Keypad on an Arduino

1. Keypad Description

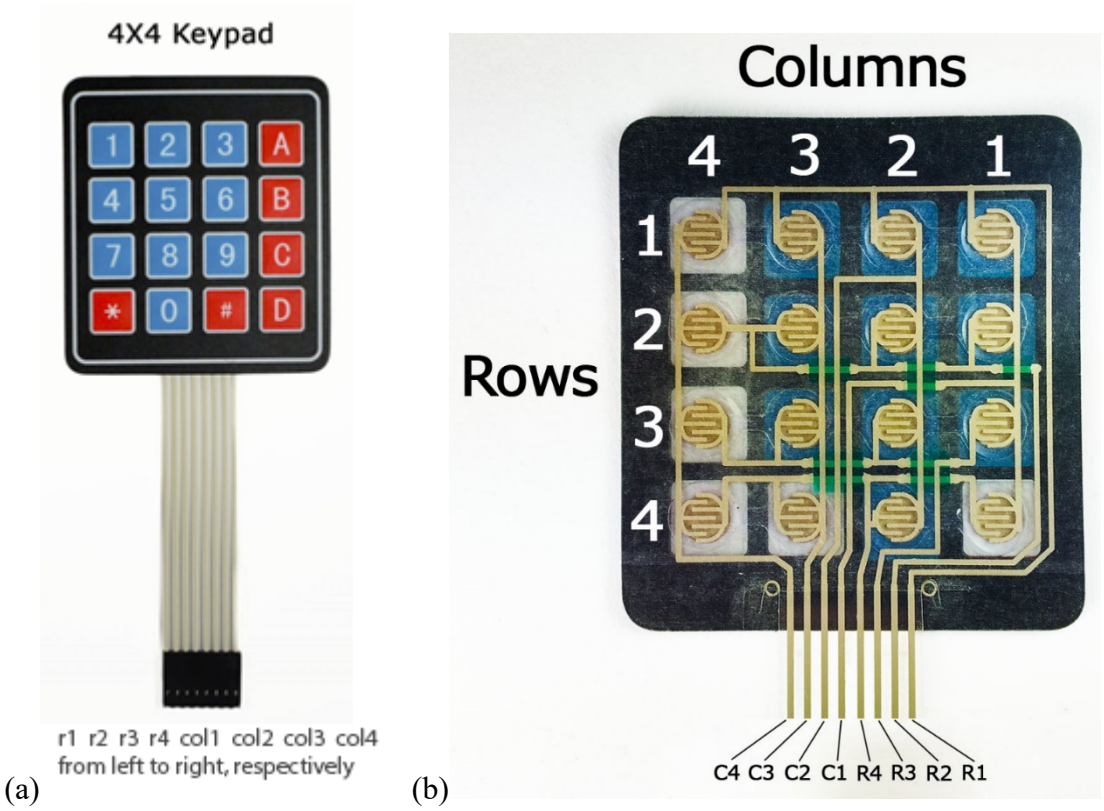
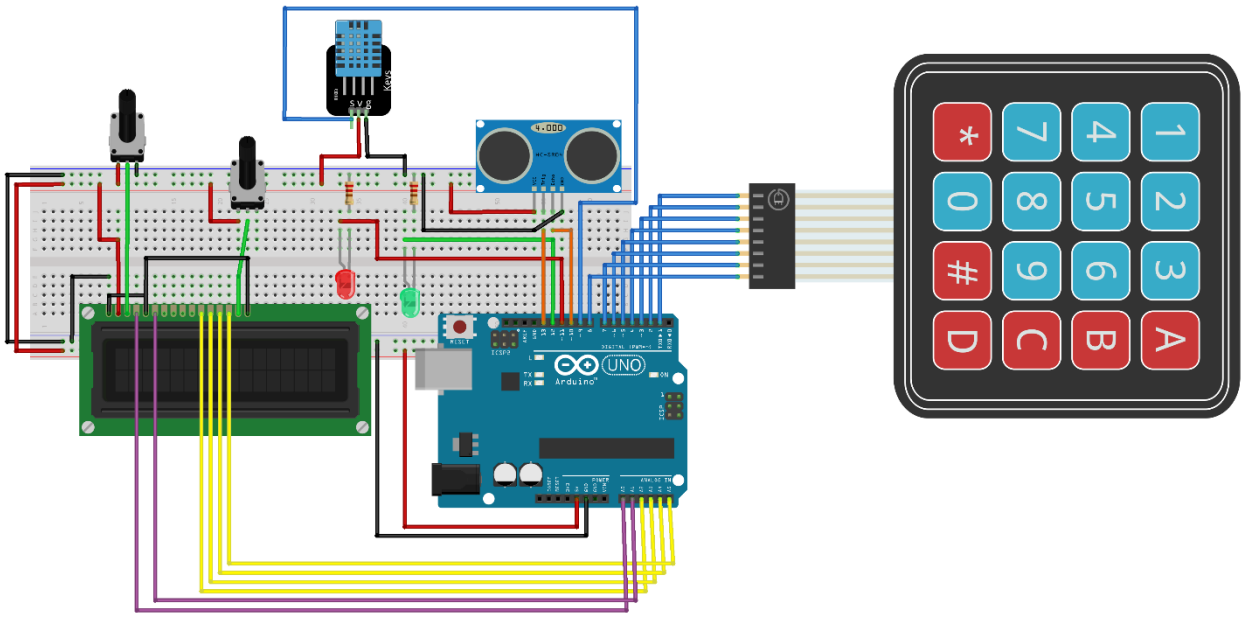


Fig 1: (a) 4x4 keypad, (b) fabrication schematic, (c) switch-based schematic

A keypad is a very useful input peripheral for entering data. Just like a keyboard is useful to a computer, Many applications may rely on a user entering basic inputs such as the ones available on this keypad.
A basic procedure to read an input from a keypad starts with a loop that connects one of the rows to ground (say row 1), then read each column. If say column 4 reads a 0, then the key that was pressed is A. Otherwise, move on, and connect row 2 to ground, and keep reading the columns until you find which key has been pressed.

