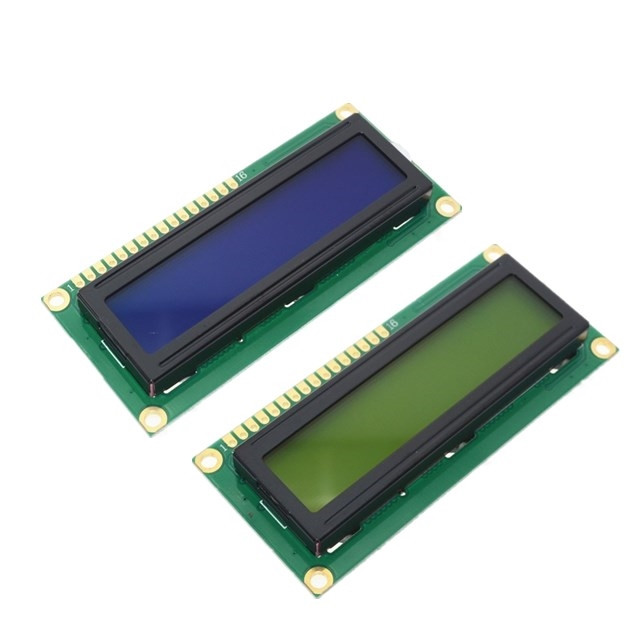
**What is LCD 16X2?**

An electronic device that is used to display data and the message is known as LCD 16×2. As the name suggests, it includes 16 Columns & 2 Rows so it can display 32 characters (16×2=32) in total & every character will be made with 5×8 (40) Pixel Dots. So, the total pixels within this LCD can be calculated as 32 x 40 otherwise 1280 pixels.



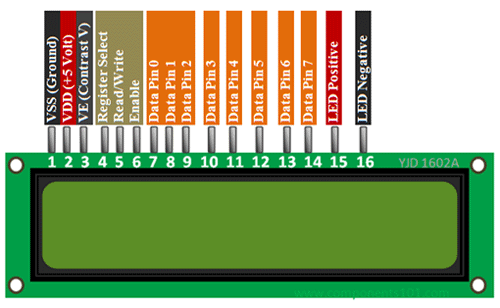
16 X2 displays mostly depend on multi-segment LEDs. There are different types of displays available in the market with different combinations such as 8×2, 8×1, 16×1, and 10×2, however, the LCD 16×2 is broadly used in devices, DIY circuits, electronic projects due to less cost, programmable friendly & simple to access.

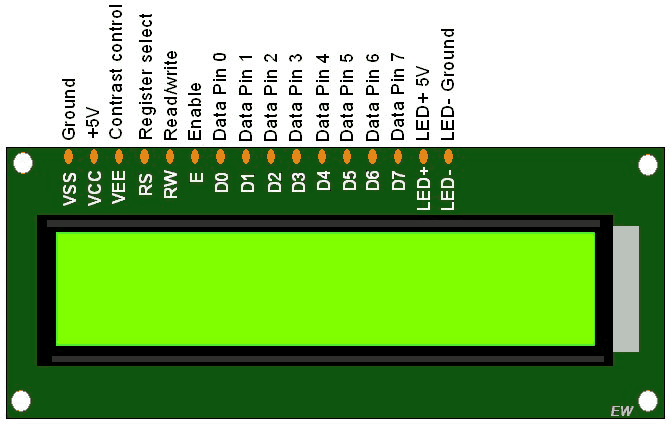
**Specifications of LCD 16X2**

The specifications of LCD 16X2 are discussed below.

