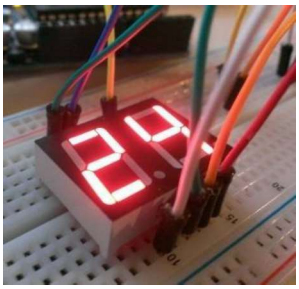
Displays

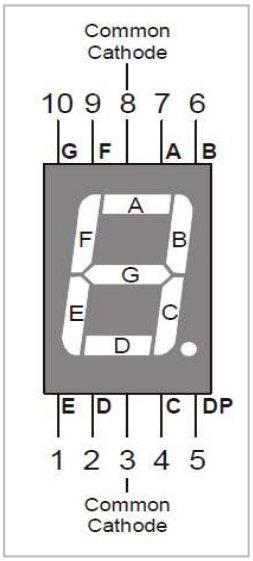
Apesar de existirem diversos atuadores que nos permitam comunicar informações de forma visual, nenhum é tão eficiente e flexível quanto os displays. Seja através de caracteres simples, textos ou até mesmo animações, os displays são amplamente utilizados em projetos de IOT pela facilidade com que os dados, alertas e informações são mostrados ao usuário (principalmente quando queremos criar centrais de controle). Em projetos IOT simples e projetos maker, são três os principais tipos utilizados pela sua facilidade e baixo custo: de 7 segmentos, LCD e OLED.

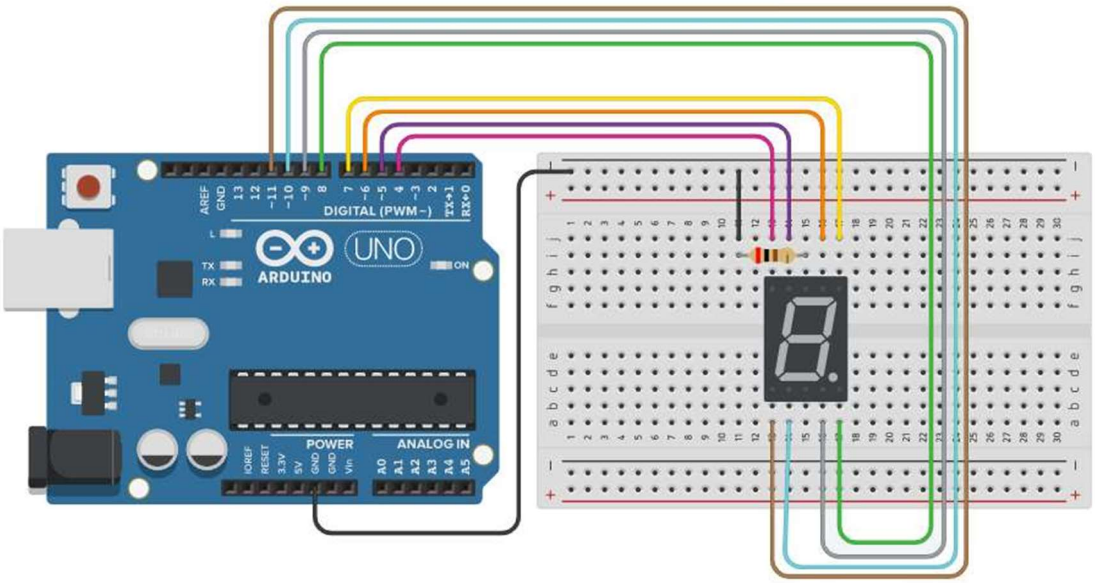
Display de 7 segmentos



O display de 7 segmentos foi o display mais popular na eletrônica ao longo de muitos anos, sendo somente nos últimos 20 anos substituído por tecnologias melhores após a queda acentuada dos seus valores (como os displays LCD e OLED). Seu uso era considerado bastante simples por conta da sua construção.

