MENU LIBRARY Version 2

TUTORIAL

WHAT IS THE MENU LIBRARY?

The Menu Library is a simple way to add a menu to your Sketches. Menus are useful if you want the person using your sketch (the user) to choose what **actions** needs to be done, and in what order.

Menu	→ Make action →	Actions
	← Return to menu ←	

HOW DOES IT WORK?

To use a menu you need three things:

- A way to display the menu (LCD, LED, the Serial Monitor, …);
  The Menu Library can handle displaying the menu if you use one of the following LCD Libraries:
  <LiquidCrystal.h> (That comes with the Arduino IDE)
  <LiquidTWI.h> (for a LCD with an i^{2}C backpack –Twin Wire Interface-)
  Whether the display is controlled by the sketch or the Menu Library, the sketch can always use it when the menu is not in Menu Display mode.
- A way to navigate the menu (usually a keypad);
  The Menu Library allows you to use four switches connected to four digital pins on the Arduino, to use four switches connected to an analog pin of the Arduino, or to use your own keypad. In this last case, it will be your responsibility to tell the Menu Library what key has been pressed. In all cases, your sketch will be able to read the keys at any time.
- A system that can help you to navigate the menu (Menu Library).

Basically, you will setup your display device, setup your keypad, and setup the menu.

SETTING UP YOUR DISPLAY DEVICE.

Please refer to the manufacturer of your device to learn how to do it properly. You should always run one of the example Sketches (Hello World) that comes with the LCD Library before attempting to use the Menu Library.

Example : the <LiquidCrystal.h> Library:

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11,      5,    4,    3,    2);
//               (RS, Enable, D4,   D5,   D6,   D7)
int lcdNumCols = 16;
int lcdNumLines = 2;
```

The Menu Library can display the menu if you use one of the following LCD Libraries:
<LiquidCrystal.h>
<LiquidTWI.h>

To do this, in the setup() section of your sketch, add the following line :

```
menu.handleLcd(&myLcd, lcdNumCols, lcdNumLines);
(Don't forget the ampersand "&" before the lcd's name.)
```

This just tells the Menu Library where to find your LCD. You still have full control of your LCD.

If your display device is not one of the two mentioned above or if you decide to print yourself the menu on your device, you will also need a routine that displays the menu. This routine should be placed **before** the setup() section of your sketch.

This is what it could look like:

```
#include <YourLcdLibrary.h>
YourLcdLibrary lcd(arg1, arg2, …, argn);
int lcdNumCols = 16;
int lcdNumLines = 2;

void showMenu() {
  if (menu.needsUpdate()) {
    lcd.clear();
    for (int i = 0 ; i < lcdNumLines ; i++) {
      lcd.setCursor(0,i);
      lcd.print(menu.lcdLine(i));
    }
    delay(100);
    menu.updated();
  }
}
```

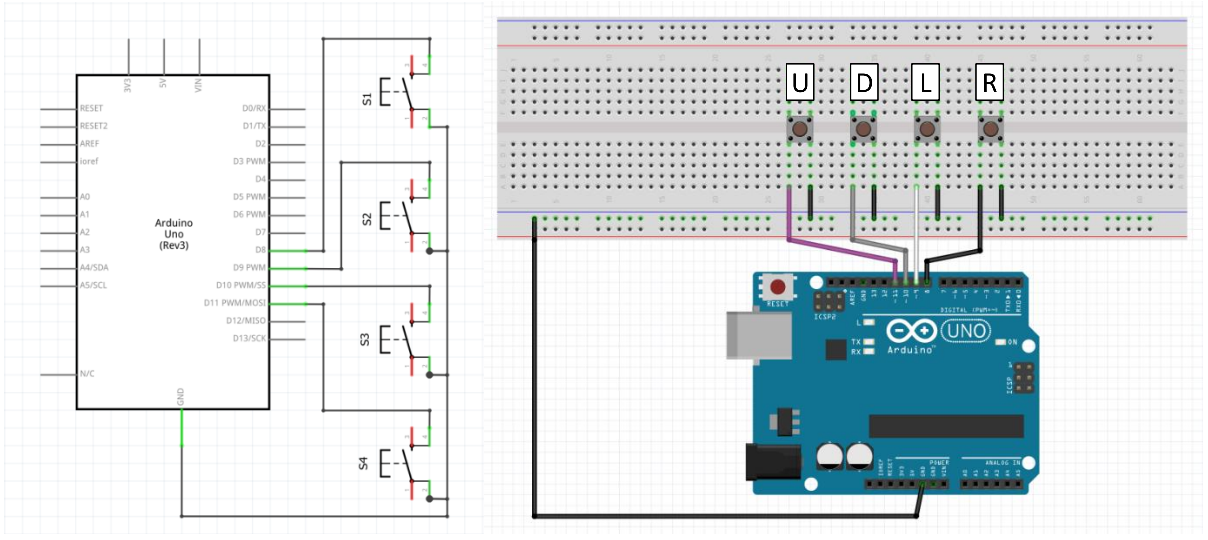
In the setup() section of the sketch, if needed:

```
lcd.begin(lcdNumCols, lcdNumLines);
```

SETTING YOUR KEYPAD.

If you want to let the Menu Library handle the keypad, you have two options. Connect four switches to four digital pins, or connect four switches to an analog pin on the Arduino.

For the first option, connect the switches in the following manner:



In this example,

- the Up key is attached to pin 11 of the Arduino
- the Down key is attached to pin 10 of the Arduino
- the Left key is attached to pin 9 of the Arduino
- the Right key is attached to pin 8 of the Arduino

In the setup() section of the sketch, add the following line :

