```
#include <Wire.h>
#include <Adafruit_RGBLCDShield.h>
#include <utility/Adafruit_MCP23017.h>
Adafruit_RGBLCDShield lcd = Adafruit_RGBLCDShield();

// Define variables for storing device information
const int MAX_DEVICES = 10;  // maximum number of devices that can be stored in the array
char devices[MAX_DEVICES][30];  // array of devices, each device can be up to 22 characters long
int num_devices = 0;  // variable to keep track of the number of devices currently stored
char status1[MAX_DEVICES][7];  // array to store the status of each device
char power1[MAX_DEVICES][7];  // array to store the power of each device
String the_devices[5] = {"PZW", "FDG", "LHT", "BDA", "HJL"};  // list of valid device types
char d_type[3];  // array to store the device type extracted from the user input
bool present = false;  // flag to check if the device type is present in the list of valid types
boolean sync = true;  // flag to keep track of whether the LCD display is in sync with the device
information in memory
int currentIndex = 0;  // variable to keep track of which device is currently selected on the LCD display
char device[30];  // array to store the name of the currently selected device
char symbol1 = ' ';  // character to represent the up arrow on the LCD display
char symbol2 = ' ';  // character to represent the down arrow on the LCD display

// Function to extract the device type from the user input
void Device_type(char input1[]) {
  d_type[0] = input1[2];
  d_type[1] = input1[3];
  d_type[2] = input1[4];

  present = false; // reset present flag for each call

  for (int i = 0; i < 5; i++) {
    if (strcmp(the_devices[i].c_str(), d_type) == 0) {  // check if the current device type matches one in
the list
```

```
      present = true;
      break;
    }
  }
}
// Function to check the amount of free memory
int freeRam() {
  extern int __heap_start, *__brkval;
  int free_memory;
  if ((int)__brkval == 0)
    free_memory = ((int)&free_memory) - ((int)&__heap_start);
  else
    free_memory = ((int)&free_memory) - ((int)__brkval);
  return free_memory;
}

// Define constants for LCD custom characters
byte downArrow[8] = {
  0b00100,
  0b01110,
  0b11111,
  0b00100,
  0b00100,
  0b00100,
  0b00100,
  0b00000
};

byte upArrow[8] = {
  0b00000,
  0b00100,
  0b00100,
  0b00100,
  0b00100,
  0b11111,
  0b01110,
  0b00100
};

// Define constants for FSM states
enum States {
  IDLE,
  TAKING_INPUT,
  STUDENT_ID,
};

States currentState = IDLE;

/*This function adds a new device to the system. The input parameter is a
 string that contains the device ID. The function checks if the ID is valid
  and not already added to the system. If the ID is valid and there is space
   to add a new device, the function adds the device to the system and
   updates the LCD display.*/
void adding(char input1[]){
  Device_type(input1);
  if (input1[1] != '-' || input1[7] != '-' || (input1[6] != 'S' && input1[6] != 'L' && input1[6] != 'O' &&
input1[6] != 'T' && input1[6] != 'C')|| input1[8] == '\0'|| present == false){
    if (num_devices == 0){
      lcd.setBacklight(7);
      lcd.clear();
      Serial.println("Error, not a valid device");
    } else if (num_devices > 0){
```

```
    selectDevice(0);

    Serial.println("Error, not a valid device");

   }

   return;

  } else if (num_devices < MAX_DEVICES) { // make sure we don't exceed the maximum number of
devices

   // Add device to the devices array

   strcpy(devices[num_devices], input1); // copy the input1 string to the next available slot in the
devices array

   memset(devices[num_devices] + strlen(input1), '\0', 22 - strlen(input1)); // fill the remaining
characters with null terminators


   // Set device status in the status1 array

   status1[num_devices][0] = input1[2]; // assign the first character of the input1 string to the first
element of the status1 array

   status1[num_devices][1] = input1[3]; // assign the second character of the input1 string to the
second element of the status1 array

   status1[num_devices][2] = input1[4]; // assign the third character of the input1 string to the third
element of the status1 array

   status1[num_devices][3] = '-';

   status1[num_devices][4] = 'O';

   status1[num_devices][5] = 'F';

   status1[num_devices][6] = 'F';

   lcd.setBacklight(3);


   power1[num_devices][0] = input1[2];

   power1[num_devices][1] = input1[3];

   power1[num_devices][2] = input1[4];

   power1[num_devices][3] = '-';

   power1[num_devices][4] = '0';

   power1[num_devices][5] = '0';

   power1[num_devices][6] = '0';
```

```
    num_devices++;

    Serial.print("Device added: ");
    Serial.println(input1);