Basicamente, ele é composto de 7 leds (ou 8 em caso do ponto decimal), cada um posicionado no interior do display em uma concavidade coberta por um material mais escuro (geralmente de plástico ou vidro). Quando cada um dos leds é ligado, percebe-se o seu segmento iluminado. Ele possui um cátodo comum a ser ligado no GND e cada um desses leds possui um pino que deve ser ligado em uma porta que controle o seu acendimento. Deste modo, a construção dos dígitos torna-se uma simples combinação de segmentos iluminados e não iluminados, controlados, no caso do Arduino, pela simples mudança de estados das portas conectadas aos pinos do display. Estes segmentos são representados por letras, para fins de organização. Para ilustrar, é mostrada a imagem ao lado com este esquema de pinos e letras. Se precisássemos gerar o dígito **3**, seria necessário ligar os leds dos segmentos A (pino 7), F (pino 9), E (pino 1), D (pino 2) e G (pino 10). Pela mesma lógica, se precisássemos gerar o dígito **2**, seria necessário ligar os leds dos segmentos A (pino 7), B (pino 6), G (pino 10), E (pino 1) e D (pino 2). Para um exemplo, vamos considerar o esquema eletrônico apresentado a seguir:





Para nosso exemplo, vamos criar um contador que a cada 1 segundo incrementa o dígito, iniciando em zero e terminando com o ponto (após chegar à 9). Uma primeira ação necessária é mapear cada um dos segmentos do display (A, B, C, D, E, F, G e PONTO) com as respectivas portas digitais onde cada um foi conectado, senão o acesso através do número da porta se tornará bem mais difícil. Um método ligaTodos também é necessário para limpar qualquer segmento ligado quando queremos trocar o dígito, evitando repetições nos desligamentos. Em nosso setup, indicamos todos os segmentos (portas) como output e chamamos o método criado para garantir que todos iniciem desligados.

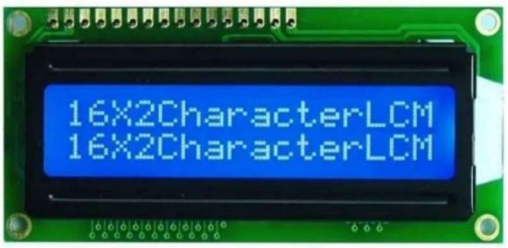
```
#define A 6    //laranja
#define B 7    //amarelo
#define C 9    //cinza
#define D 10   //turquesa
#define E 11   //marrom
#define F 5    //roxo
#define G 4    //rosa
#define P 8    //verde (ponto)

void desligaTodos(){
  digitalWrite(A, LOW);
  digitalWrite(B, LOW);
  digitalWrite(C, LOW);
  digitalWrite(D, LOW);
  digitalWrite(E, LOW);
  digitalWrite(F, LOW);
  digitalWrite(P, LOW);
}

void setup(){
  pinMode(A, OUTPUT);
  pinMode(B, OUTPUT);
  pinMode(C, OUTPUT);
  pinMode(D, OUTPUT);
  pinMode(E, OUTPUT);
  pinMode(F, OUTPUT);
  pinMode(G, OUTPUT);
  pinMode(P, OUTPUT);
  desligaTodos();
}
```

Já dentro do loop, a cada um segundo, um dígito é formado modificando o estado das portas que ligam os segmentos para HIGH e definindo uma espera de 1 segundo após a visualização, seguida de uma chama ao método que desliga todos os segmentos.