2. Circuit Test



Though the temperature and humidity sensor and the ultrasonic sensor appear on the circuit, the only peripherals that are being tested in this first circuit are the keypad and the LCD display for visualization of the messages.
The connections had to be changed from the previous cases because the 4x4 keypad required the connection to 8 digital pins. Hence, we decided to use the analog pins to interface with the LCD display.

Code

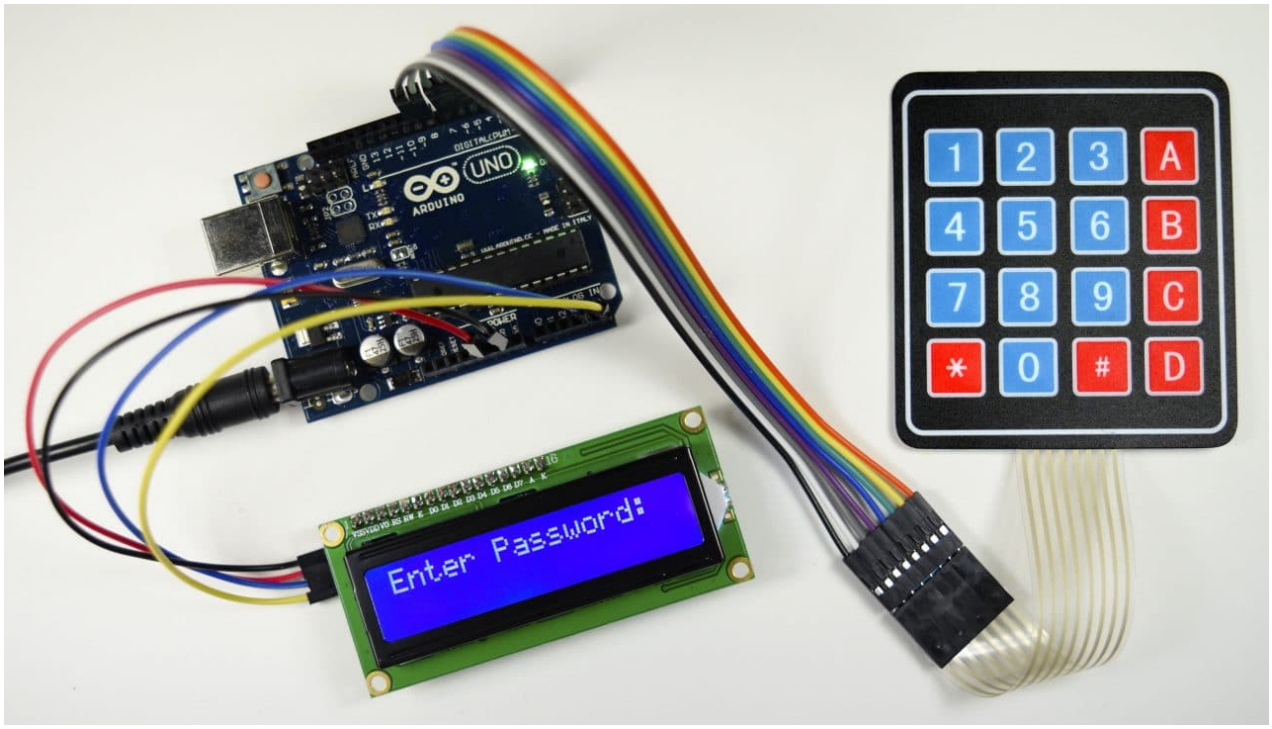
```
/* Arduino Security System with the Keypad and LCD

Simple one-digit password

 */


#include <LiquidCrystal.h> //include LCD library

#include <Keypad.h> //include keypad library
```

```
#define redLED 11 //define the LED pins
#define greenLED 12

const byte rows = 4; //number of the keypad rows
const byte cols = 4; // number of the keypad columns

char keyMap [rows] [cols] = { //define the symbols on the buttons of the keypad

 {'1', '2', '3', 'A'},
 {'4', '5', '6', 'B'},
 {'7', '8', '9', 'C'},
 {'*', '0', '#', 'D'}
};
char password ='6'; //create a password
byte rowPins [rows] = {1, 2, 3, 4}; // digital pins related to the row pins of the keypad
byte colPins [cols] = {5, 6, 7, 8}; // digital pins related to the column pins of the keypad

Keypad myKeypad = Keypad( makeKeymap(keyMap), rowPins, colPins, rows, cols);
LiquidCrystal lcd (A0, A1, A2, A3, A4, A5); // pins of the LCD. (RS, E, D4, D5, D6, D7)

void setup(){

 lcd.begin(16, 2);
 pinMode(redLED, OUTPUT);  //set the LED as an output
 pinMode(greenLED, OUTPUT);
 setLocked (true); //state of the password
}
```

```
void loop(){

 // char whichKey = myKeypad.getKey(); //define which key is pressed with getKey

 lcd.setCursor(0, 0);
 lcd.print("    Welcome");
 lcd.setCursor(0, 1);
 lcd.print(" Enter Password");

 char whichKey = myKeypad.waitForKey(); //define which key is pressed with getKey

 if(whichKey == '*' || whichKey == '#' || whichKey == 'A' ||       //define invalid keys
 whichKey == 'B' || whichKey == 'C' || whichKey == 'D')
 {
  // location=0;
  setLocked (true);
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("  Invalid Key! ");
  delay(1000);
  lcd.clear();
 }


 if(whichKey == password )
 {
  setLocked (false);
  lcd.clear();
  lcd.setCursor(0, 0);
```

```
   lcd.print("   Accepted   ");
   delay(1000);
   lcd.clear();
   lcd.setCursor(0, 0);
   lcd.clear();
   setLocked (true);
  }
  else
  {
   setLocked (true);
   lcd.clear();
   lcd.setCursor(0, 0);
   lcd.print("   Try again   ");
   delay(1000);
   lcd.clear();
   lcd.setCursor(0, 0);
   lcd.clear();
  }
  delay(200);
}

void setLocked(int locked){
  if(locked){
   digitalWrite(redLED, HIGH);
   digitalWrite(greenLED, LOW);
   }
   else{
     digitalWrite(redLED, LOW);
     digitalWrite(greenLED, HIGH);
```

```
  }
}
```



Keypads are a great way to let users interact with your project. You can use them to navigate menus, enter passwords, and control games and robots.

