LECTURE 7:

FILE I/O

COMP1002J: Introduction to Programming 2

Dr. Brett Becker (brett.becker@ucd.ie)

Beijing Dublin International College

Standard Input and Standard Output

- When we get input from a user using `scanf()`, `gets()` or `getchar()`, our program will read this from **standard input** (sometimes known as "stdin").
- By default, when we read from standard input, the user should type some input into the terminal.
- However, we can use standard input to read other types of data also.
- Similarly, when we use `printf()`, our program will print to **standard output**.
- By default, this is printed to the terminal/console window.

Basic File Handling: Redirect Input

- We can use `getchar()` and `putchar()` to build useful programs to process files.

- There is a simple trick that we can use to tell the operating system to take input from a file instead of the keyboard.
  - This is called **redirecting** standard input.
  - To redirect input to a file we use a less-than sign '<'

```
C:\> display < test.txt
```

  - Note: This is not C code – this is the command to run our program:
    - "`display`" is the name of the program
    - "`test.txt`" is the name of the file we want to read

Redirecting Output

- A similar trick can be used to direct output to a file instead of the console.
  - To redirect standard output we use a greater-than sign ‘>’

```
C:\> display > output.txt
```

  - Again, this is not C code:
    - “`display`” is the name of the program
    - “`output.txt`” is the name of the file we want to write to.
  - Now everything we print to standard output (by using `printf()` or `putchar()`) will be saved in that file.

Basic File Handling: Example

```
#include <stdio.h>
int main()
{
  int c;
  c = getchar();
  while (c != EOF)
  {
       putchar(c);
       c = getchar();
  }
}
```

Try the following:

- Copy this program and save it as “`display.c`”.
- Compile the program.
- Type the following line at the command prompt:

  ```
  display < display.c
  ```
- What do you see?

file: display.c

Basic File Handling: Example

- **What is EOF?**

  - EOF is a special value (often -1) that is used to indicate that there are no more characters in a file.
    - It stands for ‘End of File’
  - On a keyboard, we can generate an EOF symbol by using CTRL+D (sometimes CTRL+Z).
  - When redirecting standard input to a **file**, the operating system sends the EOF value **immediately** after the last character of the file.
  - The constant EOF is defined in the `stdio.h` library.

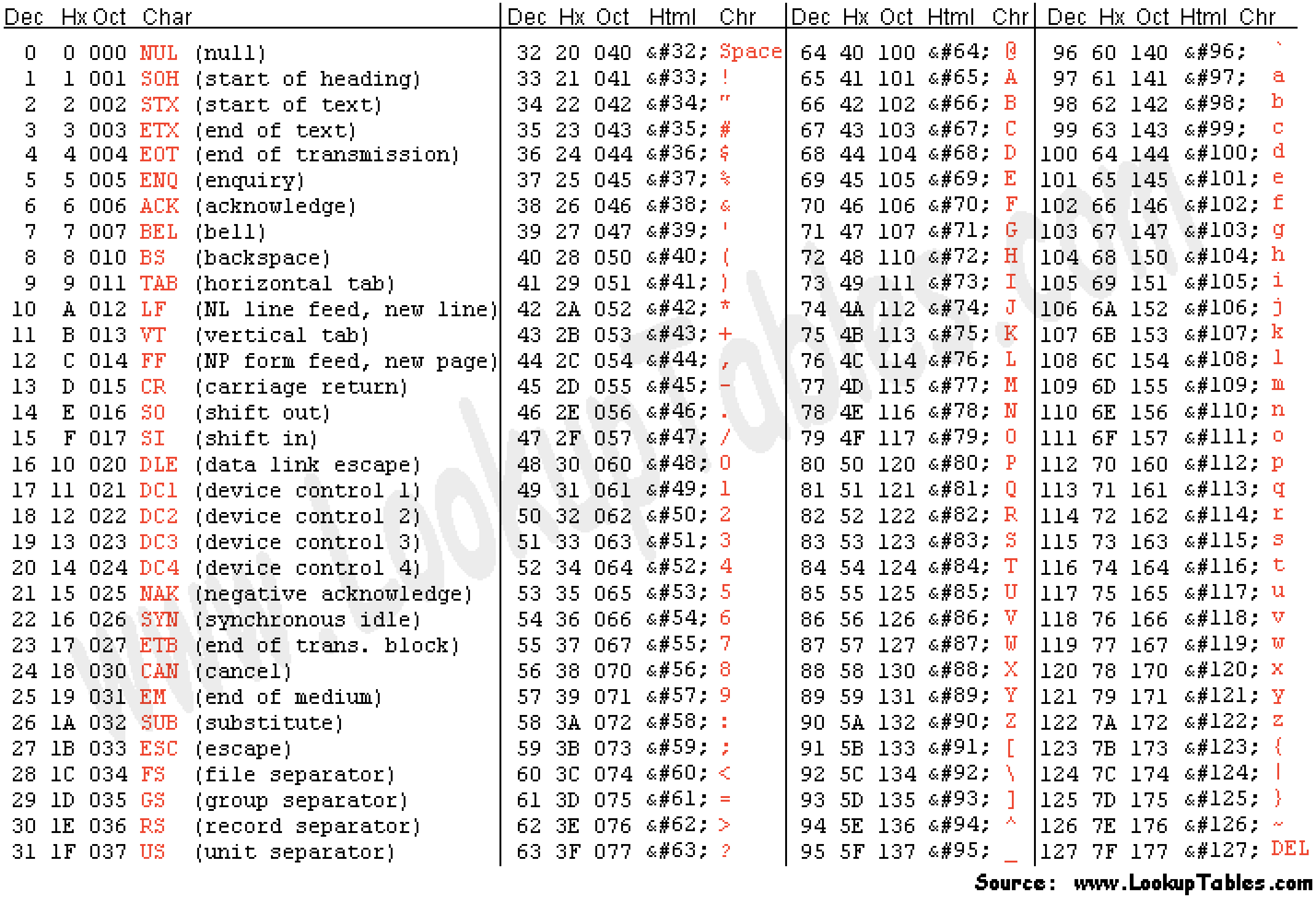
Basic File Handling: Example

```
#include <stdio.h>
int main()
{
  int c;
  c = getchar();
  while (c != EOF)
  {
      putchar(c);
      c = getchar();
  }
}
```