```
menu.handleSwitches(11,  10,     9,      8  );
//                  (Up, Down, Left, Right);   In that order…
```

You will be able to use the keypad in the parts of the sketch when the menu is not displayed (During the **action** parts of your sketch).

Before the setup() section of your sketch, prepare a variable to store the value of the last key pressed :

```
int myKey = 0;
```

The following methods will be available to read the keypad:

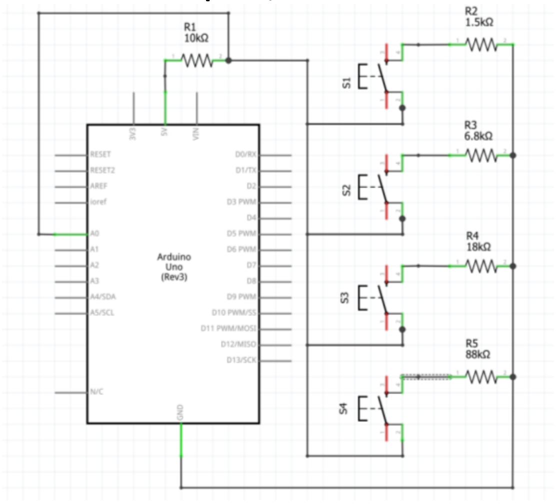
```
myKey = menu.readKey();
myKey = menu.readKeyWithRepeat(500, 10); //See the specs for this below
```

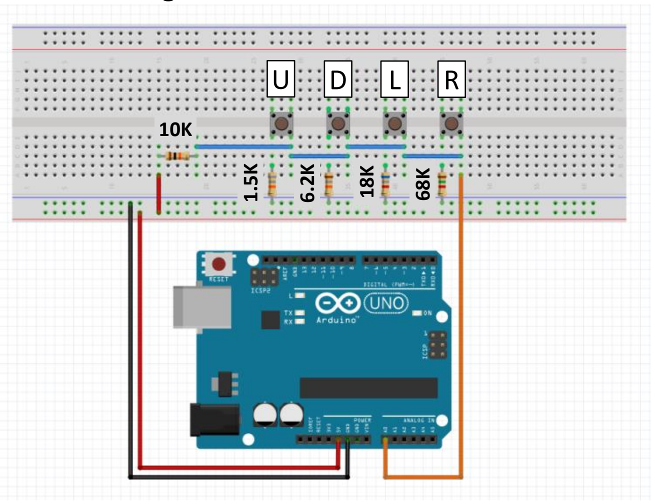
Both methods will return: `UP = 1; DOWN = 2; LEFT = 3; RIGHT = 4; No key pressed = 0.`

The following methods can be used to find out if a key is pressed:

```
if menu.isPressed(myKey) {…;}   //true if myKey is currently pressed
if menu.wasPressed(myKey) {…;}  //true if myKey is pressed (waits for the release of the key)
if menu.keyPressed() {…;}       //true if any key is currently pressed
```

For the second option, connect the switches in the following manner:





In this example:

- The pullup resistor is 10KΩ
- The resistor for the Up key is 1.5KΩ ideally, or between(1kΩ and 3kΩ)
- The resistor for the Down key is 6.2KΩ ideally, or between(5kΩ and 9kΩ)
- The resistor for the Left key is 18KΩ ideally, or between(15kΩ and 27kΩ)
- The resistor for the Right key is 68KΩ ideally, or between(56kΩ and 100kΩ)

Other values for the resistors can be used depending on the value of the pullup resistor, by simply using cross multiplication (rule of 3). If the pullup resistor was 1KΩ, The values of the four other resistors would be: 150Ω, 620Ω, 1.8KΩ and 6.8KΩ.

In the setup() section of the sketch, add the following line:

```
menu.handleSwitches(A0);  //Where A0 is the analog pin on which the switches are connected
```

You will be able to use the keypad in the parts of the sketch when the menu is not displayed (During the **action** parts of your sketch).

Before the setup() section of your sketch, prepare a variable to store the value of the last key pressed:

```
int myKey = 0;
```

The following methods will be available to read the keypad:

```
myKey = menu.readKey();
myKey = menu.readKeyWithRepeat(500, 10); //See the specs below for this method.
```

Both methods will return: `UP = 1; DOWN = 2; LEFT = 3; RIGHT = 4; No key pressed = 0.`

The following methods can be used to find out if a key is pressed:

```
if menu.isPressed(myKey) {…;}   //true if myKey is currently pressed
if menu.wasPressed(myKey) {…;}  //true if myKey is pressed (waits for the release of the key)
if menu.keyPressed() {…;}       //true if any key is currently pressed
```

If you want to use another keypad:
Please refer to the manufacturer of your device to learn how to do it properly. You should always run one of the example Sketches that comes with the Keypad Library before attempting to use the Menu Library.

Example : (Here, with a keypad found at Adafruit™)
```
#include <Keypad.h>
const byte ROWS = 4; //four rows
const byte COLS = 3; //three columns
char keys[ROWS][COLS] = {
   {'1','2','3'},
   {'4','5','6'},
   {'7','8','9'},
   {'#','0','*'}
};
byte rowPins[ROWS] = {6, 7, 8, 9}; //connect to the row pinouts of the keypad
byte colPins[COLS] = {10, 11, 12}; //connect to the column pinouts of the keypad

Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);
```

You will also need to define what directional keys will be sent to the Menu Library {Before the setup() section of your sketch}:

- You can use characters: (as with the above keypad)
  ```
  #define UP '2'
  #define DOWN '8'
  #define LEFT '4'
  #define RIGHT '6'
  ```
- Or you can use integers: (for another keypad Library that returns integers)
  ```
  #define UP 1
  #define DOWN 2
  #define LEFT 3
  #define RIGHT 4
  ```

In the setup() section of your sketch, tell the Menu Library what are the actual values that are going to be sent by the four keys of your keypad. If you used #define as above, insert this line:
```
menu.mapKeys(UP, DOWN, LEFT, RIGHT)  //In that order
```

You will also be able to use UP, DOWN, LEFT and RIGHT keys when carrying the actions that the user has selected. Just use the methods provided by your Keypad Library.

For the <Keypad.h> library, you read keys in the following manner:
```
myKey = keypad.getKey();
```

To use your keypad to move in the menu, you have to tell the Menu Library which key was pressed. In the loop() section of your sketch, if you used #define as above, you would do something like this:
```
Int action = 0;
char myKey = keypad.getKey();
if (myKey == UP or myKey == DOWN or myKey == LEFT or myKey == RIGHT) action = menu.update(myKey);
```

HOW DO I NAVIGATE THE MENU?

The directional keys are used to navigate the menu.

UP	Will select the item before the current item if it exists.
DOWN	Will select the item after the current item if it exists.
LEFT	Will select the parent of the item if it exists.
RIGHT	Will behave differently if the menu item has a submenu or not. If it has a submenu, it will select the first item of the submenu. Otherwise, the Menu Library will return an integer to say what action the user wants you to take. (No submenu means **action!**)

SETTING YOUR MENU

Describing your menu is very easy. You construct a String with the items you want in you menu, one line per item.

Example :

```
const String menuItems =
"-READ:000"
"--SENSORS:000"
"---SENSOR A1:101"
"---SENSOR A2:102"
"--SWITCHES:000"
"---SWITCH PIN 4:103"
"---SWITCH PIN 5:104"
"-SET:000"
"--SERVO ARM:105"
"--SERVO BASE:106"
"-MOVE SERVOS:107";


Menu menu(menuItems); //Set up menu
```

The first line says that it is a constant (won't change) of type String (characters), that its name is "menuItems" and that it is equal to the lines that are below. (Up to the semicolon ";")

Each line holds one menu item between quotes (") and contains:

- Dashes to specify the level of the item;
- The label of the item to be displayed on the LCD;
- A colon ":";
- A number between "000" and "999" to identify an **action** to be performed.
  (It has to be exactly 3 characters long and only be digits.)
  (Use "000" to say: "I have a submenu.")

Don't forget to place a semi-colon ";" at the end of the last line.

Finally, create an instance of the menu.

That's it, you're done!

After this, the menu library knows that:

The MAIN MENU is:

```
"-READ:000"
"-SET:000"
"-MOVE SERVOS:107"
```

READ has a submenu:

```
"--SENSORS:000"
"--SWITCHES:000"
```

SENSORS has a submenu:

```
"---SENSOR A1:101"
"---SENSOR A2:102"
```

SWITCHES has a submenu:

```
"---SWITCH PIN 4:103"
"---SWITCH PIN 5:104"
```

SET has a submenu:

```
"--SERVO ARM:105"
"--SERVO BASE:106"
```

The menu items that have submenus are (numbered 000):

```
"-READ:000"
"--SENSORS:000"
"--SWITCHES:000"
"-SET:000"
```

The menu items that spark actions are (numbered 101 to 107):

```
"---SENSOR A1:101"
"---SENSOR A2:102"
"---SWITCH PIN 4:103"
"---SWITCH PIN 5:104"
"--SERVO ARM:105"
"--SERVO BASE:106"
"-MOVE SERVOS:107"
```

To have your Sketch do the actions selected by the user, you will use a switch:

```
void make(int action) {
  switch (action) {
    case 101: { readPin(A1, ANALOG); break; }
    case 102: { readPin(A2, ANALOG); break; }
    case 103: { readPin(4, DIGITAL); break; }
    case 104: { readPin(5, DIGITAL); break; }
    case 105: { angleServoArm = changeValue(angleServoArm); break; }
    case 106: { angleServoBase = changeValue(angleServoBase); break; }
    case 107: { servoArm.write(angleServoArm); servoBase.write(angleServoBase); break; }
  }
}
```

DO I HAVE TO DO SOMETHING SPECIAL IN THE SETUP() SECTION OF THE SKETCH?

Depending on the options that you decide to use, the setup() section will be different:

Case 1: You use your own LCD and your own keypad:

```
lcd.begin(lcdNumCols, lcdNumLines);       //The setup needed for your display device
menu.defineLcd(lcdNumCols, lcdNumLines);  //Advise the Menu Library of the size of your device
menu.mapKey(UP, DOWN, LEFT, RIGHT);       //Advise the Menu Library of the keys you will send
```

Case 2: You let the Menu library display itself on the LCD and you control your own keypad:

```
lcd.begin(lcdNumCols, lcdNumLines);             //The setup needed for your display device
menu.handleLcd(&lcd, lcdNumCols, lcdNumLines); //Pass the LCD to the Menu Library
menu.mapKey(UP, DOWN, LEFT, RIGHT);             //Advise the Menu Library of the keys you will send
```

Case 3: You let the Menu library display itself on the LCD and control the keypad:

```
lcd.begin(lcdNumCols, lcdNumLines);             //The setup needed for your display device
menu.handleLcd(&lcd, lcdNumCols, lcdNumLines); //Pass the LCD to the Menu Library
menu.handleSwitches(3, 4, 5, 6);                //Pass the pins that you use for the four switches
or
menu.handleSwitches(A0);                        //Pass the analog pin for the switches
```

DO I HAVE TO DO SOMETHING SPECIAL IN THE LOOP() PART OF THE SKETCH?

Again, depending on the options that you decide to use, the loop() section will be different

Case 1: You use your own LCD and your own keypad:

```
void loop() {
  int action = 0;
  showMenu();                       //Update the LCD with the menu. See page 11
  char myKey = keypad.getKey();
  if (myKey == UP or myKey == DOWN or myKey == LEFT or myKey == RIGHT)
    action = menu.update(myKey);
  if (action > 0) {                 //If we need to act:
    make(action);                     //make it.
    menu.done();                      //We are done with this action... Go back to the menu.
  }
}
```

Case 2: You let the Menu library display itself on the LCD and you control your own keypad:

```
void loop() {
  int action = 0;
  menu.showMenu();                  //Update the LCD with the menu.
  char myKey = keypad.getKey();
  if (myKey == UP or myKey == DOWN or myKey == LEFT or myKey == RIGHT)
    action = menu.update(myKey);
  if (action > 0) {                 //If we need to act:
    make(action);                     //make it.
    menu.done();                      //We are done with this action... Go back to the menu.
  }
}
```

Case 3: You let the Menu library display itself on the LCD and control the keypad:

```
void loop() {
  menu.showMenu();                  //Menu Library updates the LCD with the menu.
  int action = menu.update();       //Read the action tagged to the (new) current menu item.
  if (action > 0) {                 //If we need to act:
    make(action);                     //make it.
    menu.done();                      //We are done with this action... Go back to the menu.
  }
}
```

WHAT ARE THE METHODS THAT THE MENU LIBRARY PROVIDES?

Mandatory methods

Those methods have to be called to use the Menu Library.

IN ALL CASES

void Menu::menu(String menuItems) (MANDATORY)

```
Menu menu(menuItems);
```

The first method is called a constructor, which creates the menu. It has to be placed **before** the setup() section of your sketch.
Menu is the name of the constructor;
menu is the name that you will use to refer to the Menu Library throughout your sketch;
menuItems is the name that you gave to the String containing your menu.

YOU CONTROL YOUR LCD

void Menu::defineLcd(int columns, int rows) (MANDATORY)

```
menu.defineLcd(20, 4);
```

This method has to be placed in the setup() section of your sketch. It tells the Menu Library how many columns and how many lines your LCD has.
columns is the number of columns your LCD has;
rows is the number of rows your LCD has;

THE MENU LIBRARY CONTROLS THE LCD

void Menu::handleLcd(LiquidTWI *Lcd, int columns, int rows) (MANDATORY)

```
menu.handleLcd(&lcd, 20, 4);
```

This method has to be placed in the setup() section of your sketch. It tells the Menu Library where to find your LCD, and how many columns and how many lines your LCD has.
&lcd is the address of your Lcd. You **have** to precede the name with the ampersand (&)
columns is the number of columns your LCD has;
rows is the number of rows your LCD has;

YOU CONTROL YOUR KEYPAD

void Menu::mapKey(char UP, char DOWN, char LEFT, char RIGHT) (MANDATORY)
void Menu::mapKey(int UP, int DOWN, int LEFT, int RIGHT) (MANDATORY) or…

```
menu.mapKey(UP, DOWN, LEFT, RIGHT); // If you used #define or…
menu.mapKey("2", "8", "4", "6"); //or…
menu.mapKey(1, 2, 3, 4);
```

This method has to be placed in the setup(). It tells the Menu Library what are the characters or integers that you will send to the library if one of the directional keys is pressed. They have to be given in that exact order: UP, DOWN, LEFT and RIGHT.

THE MENU LIBRARY CONTROLS THE KEYPAD

void Menu::handleSwitches(int pinUP, int pinDOWN, int pinLEFT, int pinRIGHT) (MANDATORY) or…
void Menu:: handleSwitches(int pinAnalog) (MANDATORY)

```
menu.handleSwitches(2, 3, 4, 5); //If you use four digital pins or…
menu.mapKey(A0);                 //If you use one analog pin
```

This method has to be placed in the setup(). It tells the Menu Library **the pins** where the switches are attached. For digital pins, they have to be given in that exact order: UP, DOWN, LEFT and RIGHT.

LCD managing methods (Needed only if you are handling your display device)

Those methods will allow you to print the menu to your LCD

String Menu::lcdLine(int line)

```
menu.lcdLine(0);
```

You will use this method to display (update) the menu on your LCD. It returns the label of the menu item that you have to print on a specific line of the LCD. Typically, you would call this method for each line that you have on the LCD:

```
lcd.clear();
for (int i = 0 ; i < lcdNumLines ; i++) {
  lcd.setCursor(0,i);
  lcd.print(menu.lcdLine(i));
}
```

The label of the menu item is preceded by a caret ">" if it is the current item. It is preceded by a space otherwise.

bool Menu::needsUpdate()

```
if (menu.needsUpdate()) { update the lcd… }
```

You will use this method to find out if you need to update the LCD. This could happen if the user clicked one of the directional keys or if you used the LCD while carrying out an action.

void Menu::updated()

