8.2 Low Level I/O - Read and Write

- “Input and output uses the read and write system calls, which are accessed from C programs through two functions called read and write.”
- “ For both, the first argument is a file descriptor. The second argument is a character array in your program where the data is to go to or to come from. The third argument is the number is the number of bytes to be transferred.”

```
int n_read = read(int fd, char *buf, int n);
int n_written = write(int fd, char *buf, int n);
```

PURDUE
UNIVERSITY

- int n_read = read(int fd, char *buf, int n);
- “Each call returns a count of the number of bytes transferred. On reading, the number of bytes returned may be less than the number requested. A return value of zero bytes implies end of file, and -1 indicates an error of some sort.”
- For writing, the return value is the number of bytes written; an error has occurred if this isn't equal to the number requested.
- int n_written = write(int fd, char *buf, int n);

- “Any number of bytes can be read or written in one call. The most common values are 1, which means one character at a time (``unbuffered''), and a number like 1024 or 4096 that corresponds to a physical block size on a peripheral device.”
- “Larger sizes will be more efficient because fewer system calls will be made.”
- Using these two syscalls, we can write a simple program to copy its input to its output, the equivalent of the Unix command *“cat”*. (I modified the code to fit my system.)
- “This program will copy anything to anything, since the input and output can be redirected to any file or device.”

```
/* #include "syscalls.h” I don’t need
this on my machine*/
#define BUFSIZ 100
main() /* copy input to output */
{
char buf[BUFSIZ];
int n;
while ((n = read(0, buf, BUFSIZ)) > 0)
    write(1, buf, n);
return 0;
}
```

PURDUE
UNIVERSITY

- Let us try compile and run the program
  - First by typing text on the keyboard
  - Next by copying a JPEG image to another
- Let us use the command “size” to see the size of the machine code (in binary) as it is loaded to the memory for execution
  - We see it is only 1823 bytes
  - The smallest jvm would be larger than this by a factor of a few dozens
- The size is different between the “size” showing and the “ls” command showing, because the a.out file has other things to assist loading

- From this simple program we see several features in C
  - Comments
  - Macros
  - The main fuction
  - Declaration statements and executable statements
- We briefly explain how the main function gets invoked, but leave the details to the OS course
- Next let's try to "view" some non-text file by "vi" or "more".
  - We'll see garbled output
  - What is a text file?
    - A sequence of ASCII characters, each taking a byte.

CHARACTER REPRESENTATION ASCII

ASCII (American Standard Code for Information Interchange)

MSB (3 bits)

LSB (4 bits)	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	‘	P
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	“	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	‘	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	I	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	I	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	m	n	~
F	SI	US	/	?	O	n	o	DEL

PURDUE UNIVERSITY

char

- The basic unit of data on the UNIX system is a byte (or *char* in a C program*)*
  - *Does not have to be an ASCII char*
- Therefore it is useful to have a high-level C function, called *getchar*, to read one character at a time from the standard input

```
/* getchar: unbuffered single character input */
int getchar(void)
{
char c;
return (read(0, &c, 1) == 1) ? (unsigned char) c : EOF;
```

PURDUE UNIVERSITY

- In this short function, we see
  - A new data type called “void”
  - A strange expression that has “?”
  - A typecast operation
  - A new data type called “unsigned char”
  - A symbolic constant EOF

Be very careful and remember that getchar returns an integer!

- Some of these things will be explained later in the course
  - For historical reasons (mainly to help the compiler when compiler techniques are still not mature), the C language has quite some “features” that are deemed by many as error prone.
  - One of the things we must do in this course is to point out the potential pitfalls
  - It is a good idea to avoid using cryptic syntax in a C program

Standard C Functions

- C comes with a set of prewritten standard functions that can be “included” in the program we write
- E.g. a much more sophisticated version of *getchar* is a part of the “standard I/O functions”

#include <stdio>

- The “include” macro tells the C compiler’s *pre-processor* to include (i.e. to *inline*) the set of I/O standard functions in the program text before compiling
- *printf* is yet another highly useful standard I/O function in <stdio>
- We now go back to the beginning of the book and trying out more program examples and explain them.

PURDUE UNIVERSITY

Using printf

```
#include <stdio.h>
main()
{
printf("hello, world\n");
```

- *printf* is actually a quite complex function. It is a “formatted output function”, to make our task of printing various types of data easy.
- *printf* is a special case of *fprintf*
- *int fprintf(FILE *stream, const char *format, ...)*

PURDUE UNIVERSITY

- It is notable that *fprintf* has a variable length of parameters
  - After the format parameter, there may be 0 or more variables as parameters
- Let us go to Appendix B.1 to learn something about <stdio.h>
  - “A stream is a source or destination of data that may be associated with a disk or other peripheral. The library supports text streams and binary streams, although on some systems, notably UNIX, these are identical.”
  - “A text stream is a sequence of lines; each line has zero or more characters and is terminated by '\n'. An environment may need to convert a text stream to or from some other representation (such as mapping '\n' to carriage return and linefeed).”

