```
/* iLoopino 1.00 - May 2016

 * Created by Armando Colangleo aka 4Knobs Effects  - All Rights Reserved!!

 * You may use, distribute and modify this code for DIY and personal use ONLY.

 * Permission is DENIED for commercial use and / or profit.

 * Most of this code is based on codes found on the Net and libraries, though the assembling

 * of codes in a "structured programm" as well most of core part, has been done by me, with a lot of time effort, and lots of help

 * from many good guys all over the World, specially the one at arduino.cc.

 * So, if you're going to use/modify this code would you please visit my guitar effects page and "like" it :) ;) :) ;)

 * here's the link to my page https://www.facebook.com/4kfx2/?ref=bookmarks

 * and, may be, leave a message if you pass by.

 * For any question you can contact me on FB https://www.facebook.com/armando.colangelo.5 */

#include "OneButton.h"

#include "Bounce2.h"

#include <EEPROM.h>

#define DEBOUNCE 5  // how many ms to debounce, 5+ ms is usually plenty

#define Trigger_Button     7  // TRIGGER BUTTON

#define Trigger_LED  6  //TRIGGER LED

#define mutePin 14

//#define NUMBUTTONS sizeof(buttons)

byte h = 0, v = 0; //variables used in for loops

const unsigned long period = 50; //little period used to prevent error

unsigned long kdelay = 0; // variable used in non-blocking delay

const byte rows = 2; //number of rows of keypad

const byte columns = 3; //number of columnss of keypad
```

```
const byte Output[rows] = {16, 15}; //array of pins used as output for rows of keypad

const byte Input[columns] = {19, 18, 17}; //array of pins used as input for columnss of keypad

byte buttons[] = {19, 18, 17, 16, 15};

int whichPreset;

int beatdelay = 1200;

int savedPreset = 0;

int event = 0;

int Dmute = 50;

int Preseton;

int WPreseton;

//int changeLoop = 0;

int sumofButtonspressed;

int oldSwitchSumButton = 0;

int relayPin[] = {5, 4, 3, 2, 1, 0};

int ledPin[] = {8, 9, 10, 11, 12, 13};

int triggerDelay = 0;

int SwitchSumButton = 0;

int sumSingle = 0;

int StateofbuttonSingle = 0;

int i = 0;

//int oldquale = 0;

int oldsingleButtonStatus = 0;

int veryBeginning = 1;

int singleButtonStatus = 0;

int debounce = 20; // ms debounce period to prevent flickering when pressing or releasing the button

int DCgap = 250; // max ms between clicks for a double click event
```

```
int holdTime = 2000; // ms hold period: how long to wait for press+hold event

int longHoldTime = 3000; // ms long hold period: how long to wait for press+hold event

int ExtralongHoldTime = 4000; // ms long hold period: how long to wait for press+hold
event

const int releDelay = 470; //delay before and after mute-unmute

boolean state; /*initial state of each key = false*/

//determine how big the array up above is, by checking the size

//track if a button is just pressed, just released, or 'currently pressed'

//byte pressed[NUMBUTTONS], justpressed[NUMBUTTONS], justreleased[NUMBUTTONS];

//byte previous_keystate[NUMBUTTONS], current_keystate[NUMBUTTONS];

boolean buttonVal = HIGH; // value read from button

boolean buttonLast = HIGH; // buffered value of the button's previous state

boolean DCwaiting = false; // whether we're waiting for a double click (down)

boolean DConUp = false; // whether to register a double click on next release, or whether to
wait and click

boolean singleOK = true; // whether it's OK to do a single click

long downTime = -1; // time the button was pressed down

long upTime = -1; // time the button was released

boolean ignoreUp = false; // whether to ignore the button release because the click+hold
was triggered

boolean waitForUp = false; // when held, whether to wait for the up event

boolean holdEventPast = false; // whether or not the hold event happened already

boolean longHoldEventPast = false;// whether or not the long hold event happened already