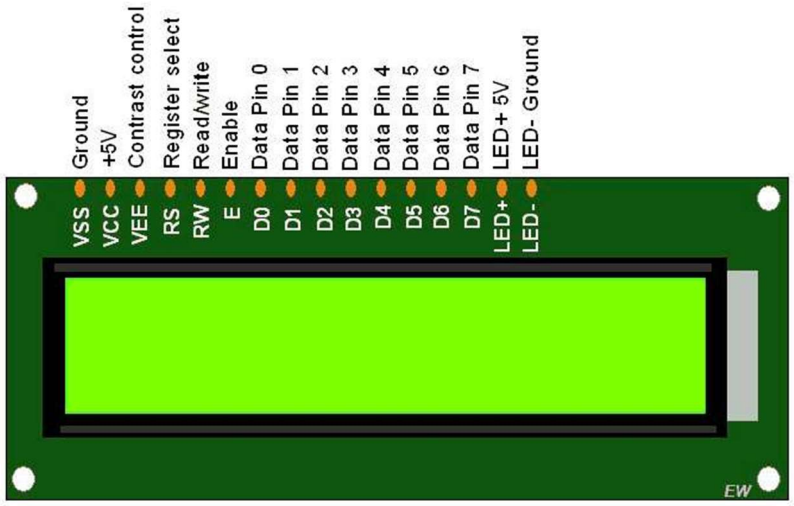
```
void loop(){

    //dígito 0
    digitalWrite(A, HIGH); digitalWrite(B, HIGH); digitalWrite(C, HIGH);
    digitalWrite(D, HIGH); digitalWrite(E, HIGH); digitalWrite(F, HIGH);
    delay(3000); desligaTodos();

    //dígito 1
    digitalWrite(B, HIGH); digitalWrite(C, HIGH);
    delay(3000); desligaTodos();

    //dígito 2
    digitalWrite(A, HIGH); digitalWrite(B, HIGH); digitalWrite(G, HIGH);
    digitalWrite(E, HIGH); digitalWrite(D, HIGH);
    delay(3000); desligaTodos();

    //dígito 3
    digitalWrite(A, HIGH); digitalWrite(B, HIGH); digitalWrite(C, HIGH);
    digitalWrite(D, HIGH); digitalWrite(G, HIGH);
    delay(3000); desligaTodos();

    //dígito 4
    digitalWrite(B, HIGH); digitalWrite(C, HIGH); digitalWrite(F, HIGH);
    digitalWrite(G, HIGH);
    delay(3000); desligaTodos();

    //dígito 5
    digitalWrite(A, HIGH); digitalWrite(F, HIGH); digitalWrite(G, HIGH);
    digitalWrite(C, HIGH); digitalWrite(D, HIGH);
    delay(3000); desligaTodos();

    //dígito 6
    digitalWrite(F, HIGH); digitalWrite(E, HIGH); digitalWrite(D, HIGH);
    digitalWrite(C, HIGH); digitalWrite(G, HIGH);
    delay(3000); desligaTodos();

    //dígito 7
    digitalWrite(A, HIGH); digitalWrite(B, HIGH); digitalWrite(C, HIGH);
    delay(3000); desligaTodos();

    //dígito 8
    digitalWrite(A, HIGH); digitalWrite(B, HIGH); digitalWrite(C, HIGH);
    digitalWrite(D, HIGH); digitalWrite(E, HIGH); digitalWrite(F, HIGH);
    digitalWrite(G, HIGH);
    delay(3000); desligaTodos();

    //dígito 9
    digitalWrite(A, HIGH); digitalWrite(B, HIGH); digitalWrite(C, HIGH);
    digitalWrite(F, HIGH); digitalWrite(G, HIGH);
    delay(3000); desligaTodos();

    //ponto
    digitalWrite(P, HIGH);
    delay(3000); desligaTodos()

}
```

O código por trás do uso do display de 7 segmentos é relativamente simples, sem uma lógica muito elaborada e, portanto, de fácil entendimento. Porém, é bastante trabalhoso e excessivamente grande para a produção de pouco efeito. Obviamente que ele poderia ser melhorado, utilizando-se array para representar as 7 ou 8 posições do display (atribuindo 0 ao segmento desligado e 1 ao segmento ligado) e passando esse array como parâmetro de um método que o percorre dando digitalWrite em cada posição de acordo com o seu valor. Porém o maior problema do uso do display de 7 segmentos não é a programação, mas o número excessivamente grande de portas digitais ocupadas.

