streams `stdin, stdout, stderr.`

Files

- You must open the file before you read or write to it (what about stdin, …).
- The system checks the file, and returns a small non-negative integer known as **file descriptor**, all reads and writes are through this file descriptor.
- 0,1,2 are reserved for stdin, stdout, and stderr.

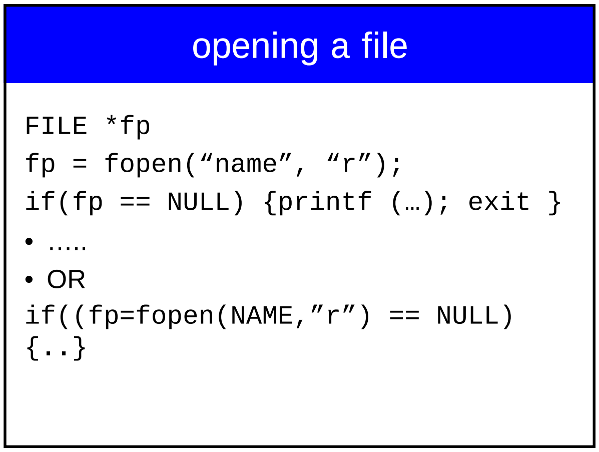
Files

- `FILE *fp1;`
- `FILE *fopen(char *name, char *mode)`
- `fp1=fopen(name, mode);`
- **Do not assume file will open, always check for a null pointer.**
- Name is a character string containing the name of the file, mode is a character string to indicate how the file will be used
- Mode could be “r”, “w”, “a”, “r+”, ...



Files

- To read or write characters from a file
- `int fgetc(FILE *fp);`
- Returns a byte from a file, or EOF if it encountered the end of file
- `int fputc(int c, FILE *fp);`
- Writes the character c to the file (where to write it?)
- Be aware of “\” in the file name it might be treated as escape char. use “/”, or “\” ”\”

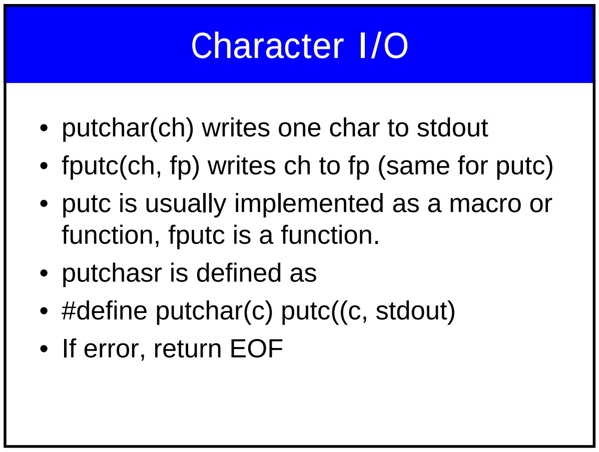


opening a file

```
FILE *fp
fp = fopen(“name”, “r”);
if(fp == NULL) {printf (…); exit }
```

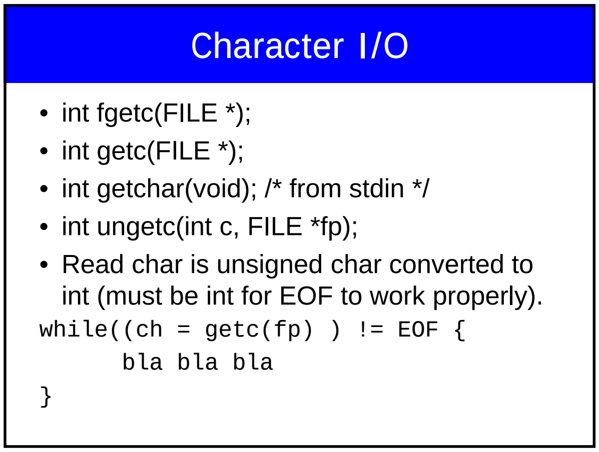
- …..
- OR

```
if((fp=fopen(NAME,”r”) == NULL)
{..}
```