In this tutorial, I'll show you how to setup a keypad on the Arduino. First I'll explain how the Arduino detects key presses, then I'll show you how to find the pinout of any keypad. As a simple example, I'll show you how to print out the key presses on the serial monitor and an LCD. Finally, I'll show you how to activate a 5V relay when a password is entered correctly.

BONUS: I made a quick start guide for this tutorial that you can download and go back to later if you can't set this up right now. It covers all of the steps, diagrams, and code you need to get started.

I'll be using a 4X4 matrix membrane keypad in this article, but there's also code and wiring diagrams for 3X4 matrix keypads as well. I like membrane style keypads because they're thin and they also have adhesive backing so you can stick them to most flat surfaces. You can also get telephone style keypads that have thicker buttons if you like that style better. Even salvaged keypads from old telephones will work with the Arduino.

How Keypads Work

The buttons on a keypad are arranged in rows and columns. A 3X4 keypad has 4 rows and 3 columns, and a 4X4 keypad has 4 rows and 4 columns:

Beneath each key is a membrane switch. Each switch in a row is connected to the other switches in the row by a conductive trace underneath the pad. Each switch in a column is connected the same way – one side of the switch is connected to all of the other switches in that column by a conductive trace. Each row and column is brought out to a single pin, for a total of 8 pins on a 4X4 keypad:

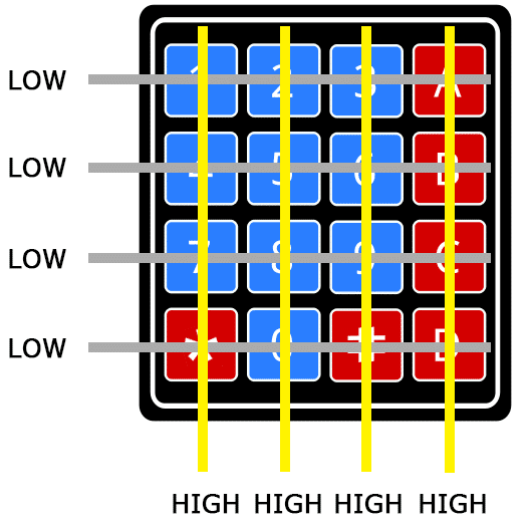
Pressing a button closes the switch between a column and a row trace, allowing current to flow between a column pin and a row pin.

The schematic for a 4X4 keypad shows how the rows and columns are connected:

The Arduino detects which button is pressed by detecting the row and column pin that's connected to the button.

This happens in four steps:

1. First, when no buttons are pressed, all of the column pins are held HIGH, and all of the row pins are held LOW:



2. When a button is pressed, the column pin is pulled LOW since the current from the HIGH column flows to the LOW row pin:

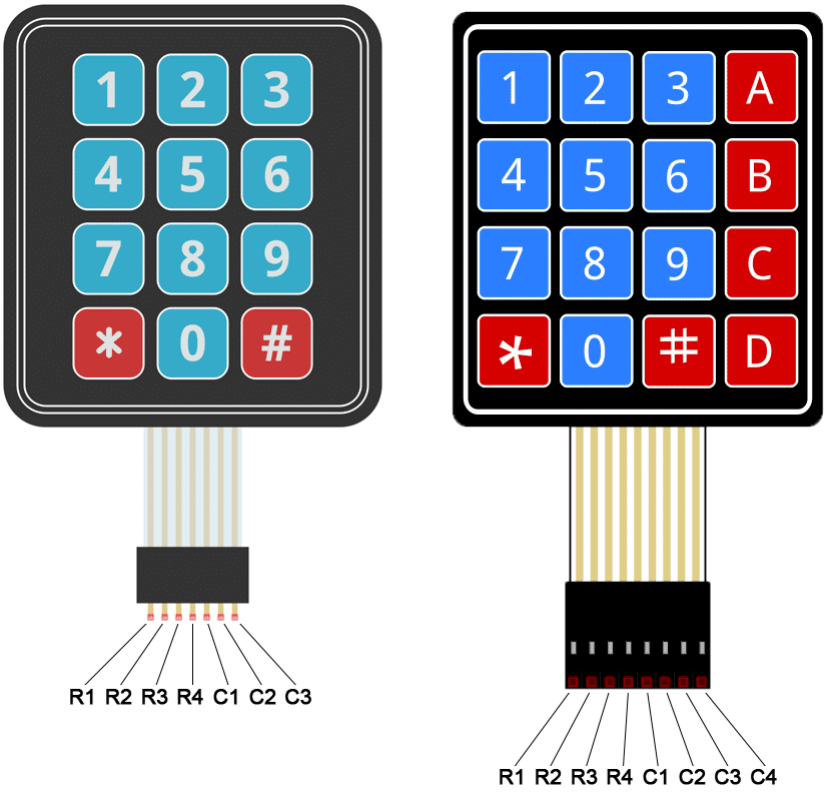
3. The Arduino now knows which column the button is in, so now it just needs to find the row the button is in. It does this by switching each one of the row pins HIGH, and at the same time reading all of the column pins to detect which column pin returns to HIGH:

4. When the column pin goes HIGH again, the Arduino has found the row pin that is connected to the button:

From the diagram above, you can see that the combination of row 2 and column 2 could only mean that the number 5 button was pressed.