* The operating voltage of this display ranges from 4.7V to 5.3V
* The display bezel is 72 x 25mm
* The operating current is 1mA without a backlight
* PCB size of the module is 80L x 36W x 10H mm
* HD47780 controller
* LED colour for backlight is green or blue
* Number of columns – 16
* Number of rows – 2
* Number of LCD pins – 16
* Characters – 32
* It works in 4-bit and 8-bit modes
* Pixel box of each character is 5×8 pixel
* Font size of character is 0.125Width x 0.200height

**LCD 16X2 Pin Configuration**





|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sr. No** | **Pin No.** | **Pin Name** | **Pin Type** | **Pin Description** | **Pin Connection** |
| 1 | Pin 1 | Ground | Source Pin | This is a ground pin of LCD | Connected to the ground of the MCU/ Power source |
| 2 | Pin 2 | VCC | Source Pin | This is the supply voltage pin of LCD | Connected to the supply pin of Power source |
| 3 | Pin 3 | V0/VEE | Control Pin | Adjusts the contrast of the LCD. | Connected to a variable POT that can source 0-5V |
| 4 | Pin 4 | Register Select | Control Pin | Toggles between Command/Data Register | Connected to a MCU pin and gets either 0 or 1.  0 -> Command Mode  1-> Data Mode |
| 5 | Pin 5 | Read/Write | Control Pin | Toggles the LCD between Read/Write Operation | Connected to a MCU pin and gets either 0 or 1.  0 -> Write Operation  1-> Read Operation |
| 6 | Pin 6 | Enable | Control Pin | Must be held high to perform Read/Write Operation | Connected to MCU and always held high. |
| 7 | Pin 7-14 | Data Bits (0-7) | Data/Command Pin | Pins used to send Command or data to the LCD. | In 4-Wire Mode  Only 4 pins (0-3) is connected to MCU  In 8-Wire Mode  All 8 pins (0-7) are connected to MCU |
| 8 | Pin 15 | LED Positive | LED Pin | Normal LED like operation to illuminate the LCD | Connected to +5V |
| 9 | Pin 16 | LED Negative | LED Pin | Normal LED like operation to illuminate the LCD connected with GND. | Connected to ground |

Registers of LCD

A 16×2 LCD has two registers like data register and command register. The RS (register select) is mainly used to change from one register to another. When the register set is ‘0’, then it is known as command register. Similarly, when the register set is ‘1’, then it is known as data register.

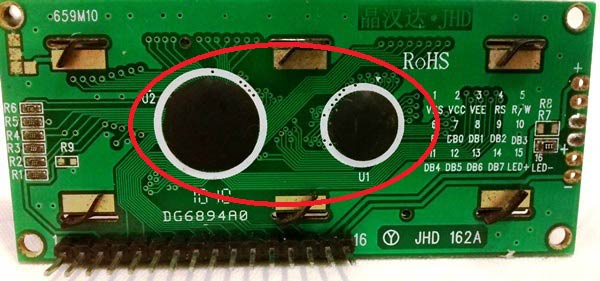
**Command Register**

The main function of the command register is to store the instructions of command which are given to the display. So that predefined tasks can be performed such as clearing the display, initializing, set the cursor place, and display control. Here commands processing can occur within the register.

**Data Register**

The main function of the data register is to store the information which is to be exhibited on the LCD screen. Here, the ASCII value of the character is the information which is to be exhibited on the screen of LCD. Whenever we send the information to LCD, it transmits to the data register, and then the process will be starting there. When register set =1, then the data register will be selected.

Back of 16X2 LCD Display



These black circles consist of an interface IC and its associated components to help us use this LCD with the MCU. Because our LCD is a 16\*2 Dot matrix LCD and so it will have (16\*2=32) 32 characters in total and each character will be made of 5\*8 Pixel Dots.  A Single character with all its Pixels enabled is shown in the below picture.

LCD-Module-Pixels

So now, we know that each character has (5\*8=40) 40 Pixels and for 32 Characters we will have (32\*40) 1280 Pixels. Further, the LCD should also be instructed about the Position of the Pixels.

It will be a hectic task to handle everything with the help of MCU, hence an **Interface IC like HD44780** is used, which is mounted on LCD Module itself. The function of this IC is to get the **Commands and Data** from the MCU and process them to display meaningful information onto our LCD Screen.

Let’s discuss the different type of mode and options available in our LCD that must be controlled by our Control Pins.

