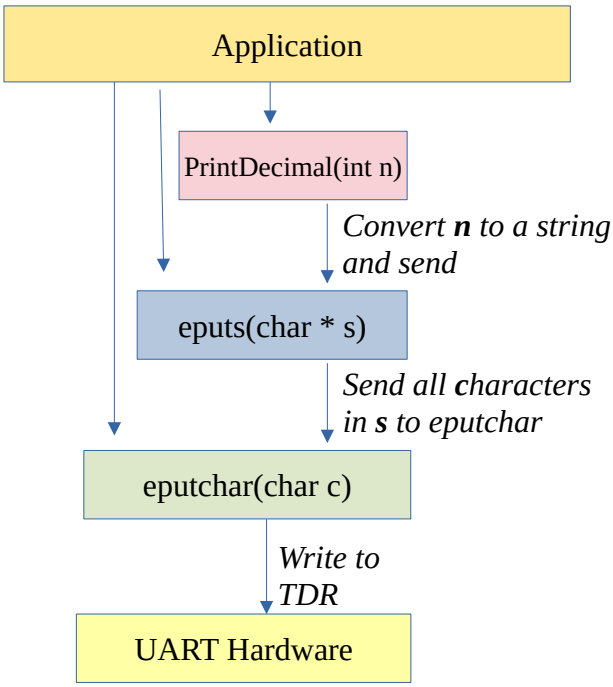
Serial communications



The figure above shows the layout of a layered communications stack. Hardware is a the bottom layer in the form of a U(S)ART module within the microcontroller. This hardware is "wrapped" by a function (**eputchar**) that can be used to send any character. Higher level functions are built on top of **eputchar**. These function can send strings and integers. Designing our software in this way improves its portability across different hardware. If we change microcontroller we simply need to rewrite the function **eputchar**.

The function eputchar is defined as follows:

```
void eputchar(char c)
{
	while( (USART2->ISR & (1 << 6))==0); // wait for ongoing transmission to finish
	USART2->TDR = c;
}
```

The first line of this consists of a while loops that waits to see if another character is currently being sent by the USART. If that is the case, the function waits until the character has gone. Without this check it is possible for one outbound character to partially overwrite another leading to data corruption.

The ISR register in the USART contains the following bits:

24.8.8 Interrupt and status register (USART_ISR)

Address offset: 0x1C

Reset value: 0x0200 00C0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	TCBGT	Res.	Res.	REACK	TEACK	WUF	RWU	SBKF	CMF	BUSY
						r			r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ABRF	ABRE	Res.	EOBF	RTOF	CTS	CTSIF	LBDF	TXE	TC	RXNE	IDLE	ORE	NF	FE	PE
r	r		r	r	r	r	r	r	r	r	r	r	r	r	r

The USART ISR from the STM32L031 reference manual

The **eputchar** function checks bit 6 : the transmit-complete (TC) bit. When this becomes a logic 1 then it is safe to write new data into the Transmit Data Register (TDR).

The function **egetchar** is defined as follows:

```
char egetchar()
{
	while( (USART2->ISR & (1 << 5))==0); // wait for character to arrive
	return (char)USART2->RDR;
}
```

This function waits for a character to be received by checking bit 5 of the ISR. This is the "Receiver Not Empty" bit. When this is a 1 then new data is available for reading from the Receive Data Register (RDR).

The function **initSerial** sets communications going and configures the various I/O pins for operation with the USART. The data rate is set to 9600 bits per second

Procedure

Download the Serial_L031 starter code from brightspace and run it on your MCU. Use the built-in terminal applet in Visual Studio Code to interact with the serial output from the MCU.
Examine the way the printDecimal function works
Can you get it to work without printing leading zeros for low numbers?
Can you create a function called printHex which can print unsigned 32 bit hexadecimal values?

Modify your main loop as follows:

```
char ch;
while(1)
{
	ch = egetchar();
	eputchar(ch);
	eputs("\r\n");
}
```

Add an LED and a resistor to your board on PA0. Initialize this pin as an output. Modify the code so that when a user presses the letter 'a' the LED turns on, otherwise it is turned off.
Can you make it case insensitive?