Connect the Keypad to the Arduino

The pin layout for most membrane keypads will look like this:



Follow the diagrams below to connect the keypad to an Arduino Uno, depending on whether you have a 3X4 or 4X4 keypad:

How to Find the Pinout of Your Keypad

If your keypad's pin layout doesn't match the ones above, you can probe the pins to figure it out. You'll need to build a test circuit by connecting an LED and a current limiting resistor to the Arduino (or any 5V power source) like this:

First, find out which keypad pins are connected to the button rows. Insert the ground (black) wire into the first pin on the left. Press any button in row 1 and hold it down. Now insert the positive (red) wire into each one of the other pins. If the LED lights up at one of the pins, press and hold another button in row 1, then insert the positive wire into each one of the other pins again. If the LED lights up on a different pin, it means the ground wire is inserted into the row 1 pin. If none of the buttons in row 1 make the LED light up, the ground wire is not connected to row 1. Now move the ground wire over to the next pin, press a button in a different row, and repeat the process above until you've found the pin for each row.

To figure out which pins the columns are connected to, insert the ground wire into the pin you know is row 1. Now press and hold any one of the buttons in that row. Now insert the positive wire into each one of the remaining pins. The pin that makes the LED light up is the pin that's connected to that button's column. Now press down another button in the same row, and insert the positive wire into each one of the other pins. Repeat this process for each one of the other columns until you have each one mapped out.

Programming the Keypad

For a basic demonstration of how to setup the keypad, I'll show you how to print each key press to the serial monitor.

Install the Library

We'll use the Keypad library by Mark Stanley and Alexander Brevig. This library takes care of setting up the pins and polling the different columns and rows. To install the Keypad library, go to Sketch > Include Library > Manage Libraries and search for "keypad". Click on the library, then click install.

The Code for a 4X4 Keypad

Once the Keypad library is installed, you can upload this code to the Arduino if you're using a 4X4 keypad:

```
#include <Keypad.h>

const byte ROWS = 4;
const byte COLS = 4;
```

```
char hexaKeys[ROWS][COLS] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};

byte rowPins[ROWS] = {9, 8, 7, 6};
byte colPins[COLS] = {5, 4, 3, 2};

Keypad customKeypad = Keypad(makeKeymap(hexaKeys), rowPins, colPins, ROWS,
COLS);

void setup(){
  Serial.begin(9600);
}

void loop(){
  char customKey = customKeypad.getKey();

  if (customKey){
    Serial.println(customKey);
  }
}
```

The Code for a 3X4 Keypad

If you're using a 3X4 keypad, you can use this code:

```
#include <Keypad.h>

const byte ROWS = 4;
const byte COLS = 3;

char hexaKeys[ROWS][COLS] = {
  {'1', '2', '3'},
  {'4', '5', '6'},
  {'7', '8', '9'},
  {'*', '0', '#'}
};

byte rowPins[ROWS] = {9, 8, 7, 6};
byte colPins[COLS] = {5, 4, 3};

Keypad customKeypad = Keypad(makeKeymap(hexaKeys), rowPins, colPins, ROWS,
COLS);

void setup(){
  Serial.begin(9600);
}

void loop(){
  char customKey = customKeypad.getKey();

  if (customKey){
```

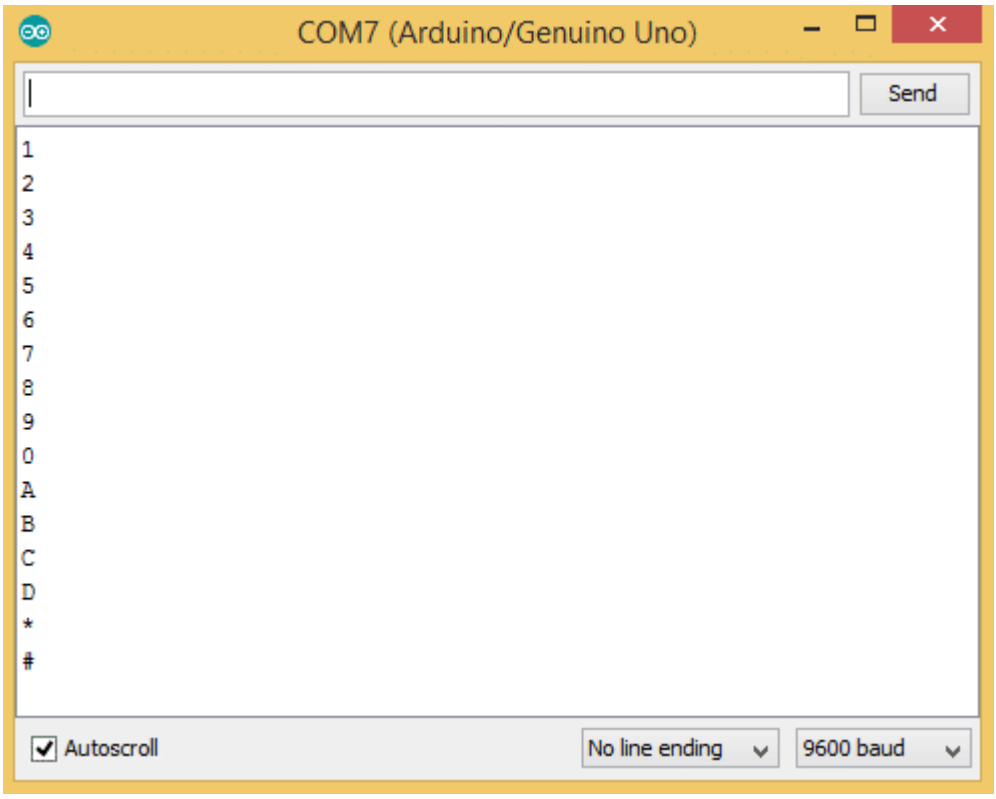
```
    Serial.println(customKey);
  }
}
```

Lines 3 and 4 in the code above set the number of rows and columns on the keypad.