4-bit and 8-bit Mode of LCD:

The LCD can work in two different modes, namely the 4-bit mode and the 8-bit mode. In **4-bit mode**we send the data nibble by nibble, first upper nibble and then lower nibble. For those of you who don’t know what a nibble is: a nibble is a group of four bits, so the lower four bits (D0-D3) of a byte form the lower nibble while the upper four bits (D4-D7) of a byte form the higher nibble. This enables us to send 8-bit data.

Whereas **in 8-bit mode** we can send the 8-bit data directly in one stroke since we use all the 8 data lines.

Now you must have guessed it, yes 8-bit mode is faster and flawless than 4-bit mode. But the major drawback is that it needs 8 data lines connected to the microcontroller. This will make us run out of I/O pins on our MCU, so 4-bit mode is widely used. No control pins are used to set these modes. It's just the way of programming that change.

Read and Write Mode of LCD:

As said, the LCD itself consists of an Interface IC. The MCU can either read or write to this interface IC. Most of the times we will be just writing to the IC, since reading will make it more complex and such scenarios are very rare. Information like position of cursor, status completion interrupts etc. can be read if required, but it is out of the scope of this tutorial.

**COMMAND LINES**

Hex Code are mentioned after “for” for each command.

The commands of LCD 16X2 include the following: -

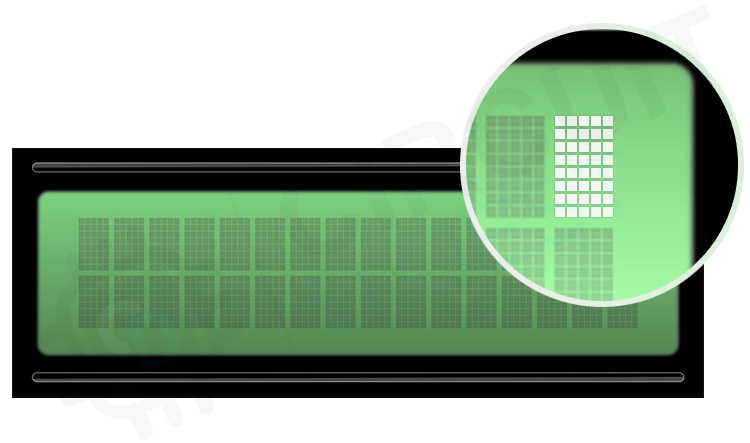
* For 0x01 (Hex Code), the LCD command will be the clear LCD screen
* For 0x02, the LCD command will be returning home
* For 0x04, the LCD command will be decrement cursor
* For 0x06, the LCD command will be Increment cursor
* For 0x05, the LCD command will be Shift display right
* For 0x07, the LCD command will be Shift display left
* For 0x08, the LCD command will be Display off, cursor off
* For 0x0A, the LCD command will be cursor on and display off
* For 0x0C, the LCD command will be cursor off, display on
* For 0x0E, the LCD command will be cursor blinking, Display on
* For 0x0F, the LCD command will be cursor blinking, Display on
* For 0x10, the LCD command will be Shift cursor position to left
* For 0x14, the LCD command will be Shift cursor position to the right
* For 0x18, the LCD command will be Shift the entire display to the left
* For 0x1C, the LCD command will be Shift the entire display to the right
* For 0x80, the LCD command will be Force cursor to the beginning (1st line)
* For 0xC0, the LCD command will be Force cursor to the beginning (2nd line)
* For 0x38, the LCD command will be 2 lines and 5×7 matrix
* For 0x38, the LCD command will cursor line 1 position 3
* For 0x3C, the LCD command will Activate second line
* For 0xC1, the LCD command will jump to second line, position 1
* For 0xC2, the LCD command will jump to second line, position 2

**INTERFACING LCD 16X2 WITH ARDUINO**

Since LCD uses HD44780 interfacing IC, Arduino had created LCD library which makes it easy to use with Arduino by including the header file and with some initialization.

Hardware Overview

True to their name, these LCDs are ideal for displaying only text/characters. A 16×2-character LCD, for example, has an LED backlight and can display 32 ASCII characters in two rows of 16 characters each.