Display LCD



Os displays LCD possuem uma matriz de leds que representam pixels, que aparecem quando expostos à uma luz de fundo. Diferente do display anterior, os caracteres não são formados por segmentos simples controlados individualmente. Além de visualmente bonitos, estes displays tem a capacidade de controlar estes pixels de forma transparente ao desenvolvedor, ofertando métodos através dos quais enviamos Strings que são processadas e transformadas automaticamente em caracteres na tela. Quando necessitamos mostrar informações com múltiplos dígitos, o uso de display de 7 segmentos se torna inviável pelo total de portas necessárias e pela quantidade desproporcional de código para implementação da lógica. Os displays LCD tornam-se alternativas mais adequadas, mais simples e visualmente mais bonitas. Antes de vermos seus dois usos mais comuns (de forma direta ou com adaptador I2C), vamos conhecer seu pinout:

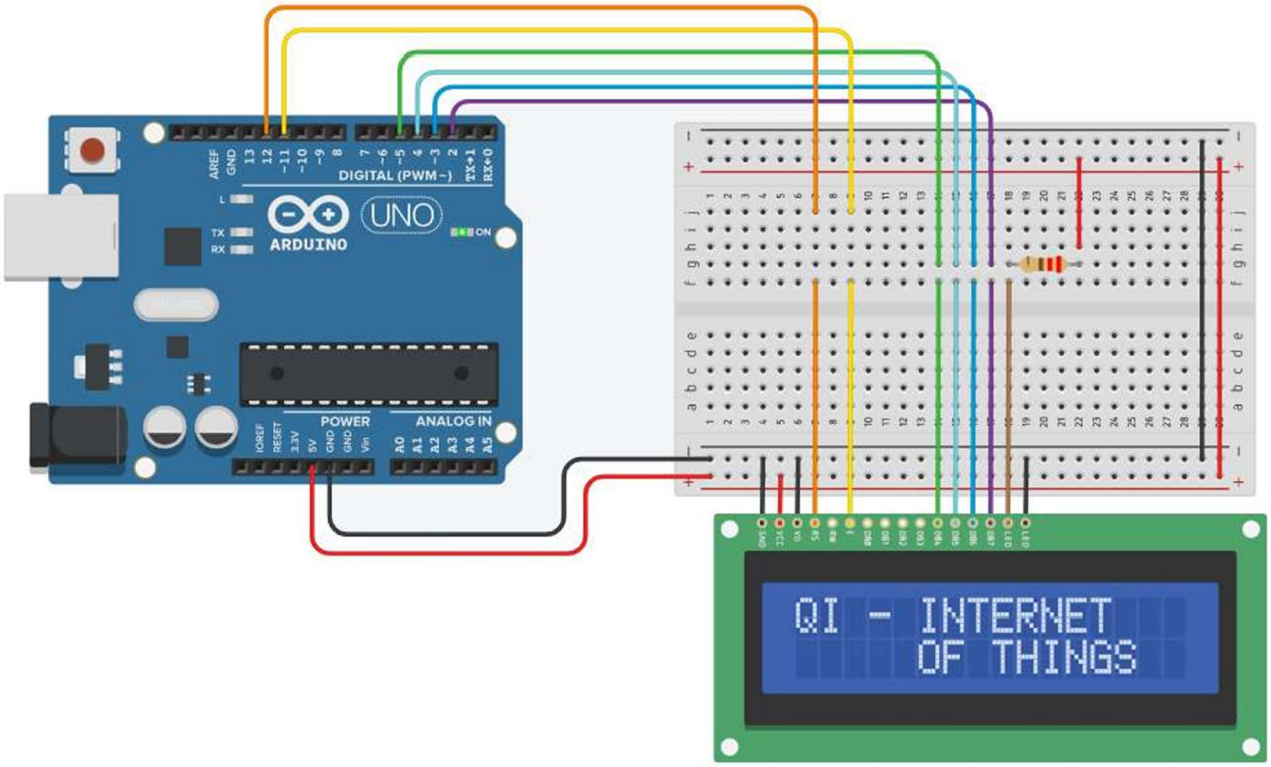


- **VSS:** pino de saída da corrente do circuito do display, ligado ao GND do Arduino;
- **VCC:** pino de alimentação do circuito lógico do display, ligado ao 5V do Arduino;
- **VEE:** pino de contraste entre fundo e letras. Em montagens simples (como as do nosso exemplo), pode ser ligado diretamente ao GND o que configura contraste máximo (para tensão 0). Mas podemos também ligar à uma digital PWM e regular a tensão, regulando assim o contraste.
- **RS:** pino de configuração do display para aceite de instruções ou visualização de dados, sendo a alternância entre estes estados controlada automaticamente pelo circuito interno e Arduino.
- **R/W:** pino de configuração do display para escrita (LOW) ou leitura (HIGH). Como os displays que utilizaremos não são táteis (que possibilitam o funcionamento também como entrada), serão sempre de saída e, portanto, basta não usar a porta e nem energizá-la (LOW).
- **E:** pino de configuração do display como habilitado ou não habilitado para receber nova leitura. É automaticamente controlado pelo circuito interno.