Lines 6-11 define which characters are printed when a particular button is pressed on the keypad. The characters are laid out just as they appear on the keypad. If your keypad has a different layout, you can define which characters are printed when you press a button. For example, say your keypad has a column of letters on the left instead of the right. You would just change it to this:

```
char hexaKeys[ROWS][COLS] = {
 {'A', '1', '2', '3'},
 {'B', '4', '5', '6'},
 {'C', '7', '8', '9'},
 {'D', '*', '0', '#'}
};
```

After you upload the code, open the serial monitor. When you press a key, the value will be printed out:



Using an LCD with the Keypad

Now let's see how to print the key presses on an LCD. 4X4 keypads use 8 pins and 3X4 keypads use 7 pins. That takes up a lot of pins, so I'm going to use an I2C enabled LCD because it only needs 4 wires to connect to the Arduino.

Install the LiquidCrystal_I2C Library

To use an I2C enabled LCD on the Arduino, you'll need to install the LiquidCrystal I2C library by Marco Schwartz. This library is nice because it includes most of the functions available in the standard LiquidCrystal library. To install it, download the ZIP file below, then go to Sketch > Include Library > Add .ZIP Library:

LiquidCrystal I2C.zip

The Wire Library

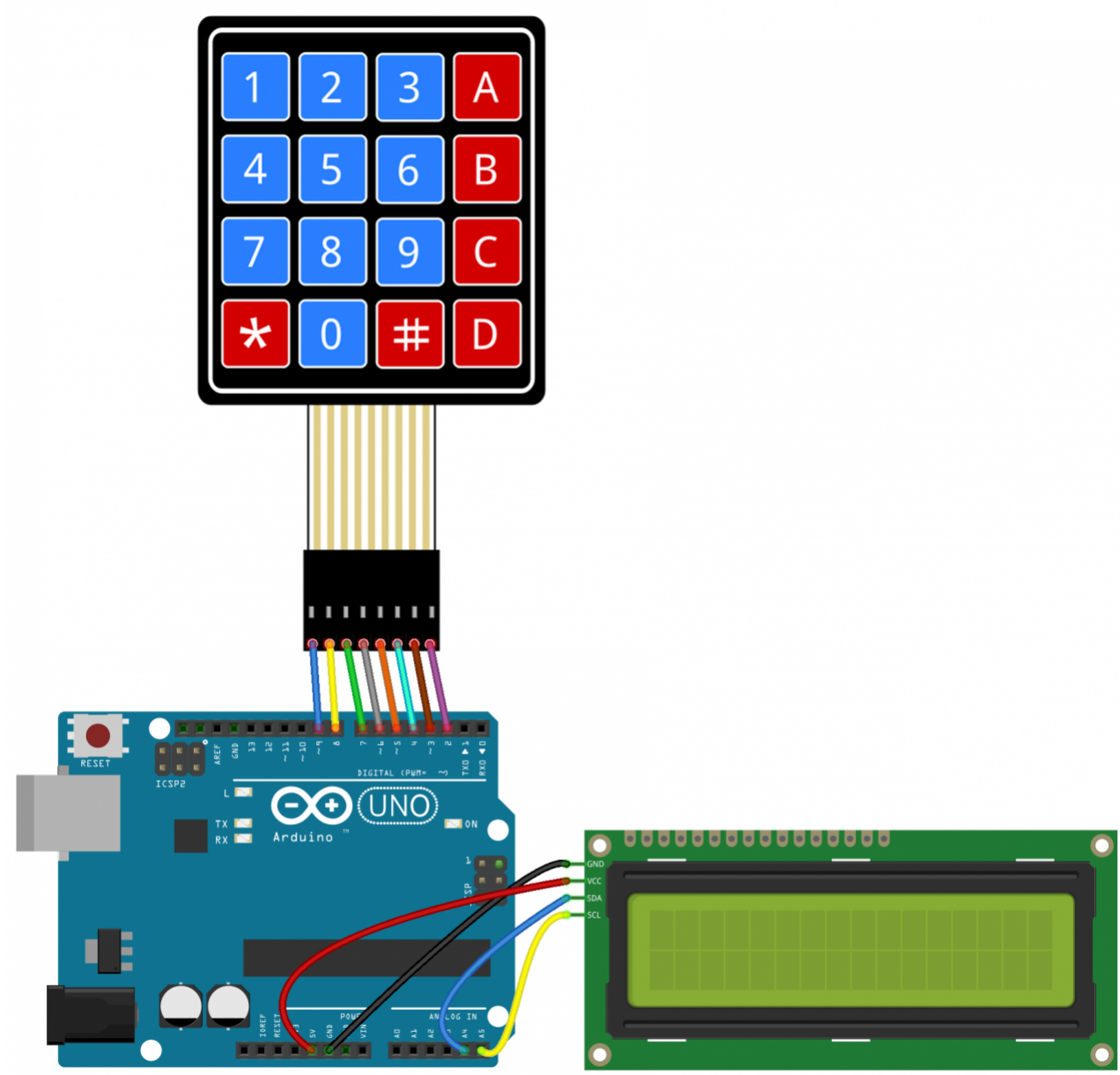
The Wire library is needed to add support for I2C communication. It comes packaged with the Arduino IDE, so there's no need to install it. But if for some reason it's not installed on your system, go to Sketch > Include Library > Manage Libraries and search for "wire" to install it.