```
menu.updated();
```

You will use this method to tell the Menu Library that you are finished updating the LCD with the menu items. Typically, after you have redrawn the menu

Here is an example code using those methods to display the menu:

```
void showMenu() {
  if (menu.needsUpdate()) {
    lcd.clear();
    for (int i = 0 ; i < lcdNumRows ; i++) {
      lcd.setCursor(0,i);
      lcd.print(menu.lcdLine(i));
    }
    delay(100);
    menu.updated();
  }
}
```

And the loop() section of your sketch:

```
void loop() {
  showMenu();                       //Update the LCD with the menu.
  int action = menu.update();       //Read the action tagged to the (new) current menu item.
  if (action > 0) {                 //If we need to act:
    make(action);                     //make it.
    menu.done();                      //We are done with this action... Go back to the menu.
  }
}
```

Remember that if you are letting the Menu Library display the menu, you only gave it a **pointer** to your LCD. You have full control of your LCD during an **action** phase of your sketch.

Keypad managing methods (Needed only if the Menu Library is handling the keypad)

bool Menu::keyPressed()

```
if menu.keyPressed() {…;}
```

You will use this method to see if any key is presently pressed. It will return true if a key is presently pressed, false otherwise.

bool Menu::isPressed(int key)

```
if menu.isPressed(myKey) {…;}
```

You will use this method to see if a specific key is presently pressed. It will return true if the key is presently pressed, false otherwise.

bool Menu::wasPressed(int key)

```
if menu.wasPressed(myKey) {…;}
```

You will use this method to see if a key is presently pressed. It will wait for the key to be released before returning true if the key is presently pressed, false otherwise.

int Menu::readKey()

```
myKey = menu.readKey()
```

You will use this method to get the ID of the key that is presently pressed. It will return one of the following: Up = 1, Down = 2, Left = 3, Right = 4 and “no key pressed” = 0.

int Menu::readKeyWithRepeat(int delay, int sensitivity)

```
while (true) {
  int myKey = menu.readKeyWithRepeat(500, 100)
  
  //Do something with that key;
  
  break; //when finished
}
```

You will use this method to get the ID of the key that is presently pressed. The ID of the key will be repeatedly sent to your sketch after a certain delay. It will return one of the following: Up = 1, Down = 2, Left = 3, Right = 4 and “no key pressed” = 0.

delay is the time in milliseconds before the method starts to send more keypresses.
sensitivity is the time in milliseconds between the subsequent keypresses are sent.

Note:

- All those methods have the same syntax whether four digital pins or one analog pin are used.
- All the keypresses are debounced. The digital keys have to sum 500 positive reads before the result is returned. The analog keys are averaged on 300 reads before the result is returned.

Menu managing method (Needed by all)

void Menu::done()

```
menu.done();
```

You will use this method to tell the Menu Library that you are finished carrying an action. This will raise the flag that will make “LcdNeedsUpdate()” return true the next time you call it.

Menu managing method (Needed only if you are handling your keypad)

You will use this method tell the Menu Library that a directional key has been pressed (and to specify which one).

int Menu::update(char key)   OR
int Menu::update(int key)

```
key = keypad.getKey();
action = menu.update(key);
```

You can pass either a character (char) or an integer (int) to the method, depending on what your keypad Library returns.

The update() method returns what action has to be carried out by your sketch according to the selected menu item. If it returns zero (0), you have no action to carry. If it returns a non-zero value, you will have to **carry out the action** corresponding to this value.

Menu managing method (Needed only if the Menu Library is handling the keypad)

int Menu::update()

```
action = menu.update();
```

You will use this method tell the Menu Library to read the keypad. The method will wait for a key to be pressed and released on the keypad. If there is no key pressed, there is no action to carry out. Waiting for the release prevents multiple calls to an action when a key if pressed for too long (In some cases, a fraction of a second could fire multiple calls to an action. The Arduino is clocked at 16Mhz)

The update() method returns what action has to be carried out by your sketch according to the selected menu item. If it returns zero (0), you have no action to carry. If it returns a non-zero value, you will have to **carry out the action** corresponding to this value.

Other useful methods

These methods will allow you to retrieve some information from the Menu Library.

String Menu::getCurrentLabel()

```
lcd.print(menu.getCurrentLabel());
```

You will use this method to get the label associated to the current menu item.

int Menu::getCurrentItem()

```
ptr = menu.getCurrentItem();
```

You will use this method to find the number of the current item

void Menu::setCurrentItem(String itemLabel)

```
menu.setCurrentItem("MOVE SERVOS"); \\Sets "MOVE SERVOS" as the current menu item
```

You will use this method to set the current item to a specific item in the menu, using its label.

void Menu::reset()

```
menu.reset();
```

You will use this method to set the first item of the menu as the current item.

int Menu::getAction()

```
action = menu.getAction();
```

You will use this method when you need to know what is the action corresponding to the current item, without using the “menu.update()” method.

EXAMPLE

```
#include <Menu.h>
#define UP 1
#define DOWN 2
#define LEFT 3
#define RIGHT 4

#define ANALOG 1
#define DIGITAL 0
#define PINMOTOR 26

////////////////////////////////////////////////////////////////////////////////////////////////////THE LCD
int lcdNumCols = 20;
int lcdNumRows = 4;