- **Why is c an int?**
  - The `char` data type only allows values in the range 0-255. The value of EOF is normally -1. This cannot be read as a `char`.
  - Instead, `getchar()` returns an `int` value (not a `char`). When the result of `getchar()` is stored in a `char`, the value is converted (cast) from an `int` to a `char`, which is a problem since that can't store -1.

Remember that word?

Remember: Characters are stored as numbers!



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	&#50;	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

Basic File Handling: Example

```
#include <stdio.h>
main()
{
  int c;
  c = getchar();
  while (c != EOF)
  {
       putchar(c);
       c = getchar();
  }
}
```

- **Now, try the following:**
- Type the following line at the command prompt:

```
display < display.c > new.c
```

- Next, type:

```
display < new.c
```

- What happened?

What does this program do?

```
#include <stdio.h>

main(){
  int c;
  int x = 0;
  int y = 0;
  c = getchar();
  while (c != EOF) {
      x++;
      if (c == '\n') y++;
      c = getchar();
  }
  if (x != 0) {
      printf(“x = %d \n", x);
      printf(“y = %d \n", y);
  }
  else
      printf("No result \n");
}
```

Try:

- counter < new.c
- counter < counter.c

What does

- counter < counter.exe

do?

file: counter.c

Basic File Handling

```
#include <stdio.h>

main(){
  int c;
  int num_lines = 0;
  int num_characters = 0;
  c = getchar();
  while (c != EOF) {
      num_characters++;
      if (c == '\n') num_lines++;
      c = getchar();
  }
  if (num_characters != 0) {
      printf("There are %d characters \n", num_characters);
      printf("There are %d lines \n", num_lines);
  }
  else
      printf("No data to count \n");
}
```

Try:

counter2 < counter.c

counter2 < counter2.c

file: counter2.c

Common C Convention

- Reading a character is often combined with testing for EOF:

```
while ( (c = getchar() ) != EOF )
{
    if ( c == '\n' ) num_lines++;
    num_characters++;
}
```

- This code is a little harder to understand, but removes one of the `getchar()` lines from the program.

Common C Convention

- NOTE: This means that assignments are also expressions.
  - *The value of an assignment is the value of expression on the right-hand side of the assignment.*
  - This is why i++ can be used as an index of an array…
  - **THIS IS NOT RECOMMENDED IN GENERAL**

Today’s aside: adding 1 to things

- There are many ways to add 1 to a variable (with one reasonable line of code):
  - a = a+1
  - a++;
  - ++a;
  - a += 1
  - a = (-(~a))
  - …

  - The last one was just for fun. See http://www.geeksforgeeks.org/add-1-to-a-given-number/ for details.

file: plus_one.c

File I/O

- So far, all the programs we have written have taken input from the standard input (keyboard or file) and displayed output to the standard output (console or file).
- Redirecting standard I/O forces all input to come from a file or all output to go to a file.
- Sometimes we want to combine printing to the console AND writing to a file…
- The Standard I/O library (stdio.h) provides a range of functions to support this…

File I/O

- The primary difference between manipulating files and standard I/O is that we must specify, in our programs, which files we wish to use.
- Specifying a file to use is known as **opening** a file.
- When you open a file, you must specify what you wish to do with it (i.e. whether you want to read the file or write to the file).
- You open a file by using the `fopen(…)` function, for example the following statement opens “myfile.txt” for reading:

```
fopen("myfile.txt", "r");
```

File I/O

- In a program, it is possible to open many files at the same time.