Connect the Keypad and LCD

Once the libraries are installed, connect the ground and Vcc pins of the LCD to the Arduino, then connect the LCD's SDA and SCL pins according to the table below for the different Arduino boards:

Arduino	SDA Pin	SCL Pin
Uno	A4	A5
Nano	A4	A5
Mini	A4	A5
101	SDA	SCL
Zero	SDA	SCL
Leonardo	2	3
Micro	2	3
Due	20	21
Mega	20	21

Then connect the keypad to the Arduino. It should look something like this (for an Arduino Uno):



Code for Output to an LCD

Once everything is connected, upload this code to the Arduino:

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

const byte ROWS = 4;
const byte COLS = 4;

char hexaKeys[ROWS][COLS] = {
  {'1', '2', '3', 'A'},
```

```
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};

byte rowPins[ROWS] = {9, 8, 7, 6};
byte colPins[COLS] = {5, 4, 3, 2};

Keypad customKeypad = Keypad(makeKeymap(hexaKeys), rowPins, colPins, ROWS,
COLS);

LiquidCrystal_I2C lcd(0x21, 16, 2);

void setup(){
  lcd.backlight();
  lcd.init();
}

void loop(){
  char customKey = customKeypad.getKey();
  if (customKey){
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print(customKey);
  }
}
```

You'll need to add the I2C address of your LCD on line 20:

```
LiquidCrystal_I2C lcd(0x21, 16, 2);
```

The I2C address of my LCD is 0x21, but your's will probably be different. The I2C address of your LCD should be provided in the datasheet, but if not, you can find it by running this I2C_Scanner sketch.

Use a Password to Activate a Relay

One of the most useful applications of a keypad is to use it for keyed entry. You can set up a password and have the Arduino activate a relay or some other module if the password is correct. The following code will activate a 5V relay when the password is entered correctly:

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

#define Password_Length 8

int signalPin = 12;

char Data[Password_Length];
char Master[Password_Length] = "123A456";
byte data_count = 0, master_count = 0;
bool Pass_is_good;
```

```
char customKey;

const byte ROWS = 4;
const byte COLS = 4;

char hexaKeys[ROWS][COLS] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};

byte rowPins[ROWS] = {9, 8, 7, 6};
byte colPins[COLS] = {5, 4, 3, 2};

Keypad customKeypad = Keypad(makeKeymap(hexaKeys), rowPins, colPins, ROWS,
COLS);

LiquidCrystal_I2C lcd(0x21, 16, 2);

void setup(){
  lcd.init();
  lcd.backlight();
  pinMode(signalPin, OUTPUT);
}

void loop(){

  lcd.setCursor(0,0);
  lcd.print("Enter Password:");

  customKey = customKeypad.getKey();
  if (customKey){
    Data[data_count] = customKey;
    lcd.setCursor(data_count,1);
    lcd.print(Data[data_count]);
    data_count++;
    }

  if(data_count == Password_Length-1){
    lcd.clear();

    if(!strcmp(Data, Master)){
      lcd.print("Correct");
      digitalWrite(signalPin, HIGH);
      delay(5000);
      digitalWrite(signalPin, LOW);
      }
    else{
      lcd.print("Incorrect");
      delay(1000);
      }

    lcd.clear();
    clearData();
  }
}
```

```
void clearData(){
  while(data_count !=0){
    Data[data_count--] = 0;
  }
  return;
}
```

You can change the password on line 10 by replacing the `123A456` text with your own password:

```
char Master[Password_Length] = "123A456";
```

The length of the password needs to be set on line 5:

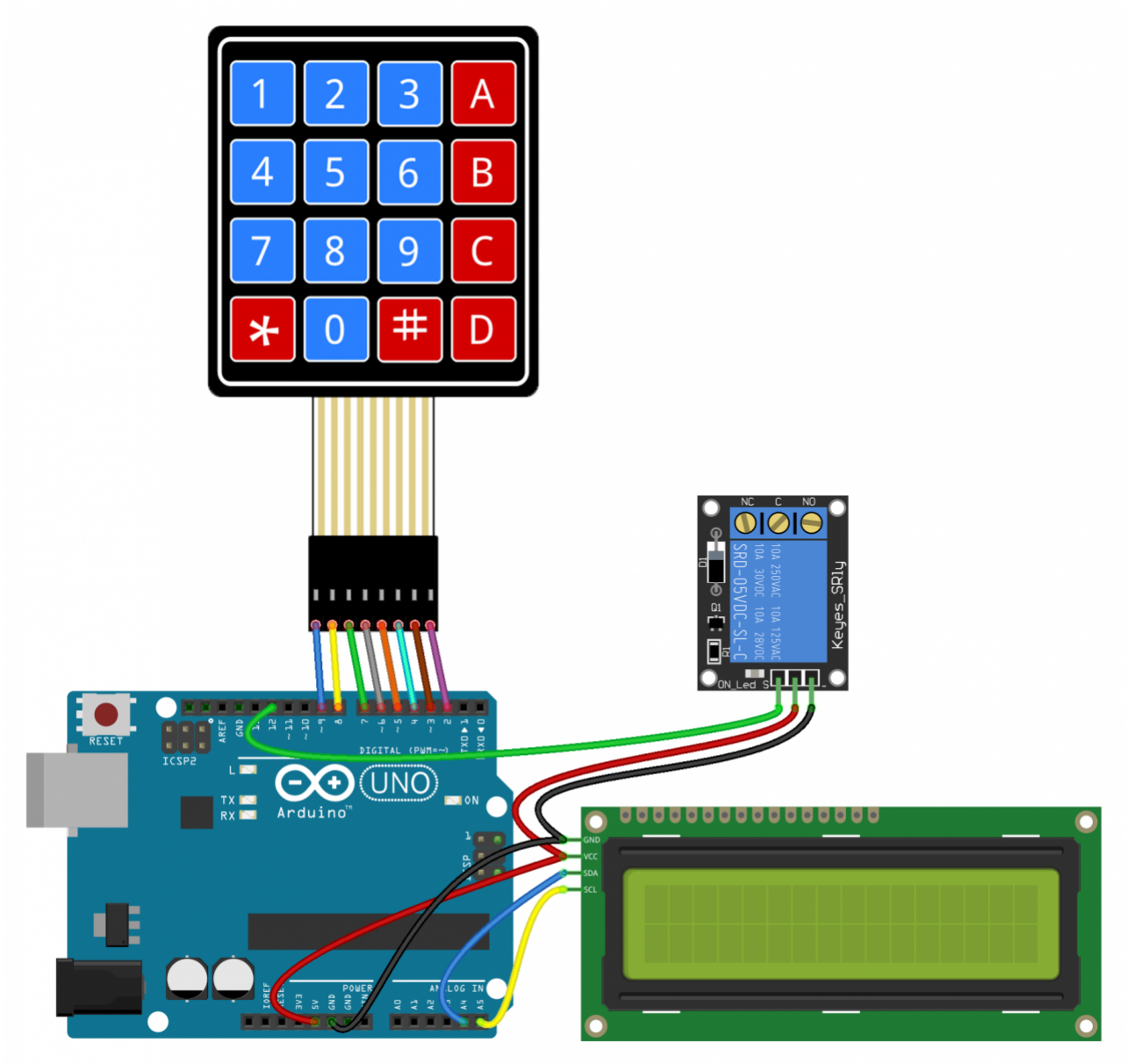
```
#define Password_Length 8
```

The password in the example above is only 7 characters long, but the password length is actually one greater than 7 because there is a null character added to the end of the string. For example, if your password is 5 characters long, you would enter 6 for the password length.

The output pin that activates the relay is defined on line 7:

```
int signalPin = 12;
```

After connecting everything to the Arduino, you should have something that looks like this:



Well that's about it. It's not hard to set up a keypad at all. I think with a little trial and error you should be able to modify the code above to work with most of the projects you'd want to use a keypad for. But if you run into problems, just let us know in the comments and we'll try to help you out.

Krishna Pattabiraman is a frequent guest writer on Circuit Basics and the founder of www.codelectron.com.

Keypad Library for Arduino

Authors: Mark Stanley, Alexander Brevig
Contact: mstanley@technologist.com
Contact: alexanderbrevig@gmail.com

I am Alexander, the lead developer for libraries
that ship with the Wiring distribution. I will be
maintaining my libraries here:
https://wiring.uniandes.edu.co/source/trunk/wiring/

As of version 1.0 - Wiring supports Arduino boards.
You are welcome to check it out! Please visit
https://wiring.org.co/download/

---

1 Navigation

---

2 Current version

3.0 2012-07-12 - Mark Stanley : Made library multi-keypress by default. (Backwards compatible)
2015-09-18 - Code has been replicated to GitHub for IDE's 1.6.2 and above. Small changes have been made to conform with the latest library specification, but the functional code is intact. To notify of changes or raise issues visit: https://github.com/Chris--A/Keypad
---

3 History

3.0 2015-09-18 - Christopher Andrews : Copied to GitHub for modern IDE library manager.
3.0 2012-07-12 - Mark Stanley : Modified pin functions to support Keypad_I2C
3.0 2012-07-12 - Stanley & Young : Fix for multiple keypad objects.

3.0 2012-07-12 - Mark Stanley : Fixed bug that caused shorted pins.
2.0 2011-12-29 - Mark Stanley : Added Nick Gammon's changes.
2.0 2011-12-29 - Mark Stanley : Added waitForKey()
2.0 2011-12-23 - Mark Stanley : Rewrote state machine.
2.0 2011-12-23 - Mark Stanley : Significant speed improvements.
1.8 2011-11-29 - Tom Putzeys : Enabled internal pull-ups on non-active columns
1.8 2011-11-21 - Mark Stanley : Added test for version 1.0 of the IDE
1.8 2009-07-08 - Alexander Brevig : Added no restrictions on sizes or keymaps
1.8 2009-07-08 - Alexander Brevig : Added no restrictions on sizes or keymaps
See source files for a complete change history.

---

4 Description

Keypad is a library for using *matrix* style keypads with the Arduino. As of version 3.0 it now supports mulitple keypresses.

This library is based upon the Keypad Tutorial.

It was created to promote Hardware Abstraction. It improves readability of the code by hiding the pinMode and digitalRead calls for the user.

Keypad library is part of the Hardware Abstraction libraries.

Version 3.0 has just been posted (19 July 2012) and was rewritten to support multi-keypresses by default. But for those who still need the original single-keypress functionality, the library is fully backwards compatible.

You won't need external resistors or diodes because the library uses the internal pullup resistors and additonally ensures that all unused column pins are high-impedance.

Support was added to allow other hardware to be used along with a keypad. *Joe Young's keypad library added support for several I2C expander chips.* You can find it here: https://github.com/joeyoung/arduino_keypads

---

5 Download, install and import

This library is now available via the Arduino IDE library manager. If you are using a modern IDE (1.6.2 or above), you can simply use the menu:

Sketch->Include Library->Manage Libraries... Then search for *Keypad*.

Once found, click on its entry and the install button will appear. The zip file below is for the retro IDE's (not recommended for use, upgrade!).

Download here: keypad.zip

Put the Keypad folder in "*arduino*\libraries\".
In the Arduino IDE, create a new sketch (or open one) and select from the menubar "Sketch -> Import Library -> Keypad".
Once the library is imported, an "#include <Keypad.h>" line will appear at the top of your Sketch.

---

6 Creation

Constructors:

1. Keypad(makeKeymap(userKeymap), row[], col[], rows, cols)

```
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] = {5, 4, 3, 2}; //connect to the row pinouts of the keypad
```

```
byte colPins[cols] = {8, 7, 6}; //connect to the column pinouts of the keypad
Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, rows, cols );
```

Instantiates a Keypad object that uses pins 5, 4, 3, 2 as row pins, and 8, 7, 6 as column pins. This keypad has 4 rows and 3 columns, resulting in 12 keys.