//Depending on the LDC library that you are using, comment out the other one

//See: https://www.arduino.cc/en/Tutorial/HelloWorld
//#include <LiquidCrystal.h>                    //Include for LiquidCrystal
//LiquidCrystal lcd(7, 6, 5, 4, 3, 2);          //Constructor for LiquidCrystal

//See: https://github.com/johnmccombs/arduino-libraries/tree/master/LiquidTWI
#include <Wire.h>
#include <LiquidTWI.h>                          //Include for LiquidTWI
LiquidTWI lcd(0);                               //Constructor for LiquidTWI

///////////////////////////////////////////////////////////////////////////////////////////////////THE MENU
const String menuItems =
"-READ PINS:000"
"--SENSORS:000"
"---SENSOR A1:101"
"---SENSOR A2:102"
"--SWITCHES:000"
"---SWITCH PIN 24:103"
"---SWITCH PIN 26:104"
"-MOTOR:000"
"--START:105"
"--STOP:106";
Menu menu(menuItems); //Set up menu

/////////////////////////////////////////////////////////////////////////////////////////ACTIONS 101 to 104
void readPin(int pin, int pinType) {
  int key = 0;                                          //Initialise key as NoKeyPressed
  lcd.clear();                                          //Clear the LCD
  lcd.setCursor(0,0);                                   //On the first row
  lcd.print(menu.getCurrentLabel());                    //Print the label for that menu item.
  while(key != LEFT && key != RIGHT) {                  //Exit on LEFT or RIGHT.
    lcd.setCursor(0, 1);                                  //On the second row
    lcd.print("   ");                                     //Erase the old value
    lcd.setCursor(0, 1);                                  //On the second row
    if (pinType == DIGITAL)                               //If it is a digital pin,
      lcd.print(digitalRead(pin) ? "HIGH" : "LOW");         //Print HIGH or LOW
    else                                                  //If it is an analog pin, (not digital)
      lcd.print(analogRead(pin));                           //Print value (0:1023)
    key = menu.readKey();                                 //Read the keypad.
    delay(100);                                           //Reduce flickering.
  }
}

//////////////////////////////////////////////////////////////////////////////////////////////ACTION SELECT
void make(int action) {
  switch (action) {
    case 101: { readPin(A1, ANALOG); break; }
    case 102: { readPin(A2, ANALOG); break; }
    case 103: { readPin(22, DIGITAL); break; }
    case 104: { readPin(24, DIGITAL); break; }
    case 105: { digitalWrite(PINMOTOR, HIGH); break; }
    case 106: { digitalWrite(PINMOTOR, LOW); break; }
  }
}
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////SETUP
void setup() {
  Serial.begin(9600);
  lcd.begin(lcdNumCols, lcdNumRows);                   //Initialize the LCD
  menu.handleLcd(&lcd, lcdNumCols, lcdNumRows);        //Give a pointer of your LCD to the Menu Library
//Comment out the Keypad that you are not using
  menu.handleSwitches(11,10,9,8);                      //Give the pins number of your digital keypad
//  menu.handleSwitches(A0);                             //Give the pin number of your analog keypad

  pinMode(22,INPUT_PULLUP);      //Setup for the actions
  pinMode(24,INPUT_PULLUP);
  pinMode(PINMOTOR, OUTPUT);
}