- **D0 à D7:** pinos de dados, por onde o display recebe os bytes dos caracteres a serem exibidos. Caso se deseje trabalhar com o alfabeto convencional e os caracteres especiais existentes em todos os teclados, utilizamos somente 4 bits, necessitando apenas dos pinos D4, D5, D6 e D7. Caso se deseje trabalhar com toda a gama de carateres da codificação UTF-8 (incluindo Kanji, alfabeto cirílico e etc), utilizamos 8 bits e, portanto, os 8 pinos.
- **LED+**: VCC dos leds do display, ligado ao 5V do Arduino
- **LED-**: GND dos leds do display, ligado ao GND do Arduino. Por segurança, é indicado baixar a tensão da porta com um resistor a partir de 100Ω. Isso pode ser feito ou no LED+ ou no LED-.

Conhecendo os pinos do display, fica mais fácil compreender a sua ligação ao circuito eletrônico que enviará as sequências de caracteres que deverão ser mostradas. O seu uso pode se dar de duas formas: direta ou com o uso de uma interface de comunicação I2C. Apesar dos métodos para o seu controle serem iguais, as ligações eletrônicas e bibliotecas são diferentes.

PRIMEIRA FORMA DE USO: DIRETA

Na forma de uso direta, necessitamos da ligação eletrônica de um número considerável de pinos nas portas do Arduino, mas nada muito diferente do necessário para um display de 7 segmentos. Para isso, consideramos as funções vistas no pinout. Teríamos, então, um circuito eletrônico similar a este:



No caso de uma ligação direta, utilizamos a biblioteca LiquidCrystal, nativa da Arduino IDE, sem a necessidade de download. A mesma nos fornece um objeto do tipo LiquidCrystal que representa um display LCD e que quando inicializado, necessita de 6 parâmetros (para usos simples, sem necessidade de cobrir todos caracteres UTF-8): porta do pino RS, porta do pino E e porta dos pinos D4, D5, D6 e D7. Inicializado o objeto (begin, informando o total de linhas e colunas), ele nos permite acesso a alguns métodos como:

```
clear();
```

limpa o display

```
print( <texto> );
```

<texto>
texto a ser escrito no display

```
setCursor( <coluna>, <linha> );
```

<coluna>
Coluna onde desejamos posicionar o display, iniciando em 0

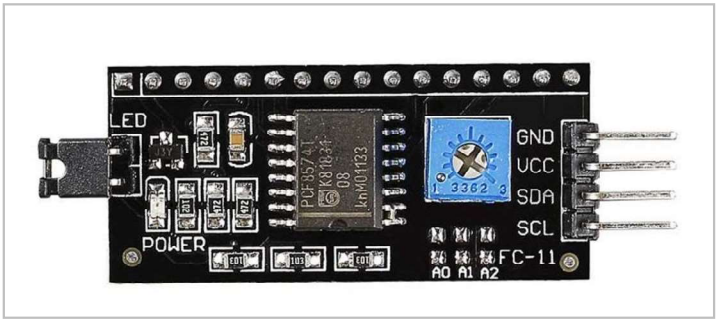
<linha>
linha onde desejamos posicionar o display, iniciando em 0