    Serial.println(num_devices);
    printDevices();
    lcdprintdevice(input1);
  } else if (num_devices == MAX_DEVICES) {
    Serial.println("Error: Too many devices");
  }
  currentState = TAKING_INPUT;
}
/*This function updates the power value of a device. The input parameter is
 a string that contains the device ID and the new power value. The function
 searches for the device with the given ID and updates its power value if
 the ID is valid and the new power value is within the allowed range. The
 function also checks if the device is a temperature sensor and the new value
 is within the allowed temperature range. The function updates the LCD display
 with the new power value if the update is successful. */
void power(char input1[]) {
  bool found = false;
  if (strlen(input1) <= 9) {
    for (int i = 0; i < num_devices; i++) {
      if (power1[i][0] == input1[2] && power1[i][1] == input1[3] && power1[i][2] == input1[4]) {
        found = true;
        int powerValue = atoi(&input1[6]); // extract the power value from the input
        if (input1[2] == 'B' && input1[3] == 'D' && (powerValue < 9 || powerValue > 32)) {
          lcd.clear();
          Serial.print("Error, outside temperature range");
          break;
        } else if (input1[2] == 'H' && input1[3] == 'J'){
          lcd.clear();

          Serial.println("Error:  No feature for device");

          break;

        } else {

          sprintf(power1[i], "%c%c%c-%03d", input1[2], input1[3], input1[4], powerValue); // store the
power value in the power1 array

          lcdprintdevice(devices[currentIndex]);

          break;

        }

      }

    }

    if (!found) {

      Serial.println("Device not found");

    }

  } else {

    Serial.println("Error: ");

    Serial.println(input1);

    Serial.println("Too long");

  }

}

/*This function takes a device ID as input and changes its status to either

"ON" or "OFF" based on the command received. The status is updated in the

status1 array and the change is reflected on the LCD screen. */

void status(char input1[]){

  bool found = false;

  if (input1[6] == 'O'){

    for (int i = 0; i < num_devices; i++) {

      if (status1[i][0] == input1[2] && status1[i][1] == input1[3] && status1[i][2] == input1[4]) {

        // Found a match based on values in index 2, 3, and 4

        found = true;

        char searchValue[] = {input1[2], input1[3], input1[4]};
        if (input1[7] == 'N') {
          status1[i][4] = ' ';
          status1[i][5] = 'O';
          status1[i][6] = 'N';
          //lcd.setBacklight(2);
        } else {
          status1[i][4] = 'O';
          status1[i][5] = 'F';
          status1[i][6] = 'F';
          //lcd.setBacklight(3);
        }
        status1[i][0] = input1[2]; // assign the first character of the input1 string to the first element of
the status1 array
        status1[i][1] = input1[3]; // assign the second character of the input1 string to the second
element of the status1 array
        status1[i][2] = input1[4]; // assign the third character of the input1 string to the third element of
the status1 array
        status1[i][3] = '-';
      }
    }
    if (!found) {
      lcd.clear();
      lcd.print("Device not found");
    }
  } else if (input1[6] != 'O'){
    Serial.println("Error:");
    Serial.println(input1);
    Serial.println("not a valid option!");
  }
}
/*This function takes a device ID as input and removes the device from the
list of connected devices. The function shifts the remaining devices to fill
```

```
the gap left by the removed device and updates the number of devices in the system.*/
void remove(char input1[]){
 bool found = false;
 if (strlen(input1) <= 5) {
   for (int i = 0; i < num_devices; i++) {
     if (devices[i][2] == input1[2] && devices[i][3] == input1[3] && devices[i][4] == input1[4]) {
       found = true;
       char temp[30];
       memcpy(temp, devices[i], sizeof(devices[i])); // Store the device in a temporary variable
       for(int j = i+1; j < num_devices; j++){
         memcpy(devices[j-1], devices[j], sizeof(devices[j])); // Shift the elements to the left
       }
       memcpy(devices[num_devices - 1], temp, sizeof(temp)); // Move the stored device to the last
position
       for (int z = 0; z < sizeof(temp); z++){
         devices[num_devices - 1][z] = '\0';
       }
       char temp1[6];
       memcpy(temp1, status1[i], sizeof(status1[i]));
       for(int j = i+1; j < num_devices; j++){
         memcpy(status1[j-1], status1[j], sizeof(status1[j]));
       }
       memcpy(status1[num_devices - 1], temp1, sizeof(temp1)); // Move the stored device to the last