- “A stream is connected to a file or device by opening it; the connection is broken by closing the stream. Opening a file returns a pointer to an object of type FILE, which records whatever information is necessary to control the stream. We will use ``file pointer'' and ``stream'' interchangeably when there is no ambiguity.”
- “When a program begins execution, the three streams stdin, stdout, and stderr are already open.”
- Formatted output:

  int fprintf(FILE *stream, const char *format, ...)

  int printf(const char *format, ...)

  printf(...) is equivalent to fprintf(stdout, ...).

PURDUE UNIVERSITY

- char *format means the (parameter) variable “format” is a pointer to a sequence (of unknown length) of characters
- const *format means the parameter is a quoted string
- So, printf expects the first argument to be a quote string which may include textual characters, escape sequences (e.g. \n), and *conversion characters*.
  - *Each conversion character defines the formatting specification for the corresponding parameter that follows the “format” parameter*
  - Our current example contains no conversion characters.
  - String is perhaps the most complicated (and difficult to understand) data type in C, due to its variable length that is often not predetermined.
  - Please follow Section 1.2 and try out variants of our example yourself

PURDUE UNIVERSITY

Example of using format specification

```
#include <stdio.h>
/* print Fahrenheit-Celsius table
for fahr = 0, 20, ..., 300 */
main()
{
  int fahr, celsius;
  int lower, upper, step;
  lower = 0; /* lower limit of temperature scale */
  upper = 300; /* upper limit */
  step = 20; /* step size */
  fahr = lower;
  while (fahr <= upper) {
      celsius = 5 * (fahr-32) / 9;
      printf("%d\t%d\n", fahr, celsius);
     fahr = fahr + step;
  }
}
```

PURDUE UNIVERSITY

- We will not explain in length syntax that is similar to Java
- We see a new escape sequence \t for printing a tab
- There are two conversion characters used in the quoted string corresponding to fahr and celsius respectively
  - %d (same as %i) is for int; signed decimal notation.
- For other conversion characters, see Table B.1

A floating point number example

```
#include <stdio.h>
/* print Fahrenheit-Celsius table
for fahr = 0, 20, ..., 300; floating-point version */
main()
{
   float fahr, celsius;
   float lower, upper, step;
   lower = 0; /* lower limit of temperature scale */
   upper = 300; /* upper limit */
   step = 20; /* step size */
   fahr = lower;
   while (fahr <= upper) {
       celsius = (5.0/9.0) * (fahr-32.0);
       printf("%3.0f %6.1f\n", fahr, celsius);
      fahr = fahr + step;
   }
}
```

PURDUE UNIVERSITY

- In this example, we see
  - Implicit type conversion (from int to float)
  - The floating point format specification
  - Width modifier for the conversion characters
    - You can find the description of the width modifier in the book
- There are five different sizes of integer types (signed and unsigned):
  - char, short, int, long, and long long
  - Internally, char is a single byte, others depending on whether the machine is a 32-bit or a 64-bit machine, e.g. on a 32-bit machine, int is normally 4 bytes, long could also be 4 bytes, short is 2 bytes, long long could be 8 bytes

A For statement example

```
#include <stdio.h>
/* print Fahrenheit-Celsius table */
main()
{
int fahr;
for (fahr = 0; fahr <= 300; fahr = fahr + 20)
    printf("%3d %6.1f\n", fahr, (5.0/9.0)*(fahr-32));
}
```

PURDUE
UNIVERSITY

- In this example, we see
  - the For loop header that has three components
    - Initial value of an induction variable, fahr,
    - The termination condition, fahr <= 300
    - The increment operation for the induction variable
    - Any component could be left as blank
  - and the loop body, which is a C statement
    - It could be a compound statement with { } to enclose a list of C statements
  - We also see an expression used as one of the arguments in printf

PURDUE
UNIVERSITY

Character I/O

```
#include <stdio.h>
/* copy input to output; 1st version
*/
main()
{
    int c;
    c = getchar();
    while (c != EOF) {
        putchar(c);
        c = getchar();
    }
}
```

PURDUE
UNIVERSITY

- int getchar(void)
- int putchar(int c)

- Please try out the examples and exercises in the textbook up to, and including, section 1.5.
- Lab 0 will be a warm up lab (not graded) to try out the submission and autograder
- Lab 0 will be the base for the graded Lab 1 in the week that follows Lab 0.
  - Therefore, it is a good idea to work on Lab 0 seriously.

PURDUE
UNIVERSITY