//////////////////////////////////////////////////////////////////////////////////////////////////////LOOP
void loop() {
  menu.showMenu();                 //Update the LCD with the menu.
  int action = menu.update();      //Read the action taged to the curent menu item.
  if (action > 0) {                //If we need to act:
    make(action);                    //make it.
    menu.done();                     //We are done with this action... Go back to the menu.
  }
}
```

IS THERE ANYTHING ELSE I SHOULD KNOW?

Yes. It is regarding the amount of memory that the sketch’s variables occupy. When you check or upload your sketch, Arduino’s IDE tells you how much memory you use for your variables. In order to let you decide how many menu items you want, the Menu Library allocates just the amount of memory necessary to store information about each of your menu item. This is done without Arduino’s IDE knowing about it.

It is called dynamic memory allocation. You don’t have to know anything about this technique, but you need to know that your sketch will use more memory for its variables than what Arduino reports.

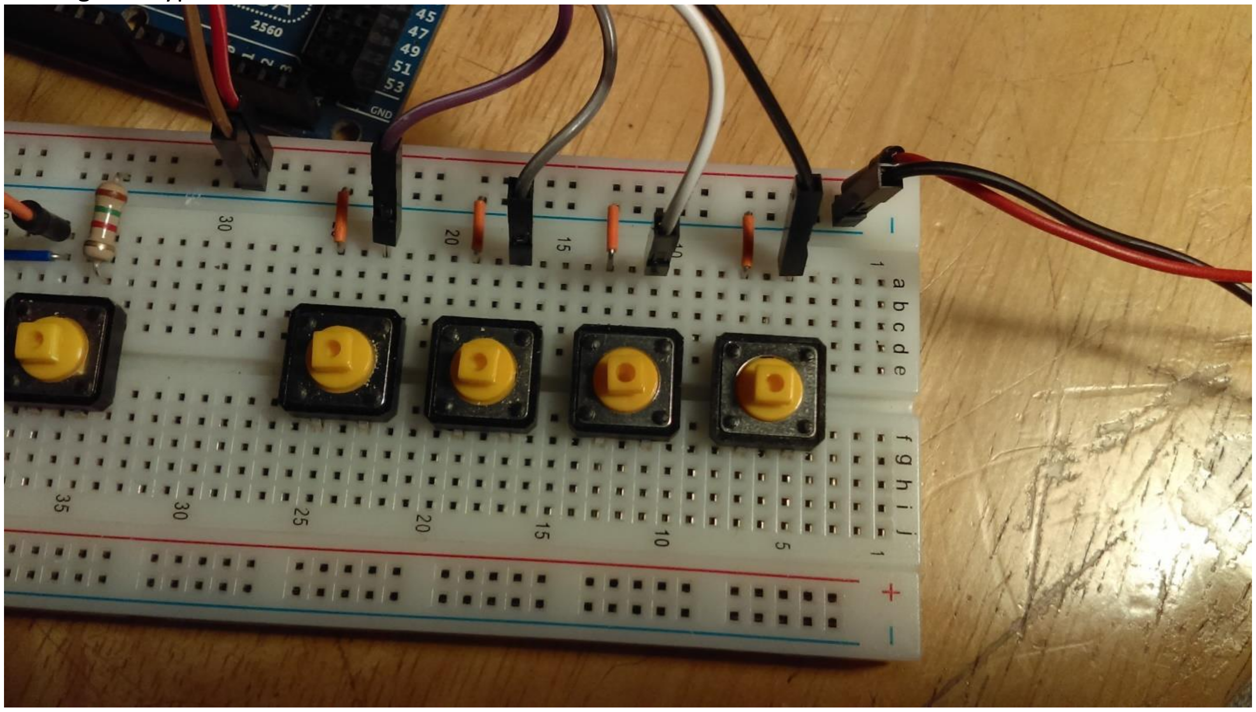
How much more? The Menu Library uses 7 bytes per menu item, plus another 7 bytes for its own use. So, if your menu has 10 items you have to add (10 x 7) + 7 = 77 bytes to the amount of dynamic memory that Arduino reports to you.

If you have loads of dynamic memory (a Mega or the like) you can have menus with as many items as you wish. If you are tight on dynamic memory, you should take this into account.

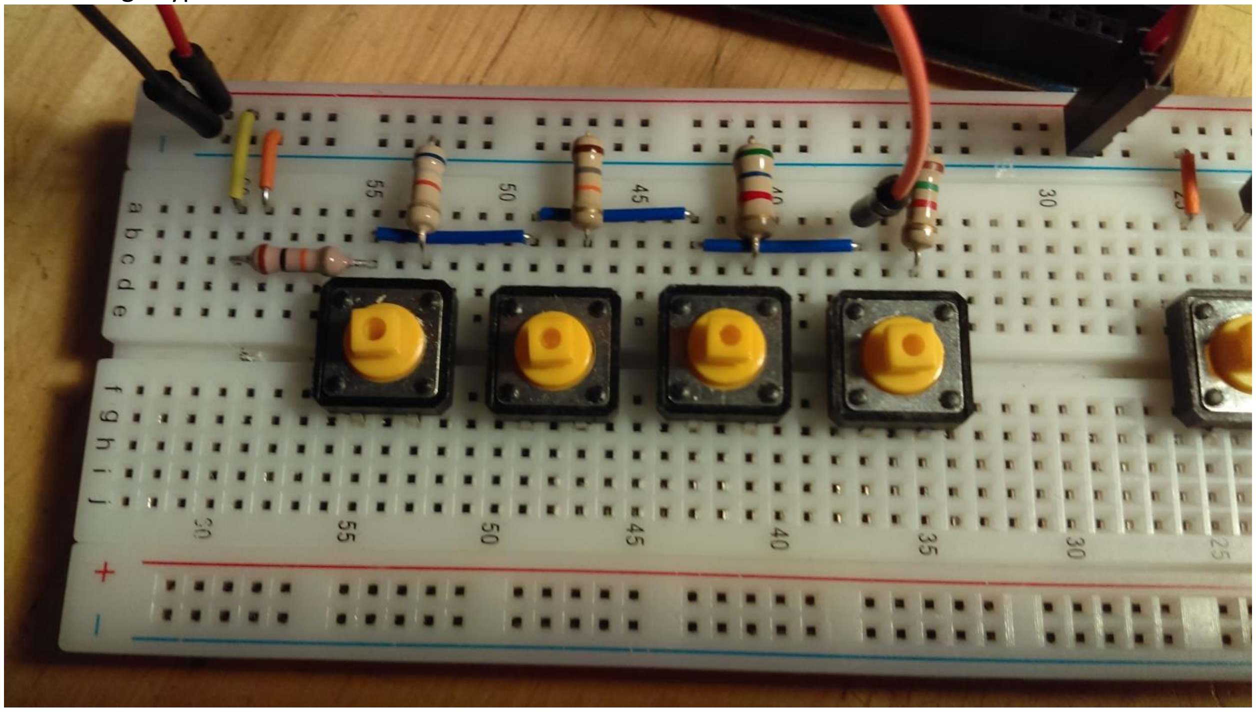
I sincerely hope that the Menu Library will help you in your projects.
Jacques Bellavance

Appendix A : The Workbench

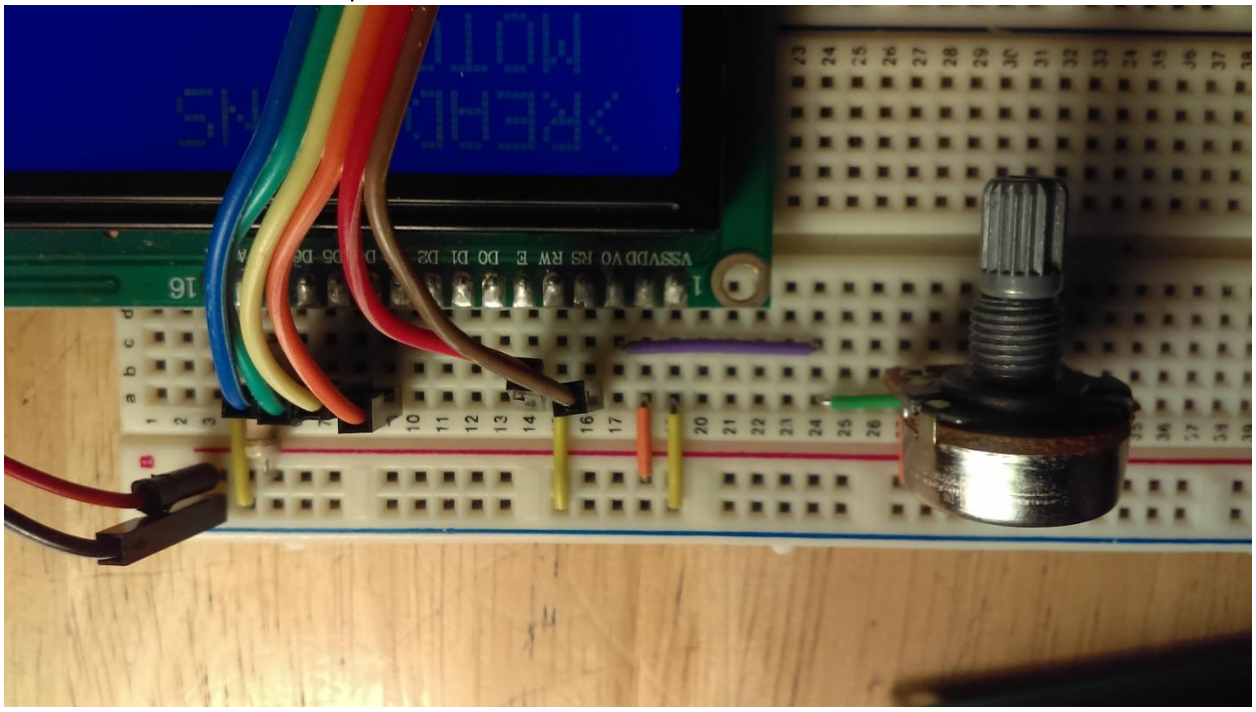
The digital keypad

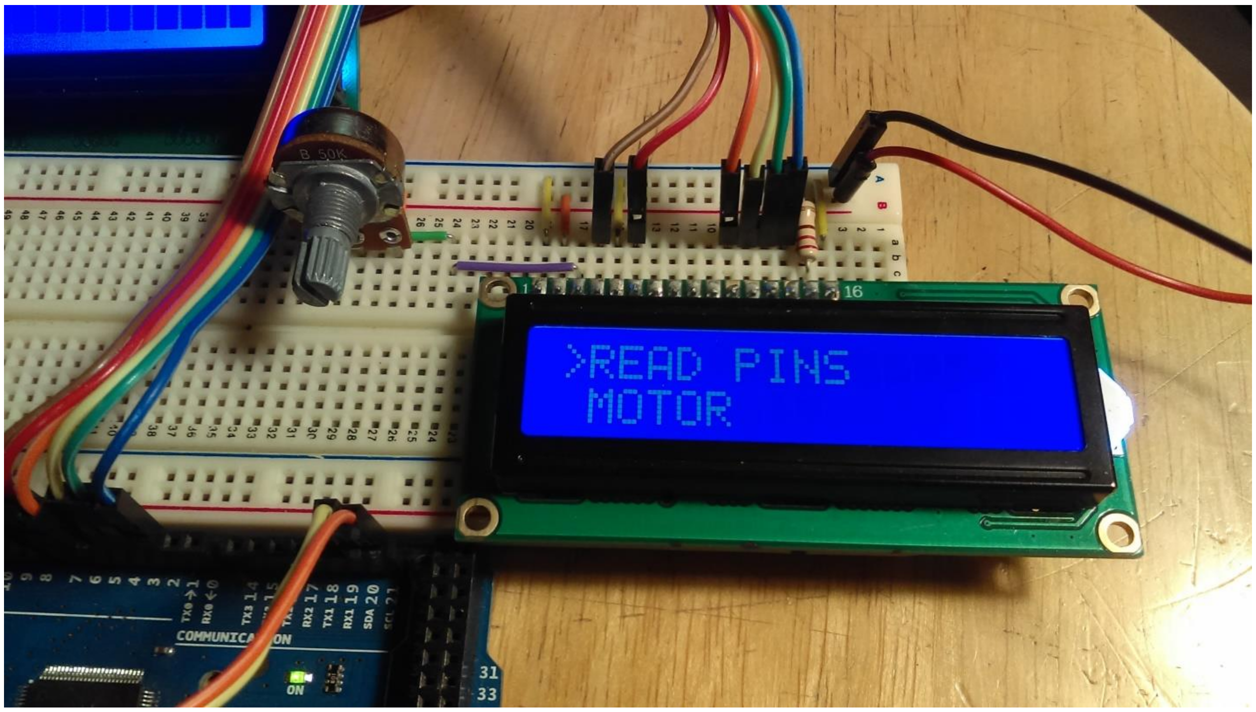


The analog keypad



The LDC without the I²C backpack





The whole thing