position
       for (int z = 0; z < sizeof(temp1); z++){
         status1[num_devices - 1][z] = '\0';
       }
       char temp2[6];
       memcpy(temp2, power1[i], sizeof(power1[i]));
       for(int j = i+1; j < num_devices; j++){
         memcpy(power1[j-1], power1[j], sizeof(power1[j]));
       }
```

```
    memcpy(power1[num_devices - 1], temp2, sizeof(temp2)); // Move the stored device to the last position

    for (int z = 0; z < sizeof(temp2); z++){

     power1[num_devices - 1][z] = '\0';

    }

    num_devices--; // Update the number of devices

    break;

   }

  }

  if (found) {

   Serial.println("Device removed successfully");

   for (int i = 0; i < num_devices +1; i++){

    Serial.println(devices[i]);

   }

  } else {

   Serial.println("Device not found");

  }

 } else {

  Serial.println("Error: ");

  Serial.println(input1);

  Serial.println("Too long");

 }

}

/*This function reads input from the serial monitor and processes it based on

the command received. The input is converted to uppercase before processing

to ensure consistency. The function calls the appropriate function based on the command received.*/

void readinput(){

 if (Serial.available() > 0){

  char input[23]; // increase the size of the input array to 23 characters to allow space for the null terminator
  int numChars = Serial.readBytesUntil('\n', input, 22); // read up to 22 characters from the serial
input
  if (numChars == 22) { // check if 22 characters were read
    char nextChar = Serial.read(); // read the next character from the serial input
    if (nextChar != '\n') { // check if the next character is not a newline character
      Serial.println("Error: Input too long");
      return;
    }
  }
  input[numChars] = '\0'; // add null terminator to the end of the input array
  char input1[23];
  strcpy(input1, input);
  for (int i = 0; i < strlen(input1); i++) {
    input1[i] = toupper(input1[i]); // Convert each character to uppercase
  }
  if (input1[0] == 'A'){
    adding(input1);
  } else if (input1[0] == 'S'){
    status(input1);
  } else if (input1[0] == 'P'){
    int p_len = sizeof(input1);
    if (p_len >= 6){
      power(input1);
    } else {
      lcd.clear();
      Serial.print("Error: ");
      Serial.print(input1);
      Serial.println(" ;no value passed");
    }
  } else if (input1[0] == 'R'){
      remove(input1);
    } else {
      Serial.println("Error: ");
      Serial.println(input1);
    }
  }
}
/*This function prints the list of devices currently stored in the devices
array to the Serial Monitor.*/
void printDevices() {
  Serial.println("Devices:");
  for (int i = 0; i < num_devices; i++) {
    Serial.println(devices[i]);
  }
}
/*This function is called when the up arrow key is pressed on the LCD.
It checks if the currently selected device is the first device in the list,
and sets the value of symbol1 accordingly.*/
void up(){
  if (currentIndex == 0) {
    symbol1 = ' ';
  } else if (currentIndex != 0) {
    symbol1 = byte(0);
  }
}
/*This function is called when the down arrow key is pressed on the LCD.
It checks if the currently selected device is the last device in the list,
and sets the value of symbol2 accordingly.*/
void down(){
  if (currentIndex + 1 == num_devices){
    symbol2 = ' ';
  } else {
    symbol2 = byte(1);
  }
}
/*This function prints the details of the currently selected device to
the LCD screen. It displays the device name, device type, status and power values.
It also sets the value of symbol1 and symbol2 based on the position of the currently selected device in the list.*/
void lcdprintdevice(char device[]){
  lcd.clear();
  lcdcolor();
  up();
  lcd.print(symbol1);
  lcd.setCursor(1,0);
  for(int i = 2; i <= 4; i++){
    lcd.print(device[i]);
  }
  lcd.setCursor(5, 0);
  for(int i = 8; i <= 22; i++){
    if (i < strlen(device)) {
      lcd.print(device[i]);
    } else {
      lcd.print(" ");
    }
  }
  lcd.setCursor(0, 1);      //device type
  down();
  lcd.print(symbol2);
  lcd.print(device[6]);

  lcd.setCursor(3, 1);       //status
  for (int i = 4; i < 7; i++) {
    lcd.print(status1[currentIndex][i]);
  }


  lcd.setCursor(7, 1);      //power
  for (int i = 4; i < 7; i++) {
    lcd.print(power1[currentIndex][i]);
  }
  if (device[6] == 'S' || device[6] == 'L'){
    lcd.setCursor(10, 1);
    lcd.print("%");
  }
  currentState = TAKING_INPUT;
}
/*This function sets the color of the LCD backlight based on the status of the device
currently selected. If the status of the device is "ON", the backlight color is
set to green. If the status of the device is "OFF", the backlight color is set to red.