If you look closely, you can see tiny rectangles for each character on the display and the pixels that make up a character. Each of these rectangles is a grid of 5×8 pixels.

They come in many sizes and colours: for example, 16×1, 16×4, 20×4, white text on a blue background or black text on a green and many others.

## Testing a Character LCD

Now we are on to the interesting stuff. Let’s test the LCD.

First, connect the 5V and GND pins from the Arduino to the breadboard power rail and plug your LCD into the breadboard.

Now we will power the LCD. The LCD has two separate power connections: One for the LCD (pin 1 and pin 2) and the other for the LCD backlight (pin 15 and pin 16). Connect pins 1 and 16 of the LCD to GND and 2 and 15 to 5V.

Most LCDs have a built-in series resistor for the LED backlight. You’ll find this near pin 15 on the back of the LCD. If your LCD does not include such a resistor or you are not sure if your LCD has one, you will need to add one between 5V and pin 15. It is safe to use a 220-ohm resistor, although a value this high may make the backlight a bit dim. For better results you can check the datasheet for maximum backlight current and select a suitable resistor value.

Next, we will make the connection for pin 3 on the LCD which controls the contrast and brightness of the display. To adjust the contrast, we will connect a 10K potentiometer between 5V and GND and connect the potentiometer’s centre pin (wiper) to pin 3 on the LCD.

Diagram

Description automatically generated with medium confidence

That’s it. Now turn on the Arduino. You will see the backlight lit up. Now as you turn the knob on the potentiometer, you will start to see the first row of rectangles. If that happens, Congratulations! Your LCD is working fine.

## Wiring a 16×2 Character LCD to an Arduino

We have already made the connections to power the LCD, now all we must do is make the necessary connections for communication.

We know that there are 8 data pins that carry data to the display. However, HD44780 based LCDs are designed in such a way that we can communicate with the LCD using only 4 data pins (4-bit mode) or 8 data pins (8-bit mode).

So, there will be 4 cases in which 2 case will be used in 4-bit mode and other 2 case will be of 8-bit mode. The difference between the two set of the cases is that in both case **R/W** pin will be connected to ground.

Case 1: **R/W** pin is connected to ground & LCD is being used in 4-bit mode

So, to interface the LCD using 4-bit mode we only need to connect 6 pins: RS, EN, D7, D6, D5, and D4.

Connect the LCD’s four data pins (D4-D7) to Arduino’s analog pins #A2-A5, the EN pin to analog pin # A1 and the RS pin to analog pin # A0.

Follow the diagram given below: -

Diagram, schematic

Description automatically generated

Case 2: **R/W** pin isn’t connected to ground & LCD is being used in 4-bit mode

So, to interface the LCD using 4-bit mode we only need to connect 7 pins: R/W, RS, EN, D7, D6, D5, and D4.

Connect the LCD’s four data pins (D4-D7) to Arduino’s digital pins #0-3, the EN pin to digital pin # 8, RS pin to digital pin # 9 & R/W pin to digital pin #10.

Follow the diagram given below: -

A picture containing graphical user interface

Description automatically generated

Case 3: **R/W** pin isn’t connected to ground & LCD is being used in 8-bit mode

So, to interface the LCD using 8-bit mode we only need to connect 11 pins: R/W, RS, EN, D0, D1, D2, D3, D7, D6, D4, D5, D6, D7.

Connect the LCD’s eight data pins (D0-D7) to Arduino’s digital pins #0-7, the EN pin to digital pin # 8, RS pin to digital pin # 9 & R/W pin to digital pin #10.

Follow the diagram given below: -

Diagram

Description automatically generated with medium confidence

Case 4: **R/W** pin is connected to ground & LCD is being used in 8-bit mode

So, to interface the LCD using 8-bit mode we only need to connect 10 pins: RS, EN, D0, D1, D2, D3, D7, D6, D4, D5, D6, D7.

Connect the LCD’s eight data pins (D0-D7) to Arduino’s digital pins #0-7, the EN pin to digital pin # 8 , RS pin to digital pin # 9.