- It is not enough simply to open a file; we also need a way of referring to that file so we can read from it and/or write to it later.

- To do this, C provides **file pointers**:

```
FILE *fp, *fp2, ...;
```

- File pointers are created when a file is opened:

```
fp = fopen("myfile.txt", "r");
```

File I/O: Reading & Writing Chars

- Just like we can read characters from and write characters to Standard I/O, we can also read characters from and write characters to a file.

- The function `getc(fp)` is the file I/O equivalent of `getchar()`:
  - The argument identifies *which file* the character is read from.

```
c = getc(fp);
```

- The function `putc(c, fp)` is the file I/O equivalent of `putchar()`:
  - The first argument is the character to be written, and the second identifies the file it is to be written to.

```
putc('a', fp); /* write the character 'a' to the file */
```

File I/O: Example

```
#include <stdio.h>

main()
{
 FILE *fp;
 int c;

 fp = fopen("display.c", "r"); // open file
 while ( (c = getc(fp)) != EOF)
 {
    putchar(c); // print to standard output
 }
 fclose(fp); // close the file
}
```

file: io_example.c

File I/O: Example

- The program reads the contents of the file “display.c” and prints it out to the console.
- When working with files, you must not only open the file, but you must also close the file. This is done by the `fclose(…)` function.
- This must be done for two reasons:
  - Closing the file destroys the file pointer; failing to do this can cause problems if your program runs for a long time…
  - If you open the file for writing, the operating system can lock the file. Sometimes the lock is not released, meaning that you cannot reopen the file…
- What happens if you change the name to a file that does not exist?
  - If opened for reading, you get a segmentation fault when you run the program
  - If opened for writing, the file gets created!

File I/O: Testing the File Pointer

- We can check if a file was successfully opened by checking the file pointer.
- If the value of a pointer is null, then the pointer is not pointing at anything:

```
fp = fopen ( "file.txt", "r" );
if ( fp == NULL ) {
  printf( "Cannot open file.txt for reading\n" );
  exit(1);
}
```

Using the `exit()` Function

- The `exit()` function can be used to immediately stop a program.
- The function takes one integer parameter that is used to indicate whether the program terminated successfully (0) or failed (1).
- Two constants EXIT_SUCCESS and EXIT_FAILURE are defined for this argument.
- To use the exit function you should first include the Standard Library:

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

File I/O

- The following code prompts the user to enter a filename:

```
char  filename[80];
FILE *fp;
int c;

printf( "Enter the file to display: " );
gets(filename); // read from standard input (the user)
fp = fopen(filename, "r");
if (fp == NULL)
{
    printf("Could not open file %s\n", filename);
    exit(EXIT_FAILURE);
}
```

- How would you modify this program to make the user to enter a valid filename?

File I/O

- To make the user enter a valid filename, we should use a loop:

```
while ((fp = fopen(filename, "r")) == NULL)
{
  printf("Cannot open %s for reading \n", filename);
  printf("\nEnter filename: ");
  gets(filename);
}
```

- How would you modify this program to prompt the user 3 times and then quit if it is not done…?

File I/O: Key Rule

ALWAYS

check when opening files that `fopen()` succeeds in opening the file

Obeying this rule will save you heartache in debugging your file handling programs!

Common C Conventions

- Only use these if you understand how they work!
- Opening a file:

```
if ( (fp = fopen( filename, "r" )) == NULL )
{
    printf("Cannot open %s for reading \n", filename );
    exit(EXIT_FAILURE);
}
```

- Combining reading a character with the EOF test:

```
while ( (c = getc( fp ) ) != EOF )
{
    if ( c == '\n' ) lines++;
    num_chars++;
}
```

File Handling: Challenge

- Write a program to display its input contents 10 lines at a time. The program should pause after displaying 10 lines until the user presses either Q to quit or Return to display the next 10 lines.

- Sketch of Solution (Pseudo-code):

```
read character from file
while ( (not end of file) and (user not finished) )
    display character
    if character is newline then
        linecount = linecount + 1;
    if (linecount == 10) then
        linecount = 1;
        Prompt user and get reply;
     read next character from file
```