Character I/O

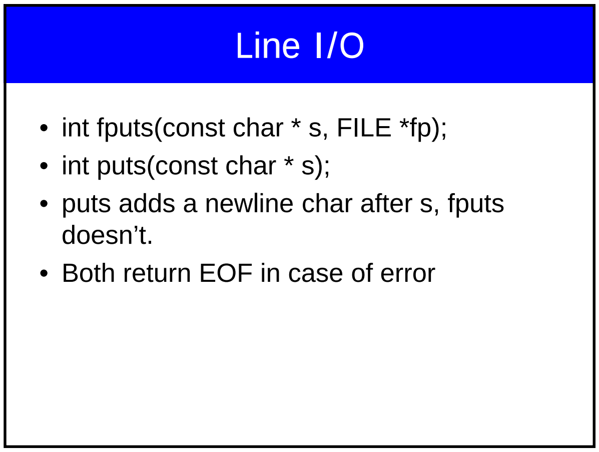
- putchar(ch) writes one char to stdout
- fputc(ch, fp) writes ch to fp (same for putc)
- putc is usually implemented as a macro or function, fputc is a function.
- putchasr is defined as
- #define putchar(c) putc((c, stdout)
- If error, return EOF



Character I/O

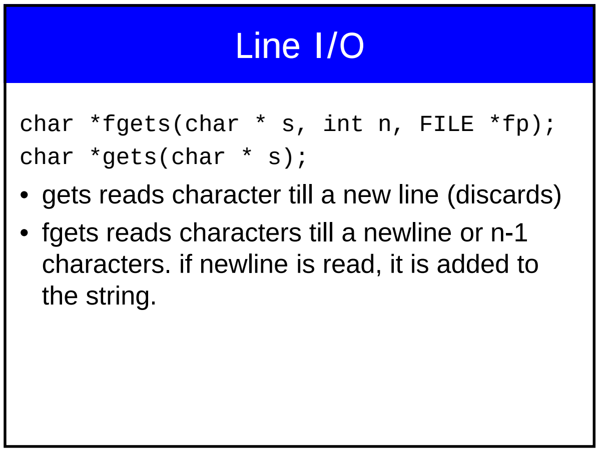
- int fgetc(FILE *);
- int getc(FILE *);
- int getchar(void); /* from stdin */
- int ungetc(int c, FILE *fp);
- Read char is unsigned char converted to int (must be int for EOF to work properly).

```
while((ch = getc(fp) ) != EOF {
      bla bla bla
}
```



Line I/O

- int fputs(const char * s, FILE *fp);
- int puts(const char * s);
- puts adds a newline char after s, fputs doesn’t.
- Both return EOF in case of error



Line I/O

```
char *fgets(char * s, int n, FILE *fp);
char *gets(char * s);
```

- gets reads character till a new line (discards)
- fgets reads characters till a newline or n-1 characters. if newline is read, it is added to the string.

Block I/O

```
size_t fread(void * ptr, size_t
size, size_t nmemb, FILE *fp);
size_t fwrite(void * ptr, size_t
size, size_t nmemb, FILE *fp);
```

- return the actual number of elements read/written.

Position in Files

- int fseek(FILE *stream, long offset, int whence);
- The fseek() function shall set the file-position indicator for the stream pointed to by stream. If a read or write error occurs, the error indicator for the stream shall be set and fseek() fails.
- The new position, measured in bytes from the beginning of the file, shall be obtained by adding offset to the position specified by whence. The specified point is the beginning of the file for SEEK_SET, the current value of the file-position indicator for SEEK_CUR, or end-of-file for SEEK_END.

Position in File

- some problems when dealing with text files.
- See example in the lecture.

Formatted I/O

- we can use fprintf and fscanf with the first parameter a file pointer.
- Error?

Formatted I/O

- for scanf and fscanf, error may be
- *End-of-file* `feof(fp)` returns a non-zero value
- *Read error* `ferror(fp)` returns a non-zero value
- *A matching error*, neither of the above two indicators returns a non-zero.