---

7 Functions

7.1 void begin(makeKeymap(userKeymap))

Initializes the internal keymap to be equal to userKeymap
[See File -> Examples -> Keypad -> Examples -> CustomKeypad]

7.2 char waitForKey()

This function will wait forever until someone presses a key. **Warning:** It blocks all other code until a key is pressed. That means no blinking LED's, no LCD screen updates, no nothing with the exception of interrupt routines.

7.3 char getKey()

Returns the key that is pressed, if any. This function is non-blocking.

7.4 KeyState getState()

Returns the current state of any of the keys.
The four states are IDLE, PRESSED, RELEASED and HOLD.

7.5 boolean keyStateChanged()

New in version 2.0: Let's you know when the key has changed from one state to another. For example, instead of just testing for a valid key you can test for when a key was pressed.

7.6 setHoldTime(unsigned int time)

Set the amount of milliseconds the user will have to hold a button until the HOLD state is triggered.

7.7 setDebounceTime(unsigned int time)

Set the amount of milliseconds the keypad will wait until it accepts a new keypress/keyEvent. This is the "time delay" debounce method.

7.8 addEventListener(keypadEvent)

Trigger an event if the keypad is used. You can load an example in the Arduino IDE.
[See File -> Examples -> Keypad -> Examples -> EventSerialKeypad] or see the KeypadEvent Example code.

7.9 For Now

Here's the list of multi-keypress functions and the keylist definition. I will complete their descriptions this weekend.

- Key key[LIST_MAX]
- bool getKeys()
- bool isPressed(char keyChar)
- int findInList(char keyChar)

---

8 Example

```
#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] = {5, 4, 3, 2}; //connect to the row pinouts of the keypad
byte colPins[COLS] = {8, 7, 6}; //connect to the column pinouts of the keypad

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

void setup(){
 Serial.begin(9600);
}

void loop(){
```

```
  char key = keypad.getKey();

  if (key != NO_KEY){
    Serial.println(key);
  }
}
```

[Get Code]

---

9 FAQ

- **How do I use multiple Keypads?**

Keypad is a class. Therefore to use multiple Keypad, you must create an instance for each of them. In the example above, the Keypad instance *keypad*) was bound to the digital pins 2, 3, 4, 5, 6, 7 and 8.

To add a Keypad bound to digital pins 9, 10, 11, 12, 13, 14, 15 and 16, you could create the following instance *keypad2*:

```
const byte ROWS2 = 4; //four rows
const byte COLS2 = 4; //four columns
char keys2[ROWS2][COLS2] = {
  {'.','a','d','1'},
  {'g','j','m','2'},
  {'p','t','w','3'},
  {'*',' ','#','4'}
};
byte rowPins2[ROWS2] = {12, 11, 10, 9}; //connect to the row pinouts of the
keypad
byte colPins2[COLS2] = {16, 15, 14, 13}; //connect to the column pinouts of
the keypad

Keypad keypad2 = Keypad( makeKeymap(keys2), rowPins2, colPins2, ROWS2, COLS2
);
```

And now it's just a matter of using whatever function is wanted on each keypad:

```
//update instances and possibly fire funcitons
void loop(){
  char key1 = keypad.getKey();
  char key2 = keypad2.getKey();

  if (key1 != NO_KEY || key2 != NO_KEY){
    Serial.print("You pressed: ");
    Serial.print(key1 != NO_KEY ? key1 : "nothing on keypad");
        Serial.print(" and ");
    Serial.print(key2 != NO_KEY ? key2 : "nothing on keypad2");
```

```
    Serial.println(".");
  }
}
```

- **How do I use setDebounceTime(unsigned int time)?**

In Arduino follow the main menu from File-> Examples-> Keypad-> Examples-> DynamicKeypad. Once the sketch is open find setup() and there you will see:

```
void setup()  {
  Serial.begin(9600);
  digitalWrite(ledPin, HIGH);             // Turns the LED on.
  keypad.addEventListener(keypadEvent);  // Add an event listener.
  keypad.setHoldTime(500);                // Default is 1000mS
  keypad.setDebounceTime(250);            // Default is 50mS
}
```

This shows that the debounce time will allow one key press every 250 milliseconds. If multiple key presses occur within that time frame (as would happen when a key is bouncing) then those extra presses are simply ignored.

---

10 Information about this page

Part of AlphaBeta Libraries.

Last Modified: September 17, 2015, at 08:08 PM

By: Christopher Andrews

10.1.1 Functions

void begin(makeKeymap(userKeymap))

Initializes the internal keymap to be equal to userKeymap
[See File -> Examples -> Keypad -> Examples -> CustomKeypad]

char waitForKey()

This function will wait forever until someone presses a key. **Warning:** It blocks all other code until a key is pressed. That means no blinking LED's, no LCD screen updates, no nothing with the exception of interrupt routines.

char getKey()

Returns the key that is pressed, if any. This function is non-blocking.

KeyState getState()

Returns the current state of any of the keys.
The four states are IDLE, PRESSED, RELEASED and HOLD.

boolean keyStateChanged()

New in version 2.0: Let's you know when the key has changed from one state to another. For example, instead of just testing for a valid key you can test for when a key was pressed.

setHoldTime(unsigned int time)

Set the amount of milliseconds the user will have to hold a button until the HOLD state is triggered.

setDebounceTime(unsigned int time)

Set the amount of milliseconds the keypad will wait until it accepts a new keypress/keyEvent. This is the "time delay" debounce method.

addEventListener(keypadEvent)

Trigger an event if the keypad is used. You can load an example in the Arduino IDE.
[See File -> Examples -> Keypad -> Examples -> EventSerialKeypad] or see the KeypadEvent Example code.

For Now

Here's the list of multi-keypress functions and the keylist definition.

- Key key[LIST_MAX]
- bool getKeys()
- bool isPressed(char keyChar)
- int findInList(char keyChar)