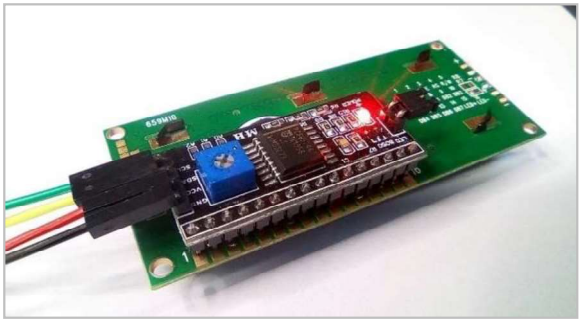
Como exemplo de código, vamos escrever o mesmo texto que aparece na montagem eletrônica, com a String "QI – INTERNET" na primeira linha (0) a partir da primeira coluna (0) e com o a String "OF THINGS" na segunda linha (1) a partir da sexta coluna (5).

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup(){
  lcd.begin(16, 2); //display do exemplo possui 2 linhas de 16 colunas cada
}

void loop(){
  //limpando o display
  lcd.clear();

  //posicionando e escrevendo a primeira parte da String
  lcd.setCursor(0, 0);   //coluna 0 (primeira) da linha 0 (primeira)
  lcd.print("QI - INTERNET");

  //posicionando e escrevendo segunda parte da String
  lcd.setCursor(5, 1);   //coluna 5 (sexta) da linha 1 (segunda)
  lcd.print("QI - INTERNET");

  delay(1000);
}
```

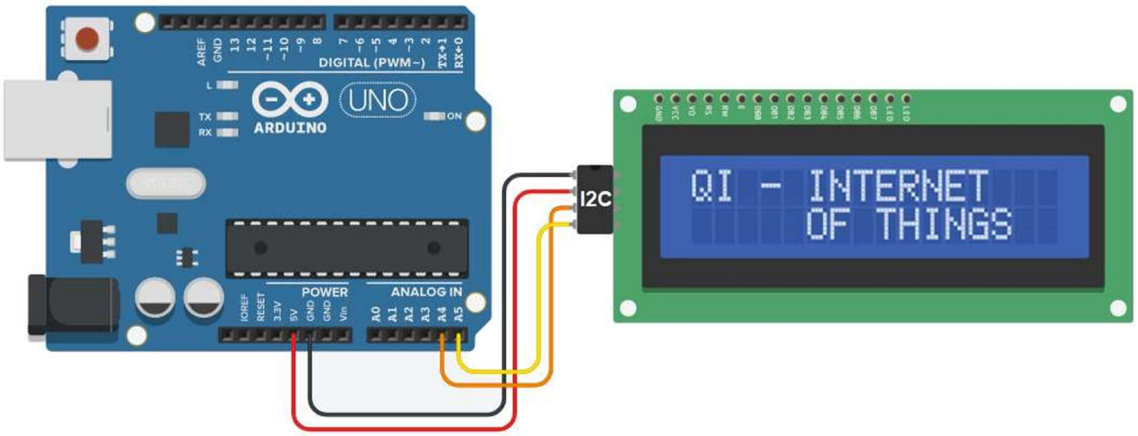
SEGUNDA FORMA DE USO: I2C

Na segunda forma de uso direta, podemos ligar o nosso display à um adaptador I2C para displays, permitindo o envio das informações através de uma comunicação Serial com endereçamento. Assim, utilizamos apenas a porta SDA e SCL do Arduino. Entretanto, este adaptador deve ser soldado às costas do display, cobrindo todos os seus 16 pinos:





Após a soldagem, ao invés de usarmos os pinos do display, utilizamos os pinos do adaptador I2C, como no esquema eletrônico a seguir:

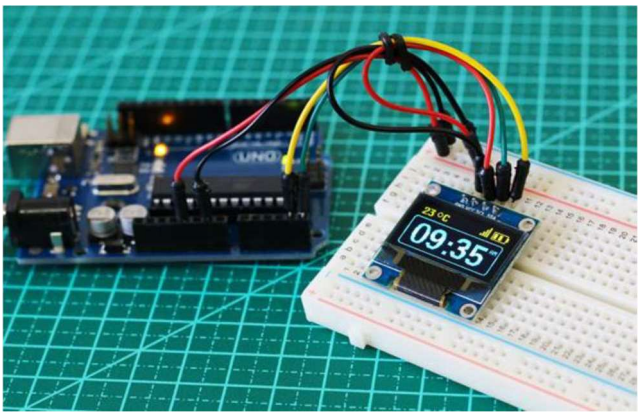


No caso de uma ligação deste tipo, precisamos da biblioteca LiquidCrystal_I2C, não nativa da Arduino IDE e cujo download pode ser feito neste link: https://bit.ly/arduinoqi. A sua instalação é análoga ao processo demonstrado no passo a passo do sensor ultrassônico. Além disso, a biblioteca nativa Wire também é fundamental pois ela é a responsável por controlar a comunicação I2C. Temos a disposição então um objeto do tipo LiquidCrystal_I2C que representa um display LCD com interface I2C e que quando inicializado, necessita de 3 parâmetros: endereço do display (0x27, caso não alterado mecanicamente através de solda nos pinos de endereçamento), total de colunas e total de linhas do display. Após instanciado, o objeto nos fornece exatamente os mesmos métodos da biblioteca LiquidCrystal do exemplo anterior. Assim, teríamos um código muito similar, como podemos ver abaixo:

```
#include <LiquidCrystal_I2C.h>
#include <Wire.h>

LiquidCrystal_I2C lcd(0x27, 16, 2);

void setup(){
  lcd.init(); //inicializa o display com comunicação I2C
  lcd.backlight(); //ativa a luz de fundo do display
}

void loop(){
  //limpando o display
  lcd.clear();

  //posicionando e escrevendo a primeira parte da String
  lcd.setCursor(0, 0);   //coluna 0 (primeira) da linha 0 (primeira)
  lcd.print("QI - INTERNET");

  //posicionando e escrevendo segunda parte da String
  lcd.setCursor(5, 1);   //coluna 5 (sexta) da linha 1 (segunda)
  lcd.print("OF THINGS");

  delay(1000);
}
```

Display OLED

Os displays OLED são uma evolução dos displays LCD. Apesar de também serem baseados no uso de leds iluminados individualmente, aqui temos componentes sólidos e orgânicos no lugar do cristal líquido da tecnologia anterior. Pela sua construção e modo de funcionamento, os displays OLED não necessitam de uma luz de fundo (backlight) e assim conseguem reproduzir o preto puro (algo inviável nos displays LCD). Mesmo os displays OLED mais simples (como os utilizados para prototipagem), possuem uma resolução mais alta, com pixels menores, otimização na geração de imagens e até a possibilidade de trabalhar com mais de uma cor (como azul e amarelo). Geralmente utilizam comunicação I2C (sendo necessárias, então, somente duas portas do Arduino) e contam com múltiplos modelos, cada um com suas bibliotecas e métodos próprios, dependentes de sua construção e dos recursos disponibilizados. O uso de displays OLED, por sua variedade e diferenças profundas no uso de cada modelo, foge do escopo de nosso curso. Porém, para uma ideia geral, é disponibilizado abaixo o link de um projeto desenvolvido por *Adilson Thomsen*, do site FelipeFlop, exemplificando o uso de uma das dezenas de modelos encontrados no mercado: o SSD1306, de resolução 128 x 64 e 3 cores.



https://www.filipeflop.com/blog/como-conectar-display-oled-arduino/