Follow the diagram given below: -

Diagram

Description automatically generated

The code is very simple for printing text on LCD as there are already predefined variable in the header which can be used to display text on LCD.

Code as shown below:-

Case 1 :

#include <LiquidCrystal.h>

int rs=A0;

int en=A1;

int d4=A2;

int d5=A3;

int d6=A4;

int d7=A5;

LiquidCrystal lcd(rs,en,d4,d5,d6,d7); /\* For 4-bit mode \*/

void setup() {

lcd.begin(16,2); /\* Initialize 16x2 LCD \*/

lcd.clear(); /\* Clear the LCD \*/

}

void loop() {

lcd.setCursor(0,0); /\* Set cursor to column 0 row 0 \*/

lcd.print("LCD IS "); /\* Print data on display \*/

lcd.setCursor(0,1);

lcd.print(" WORKING !!!!");

}

Case 2 :

#include <LiquidCrystal.h> //header file

int rs=9;

int rw=10;

int en=8;

int d4=0;

int d5=1;

int d6=2;

int d7=3;

// create a LCD object. Parameter : (rs,rw,en,d4,d5,d6,d7)

LiquidCrystal lcd(rs,rw,en,d4,d5,d6,d7); /\* For 4-bit mode \*/

void setup() {

lcd.begin(16,2); /\* Initialize 16x2 LCD \*/

lcd.clear(); /\* Clear the LCD \*/

}

void loop() {

lcd.setCursor(0,0); /\* Set cursor to column 0 row 0 \*/

lcd.print("LCD IS "); /\* Print data on display \*/

lcd.setCursor(0,1);

lcd.print(" WORKING !!!!");

}

Case 3 :

#include <LiquidCrystal.h>

int rs=9;

int rw=10;

int en=8;

int d0=0;

int d1=1;

int d2=2;

int d3=3;

int d4=4;

int d5=5;

int d6=6;

int d7=7;

LiquidCrystal lcd(rs,rw,en,d0,d1,d2,d3,d4,d5,d6,d7); /\* For 8-bit mode \*/

void setup() {

lcd.begin(16,2); /\* Initialize 16x2 LCD \*/

lcd.clear(); /\* Clear the LCD \*/

}

void loop() {

lcd.setCursor(0,0); /\* Set cursor to column 0 row 0 \*/

lcd.print("LCD IS "); /\* Print data on display \*/

lcd.setCursor(0,1);

lcd.print(" WORKING !!!!");

}

Case 4 :

#include <LiquidCrystal.h>

int rs=9;

int en=8;

int d0=0;

int d1=1;

int d2=2;

int d3=3;

int d4=4;

int d5=5;

int d6=6;

int d7=7;

LiquidCrystal lcd(rs,en,d0,d1,d2,d3,d4,d5,d6,d7); /\* For 8-bit mode \*/

void setup() {

lcd.begin(16,2); /\* Initialize 16x2 LCD \*/

lcd.clear(); /\* Clear the LCD \*/

}

void loop() {

lcd.setCursor(0,0); /\* Set cursor to column 0 row 0 \*/

lcd.print("LCD IS "); /\* Print data on display \*/

lcd.setCursor(0,1);

lcd.print(" WORKING !!!!");

}

In the ‘setup’ we call two functions. The first function is **lcd. begin ()**. It is used to specify the dimensions (number of columns and rows) of the display. If you are using a 16×2-character LCD, pass the 16 and 2; If you’re using a 20×4 LCD, pass 20 and 4.

And second is **lcd. clear ().** This is used to clear the lcd screen and move the cursor to top left corner of the LCD.

**Print (“”)** function is used to print text on LCD.

// Print a message to the LCD.

lcd.print (" Hello world!");

After or before that we set the cursor position by calling the function **set. Cursor ().** The cursor position specifies the location where you want the new text to be displayed on the LCD. The upper left corner is assumed to be col=0, row=0.

// Set cursor where you want your next text

lcd. setCursor (0, 1);

lcd.print (" LCD Tutorial");

This will start printing text from the beginning of the second row.

