There are 3 registers associated with ADC:

ADMUX

1. Set the `ADLAR` bit to left or right justify the results. You would justify if you only need 8 bits of precision and just use the `ADCH` byte. When you right justify, use both the `ADCH` and `ADCL` bytes for 10 bits of precision.
   How would the following registers be filled by the ADC hardware when the result of the conversion is `1110001100`?

Left Aligned: `ADLAR=0`

b_{15}	b_{14}	b_{13}	b_{12}	b_{11}	b_{10}	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
0	0	0	0	0	0	1	1	1	0	0	0	1	1	0	0

Right Aligned: `ADLAR=1`

b_{15}	b_{14}	b_{13}	b_{12}	b_{11}	b_{10}	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
1	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0

2. Set `REFS1` and `REFS0` to set your reference voltage. If you choose `00` for these bits you must connect a voltage source to the `AREF` pin externally.
   Write the instruction(s) that initializes the `ADMUX` register so that the reference voltage is `AVCC` (On an Arduino platform, `AVCC` is internally connected to 5V `VCC`)

```
ADMUX |= 0x40;
```

3. Choose your analog source pin (from port C) and enable it using the `MUX3`-`MUX0` bits.
   Write the instruction(s) to select pin PC2 as the analog source pin to the ADC.

```
ADMUX |= 0x02;
```

ADCSRA

1. Choose a clock divider given the frequency of your system clock. Remember that ADCs work best between 50-200kHz. The Arduino clock frequency is 16MHz but other systems may be different. Prescale is set with `ADPS2`-`ADPS0` bits.
   What are the only clock divider options we can use to ensure a frequency between 50-200kHz range on an Arduino platform?
   `111` and `110`, or 64 and 128
2. Do you want to trigger an interrupt when the conversion is complete? If so, set the `ADIE` bit. If you trigger an interrupt, you will also need an ISR.
   What is the name of the interrupt vector for the ADC interrupt?
   `ADC_vect`

3. If you don't use an interrupt, you can poll the `ADIF` bit to know when the conversion is complete. It will go high.
   Write the code for a While loop that waits for the `ADIF` bit to go high.

```
while(ADCSRA & 0x10){}
```

4. Do you want to start the conversion under program control, or upon the occurrence of another event (or free running mode)? Set the `ADATE` bit accordingly- 0 for program controlled conversions, and 1 for free-running mode and event-driven conversions.
   If you are initializing the conversion on the occurrence of another event, what other register needs to be initialized in `setup()`?
   The interrupt that corresponds with the other event. (i.e. The pin-change interrupt or the timer interrupt.)
5. Turn on the ADC circuitry with the `ADEN` bit.
6. Finally, use the `ADSC` to start the conversions. In free running mode, do this at the end of setup. In program controlled conversions, it gets set in the main loop at the time a conversion is needed.
   If you are not in free-running mode, why should you wait to set the `ADSC` bit in `loop()`?

If you don't, then it will only run the conversion once, then stop. Putting it in the loop ensures it runs every instance that you want it to.

ADCSRA

1. If you want free-running mode or an event-triggered interrupt, use the `ADTS2`-`ADTS0` bits to set that.
   Write the instructions to initialize the ADC to start a conversion on a `TIMER1` overflow interrupt.

```
void setup(){
    //assumed timer1 setup here
    //other ADC setup not established in this question
    ADCSRB |= 0x07;
}
```