For all other cases, the backlight color is set to white.*/
void lcdcolor() {
  if (status1[currentIndex][5] == 'O') { //&& status1[currentIndex][6] == 'N') { // If device status is ON
    lcd.setBacklight(2); // Set backlight color to green
  } else if (status1[currentIndex][4] == 'O' && status1[currentIndex][5] == 'F') { // &&
status1[currentIndex][6] == 'F') { // If device status is OFF
    lcd.setBacklight(3); // Set backlight color to red
  } else {
    lcd.setBacklight(7); // Set backlight color to white for all other cases
  }
}
/*This function selects the device at the given index in the devices array and copies
the device string to the device array. It then calls the lcdprintdevice() function to
display the device information on the LCD.*/
void selectDevice(int index) {
  if (index >= 0 && index < num_devices + 1) { // make sure the index is within range
    currentIndex = index;
    strcpy(device, devices[index]); // copy the device string to the device array
    lcdprintdevice(device);
  } else {
    lcd.clear();
    lcd.print("Invalid index");
  }
}
/*This function displays a message on the LCD that includes the user's
ID and the amount of free RAM available. It sets the backlight color to purple.*/
void S_ID(){
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("F225694");
  lcd.setCursor(0, 1);
  lcd.print("Free Ram: ");
  lcd.print(freeRam());
  lcd.print(" bytes!");
  lcd.setBacklight(5);
}
/*This function reads the state of the four buttons on the LCD and performs the appropriate action
based on the button pressed.
If the left button is pressed, it selects the first device in the devices array. If the up button is pressed,
it selects the device before the currently selected device. If the down button is pressed,
it selects the device after the currently selected device. If the select button is pressed,
it calls the S_ID() function to display the user's ID and free RAM.*/
void readButtons(){
  uint8_t buttons = lcd.readButtons();


  if (buttons & BUTTON_LEFT){
    if (num_devices != 0) {
      delay(500);
      selectDevice(0);
      lcdcolor();
    } else {
      lcd.setCursor(0, 0);
      lcd.print("Invalid index");
    }
  } else if (buttons & BUTTON_UP) {
    // select device before current device in array
    delay(500);  // Wait for 500 milliseconds to debounce the button
    int nextIndex = (currentIndex - 1);
    selectDevice(nextIndex);
    lcdcolor();
    //currentIndex--;
  } else if (buttons & BUTTON_DOWN) {
    // select the device next in the array
    delay(500);  // Wait for 500 milliseconds to debounce the button
    int nextIndex = (currentIndex + 1);
    selectDevice(nextIndex);
    lcdcolor();
    //currentIndex++;
  } else if (buttons & BUTTON_SELECT) {
    S_ID();
    while (lcd.readButtons() & BUTTON_SELECT) {}
    if (num_devices == 0){
      lcd.setBacklight(7);
      lcd.clear();
    } else if (num_devices > 0){
      selectDevice(0);
    }
  }
}
/*This function handles the synchronization process between the LCD display and the serial monitor.
If the sync variable is true, the function sends the character "Q" to the LCD display and serial
monitor,
waits for one second, and checks if there is any data available on the serial monitor. If the received
data is "X",
the synchronization process is marked as complete by setting sync to false, and the function prints a
message to
the serial monitor. If the received data is a newline or carriage return, nothing happens. Otherwise,
an error
message is printed to the serial monitor, and the currentState variable is set to IDLE.*/
void synchronisation(){
  if (sync) { // if synchronisation is not complete
    lcd.print("Q"); // send Q to lcd
    Serial.print('Q');
    delay(1000); // wait for one second
    lcd.clear();
    lcd.print("");
    delay(1000);
    if (Serial.available() > 0) { // check if there is any data
      char input = Serial.read(); // read a character from serial
      if (input == 'X') { // if X is received
        sync = false; // set synchronisation to false
        Serial.println("UDCHARS,FREERAM"); // print a message
        lcd.setBacklight(7);
        currentState = TAKING_INPUT;
        while (Serial.available() > 0) { // Consume any remaining newline or carriage return characters
          Serial.read();
        }
      }
      else if (input == '\n' || input == '\r') { // if newline or carriage return is received
```

```
       //do Nothing
      }
      else { // For any other invalid input
       Serial.print("Error: ");
       Serial.println(input); // print an error message with the invalid input
       currentState = IDLE;
      }
    }
  }
}

void setup() {
  lcd.begin(16,2);
  Serial.begin(9600);
  lcd.setBacklight(5);

  //defining the custom characyers so i can print them to the display
  lcd.createChar(0, downArrow);
  lcd.createChar(1, upArrow);
}

/*This function is called repeatedly during the program execution. It checks the value of currentState
and executes
the appropriate case. In the IDLE case, it calls the synchronization() function. In the TAKING_INPUT
and STUDENT_ID cases,
it calls the readButtons() and readinput() functions.*/
void loop() {
  switch (currentState) {
    case IDLE:   //waits for input from user to synchronisation, synchronisation must happen before
the user adds a device, this case reads the input and then passes it into the next case
      synchronisation();
      break;
    case TAKING_INPUT: //must take input from user about adding, status, power or remove, also
button presses happen here.
      readButtons();
      readinput();
      break;
    case STUDENT_ID:
      S_ID();
      readButtons();
      readinput();
      break;
  }
}
```