* lcd.home () function is used to position the cursor in the upper left of the LCD without clearing the display.
* lcd.blink () function displays a blinking block of 5×8 pixels at the position at which the next character is to be written.
* lcd.cursor () displays an underscore (line) at the position at which the next character is to be written.
* lcd.noBlink () function turns off the blinking LCD cursor.
* lcd.noCursor () hides the LCD cursor.
* lcd.scrollDisplayRight () function scrolls the contents of the display one space to the right. If you want the text to scroll continuously, you must use this function inside a for loop.
* lcd.scrollDisplayLeft () function scrolls the contents of the display one space to the left. Like above function, use this inside a for loop for continuous scrolling.

## **Custom Character Generation for 16×2 Character LCD**

If you find the characters on the display dull and boring, you can create your own custom characters (glyphs) and symbols for your LCD. They are extremely useful when you want to display a character that is not part of the [standard ASCII character set](https://www.asciitable.com/).

A character is made up of a 5×8-pixel matrix, so you need to define your custom character within that matrix. You can use the [createChar()](https://www.arduino.cc/en/Reference/LiquidCrystalCreateChar) function to define a character.

To use createChar() you first set up an array of 8 bytes. Each byte in the array represents a row of characters in a 5×8 matrix. Whereas 0 and 1 in a byte indicate which pixel in the row should be ON and which should be OFF.

All these user defined characters are stored in the CGRAM of the LCD.

CGROM and CGRAM

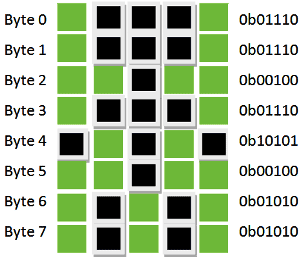
All Hitachi HD44780 controller-based LCDs have two types of memory – CGROM and CGRAM (Character Generator ROM and RAM).

CGROM is non-volatile memory and cannot be modified whereas CGRAM is volatile memory and can be modified at any time.

CGROM is used to store all permanent fonts that are displayed using their ASCII codes. For example, if we send 0x41 to the LCD, the letter ‘A’ will be printed on the display.

CGRAM is another memory used to store user defined characters. This RAM is limited to 64 bytes. For a 5×8 pixel-based LCD, only 8 user-defined characters can be stored in CGRAM. And for 5×10 pixel-based LCD only 4 user-defined characters can be stored.

The first step is to generate a pattern or the custom character. As we know each character is a combination of 5\*8 dots. We must select which dot (pixel) should go high and which should stay low. Simply draw a box like below and shade the regions based on your character. My character here is a stick man (hope it looks like one). Once shaded, simple write the equivalent binary value of each byte as shown below.



Simply put a ‘1’ on the shaded region and a ‘0’ on the un-shaded region for each byte, and that is it our custom pattern is ready. Similarly, I have made 8 custom pattern codes for our **8 memory spaces present it the CGROM**. They are listed in the table below.

|  |  |  |
| --- | --- | --- |
| **S.NO:** | **Custom Character** | **Pattern Code** |
| 1 | creating-binary-code-for-custom-character-1 | 0b01110, 0b01110, 0b00100, 0b01110, 0b10101, 0b00100, 0b01010, 0b01010 |
| 2 | creating-binary-code-for-custom-character-2 | 0b00000, 0b00000, 0b01010, 0b00100, 0b00100, 0b10001, 0b01110, 0b00000 |
| 3 | creating-binary-code-for-custom-character-3 | 0b00100, 0b01110, 0b11111, 0b11111, 0b01110, 0b01110, 0b01010, 0b01010 |
| 4 | creating-binary-code-for-custom-character-4 | 0b01110, 0b10001, 0b10001, 0b11111, 0b11011, 0b11011, 0b11111, 0b00000 |
| 5 | creating-binary-code-for-custom-character-5 | 0b01110, 0b10000, 0b10000, 0b11111, 0b11011, 0b11011, 0b11111, 0b00000 |
| 6 | creating-binary-code-for-custom-character-6 | 0b00000, 0b10001, 0b01010, 0b10001, 0b00100, 0b01110, 0b10001, 0b00000 |
| 7 | creating-binary-code-for-custom-character-7 | 0b00000, 0b00000, 0b01010, 0b10101, 0b10001, 0b01110, 0b00100, 0b00000 |
| 8 | creating-binary-code-for-custom-character-8 | 0b11111, 0b11111, 0b10101, 0b11011, 0b11011, 0b11111, 0b10001, 0b11111 |

Note: It is not mandatory to load all the 8 spaces provided in the CGRAM.

You can create your own pattern based upon examples shown.

Arduino Code shown below : -

// include the library code:

#include <LiquidCrystal.h>

int rs=9;

int en=10;

int d4=A0;

int d5=A1;

int d6=A2;

int d7=A3;

// initialize the library with the numbers of the interface pins

LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

// make some custom characters:

byte Heart[8] = {

0b00000,

0b01010,

0b11111,

0b11111,

0b01110,

0b00100,

0b00000,

0b00000

};

byte Bell[8] = {

0b00100,

0b01110,

0b01110,

0b01110,

0b11111,

0b00000,

0b00100,

0b00000

};

byte Alien[8] = {

0b11111,

0b10101,

0b11111,

0b11111,

0b01110,

0b01010,

0b11011,

0b00000

};

byte Check[8] = {

0b00000,

0b00001,

0b00011,

0b10110,

0b11100,

0b01000,

0b00000,

0b00000

};

byte Speaker[8] = {

0b00001,

0b00011,

0b01111,

0b01111,

0b01111,

0b00011,

0b00001,

0b00000

};

byte Sound[8] = {

0b00001,

0b00011,

0b00101,

0b01001,

0b01001,

0b01011,

0b11011,

0b11000

};

byte Skull[8] = {

0b00000,

0b01110,

0b10101,

0b11011,

0b01110,

0b01110,

0b00000,

0b00000

};

byte Lock[8] = {

0b01110,

0b10001,

0b10001,

0b11111,

0b11011,

0b11011,

0b11111,

0b00000

};

void setup()

{

// initialize LCD and set up the number of columns and rows:

lcd.begin(16, 2);

// create a new character

lcd.createChar(0, Heart);

// create a new character

lcd.createChar(1, Bell);

// create a new character

lcd.createChar(2, Alien);

// create a new character

lcd.createChar(3, Check);

// create a new character

lcd.createChar(4, Speaker);

// create a new character

lcd.createChar(5, Sound);

// create a new character

lcd.createChar(6, Skull);

// create a new character

lcd.createChar(7, Lock);

// Clears the LCD screen

lcd.clear();

// Print a message to the lcd.

lcd.print("Custom Character");

}

// Print All the custom characters

void loop()

{

lcd.setCursor(0, 1);

lcd.write(byte(0));

lcd.setCursor(2, 1);

lcd.write(byte(1));

lcd.setCursor(4, 1);

lcd.write(byte(2));

lcd.setCursor(6, 1);

lcd.write(byte(3));

lcd.setCursor(8, 1);

lcd.write(byte(4));

lcd.setCursor(10, 1);

lcd.write(byte(5));

lcd.setCursor(12, 1);

lcd.write(byte(6));

lcd.setCursor(14, 1);

lcd.write(byte(7));

}

### Code Explanation:

After including the library, we must define a custom character array of eight bytes like this:

byte Heart[8] = {

0b00000,

0b01010,

0b11111,

0b11111,

0b01110,

0b00100,

0b00000,

0b00000

};

In setup we need to create custom character using createChar() function. This function takes two parameters. The first parameter is a number between 0 and 7 to reserve one of the 8 supported custom characters. The second is the name of the array.

// create a new character

lcd.createChar(0, Heart);

Next in the loop, to display the custom character we simply use the write() function and pass it the number of the character we reserved earlier.

// byte(0) represents Heart character.

lcd.write(byte(0));