boolean ExtralongHoldEventPast = false;

void setup() {

//  writetempEprom();// REMOVE BEFORE FINAL PROJECT

  pinMode(Trigger_Button, INPUT_PULLUP);

  pinMode(Trigger_LED, OUTPUT);
```

```
  for (i = 0; i < 6; i++)  {
    pinMode(relayPin[i], OUTPUT);
    pinMode(ledPin[i], OUTPUT);
  }
  for (i = 0; i < 6; i++) {
    pinMode(buttons[i], INPUT_PULLUP);
    digitalWrite(buttons[i], HIGH);
  }
  pinMode(mutePin, OUTPUT);// MUTE Pin
  for (byte i = 0; i < rows; i++)   { //for loop used to make pin mode of outputs as output
    pinMode(Output[i], OUTPUT);
  }
  for (byte s = 0; s < columns; s++) {//for loop used to make pin mode of inputs as
inputpullup
    pinMode(Input[s], INPUT_PULLUP);
  }
  startupBlinking();
}
void loop() {
  //delay(300);
  int singleButtonStatus = checkButton();
  if (oldsingleButtonStatus == 4 && singleButtonStatus == 1) { // upon exit 3-save turns off all
LEDs/relays then recall saved preset
    delay(Dmute);
    mute();
    AllPresetLedsOff();
    AllPresetRelaysOff();
    delay(600);
```

```
  SwitchSumButton = 10 + savedPreset;
  SwitchCore();
  delay(Dmute);
  unmute();
 }
 if (oldsingleButtonStatus == 5 && singleButtonStatus == 1) { // upon exit 4-bypass turns off all LEDs/relays then recall last active loop
  delay(Dmute);
  mute();
  AllPresetLedsOff();
  AllPresetRelaysOff();
  delay(600);
  SwitchSumButton = EEPROM.read(150);
  SwitchCore();
  delay(Dmute);
  unmute();
 }
 if (oldsingleButtonStatus != singleButtonStatus && singleButtonStatus != 0)
 {
  oldsingleButtonStatus = singleButtonStatus;
  if (oldsingleButtonStatus == 3)  // 3 = build loop -
  {
   //     changeLoop = 1;
    buildtempLoop(); // check rele status of selected preset, if rele is on turn on the respective LED, otherwise off
  }
  else
  {
```

```
    //      changeLoop = 0;
   }
   StateofbuttonSingle = oldsingleButtonStatus;
   triggerDelay = 380 - (oldsingleButtonStatus * 60); // 3 different delay time based on single button
   if (oldsingleButtonStatus == 1) { // return to liveMode, presets can be selected
     if (veryBeginning != 0) {
        blinkLedfortesting();
        greenSingleledoff();
        triggerDelay = 80;
        redSingleledblink();
        redSingleledon();
       }
        redSingleledon();
     if (veryBeginning == 0) {
       SwitchSumButton = EEPROM.read(150); // at startup load last used preset before turn off
       SwitchCore();
     }
   }
   if (oldsingleButtonStatus == 3) {//modify preset
     StateofbuttonSingle = 2;
     redSingleledoff();
     greenSingleledblink();
     greenSingleledon();
   }
   if (oldsingleButtonStatus == 4 ) {//save preset
     StateofbuttonSingle = 3;
```

```
    greenSingleledblink();
    greenSingleledon();
   }
   if (oldsingleButtonStatus == 5) {//bypass
    beatdelay = 1200;
    StateofbuttonSingle = 4;
    triggerDelay = 80;
    for (int i = 0; i < 12; i++)
    {
     AllPresetLedsOn();
     delay(50);
     AllPresetLedsOff();
     delay(50);
    }
    byPassAll();
   }
   sumSingle = StateofbuttonSingle * 10; // sumsingle is needed to build a case for switch
   //    oldquale = 0;
  }
/*  if (oldsingleButtonStatus == 5)  //alternate blinking *** noisy ***
  {
   blinkledinByPassMode();
  }*/
  if (oldsingleButtonStatus == 3 || oldsingleButtonStatus == 4 )// rimuovere
oldsingleButtonStatus == 4 se difficoltosa la selezione del preset da salvare
  {
   beatdelay = 70;
```

```
  whichPreset = EEPROM.read(150) - 11;
  blinkledpresetmod(whichPreset);
 }
 if (millis() - kdelay > period) //used to make non-blocking delay
 {
  kdelay = millis(); //capture time from millis function
  switch (keypad()) //switch used to specify which button
  {
    case 0://1
      SwitchSumButton = sumSingle + 1; // sum single+1-6 buttons to build a variable for case switch
      break;
    case 1://2
      SwitchSumButton = sumSingle + 2;
      break;
    case 2://3
      SwitchSumButton = sumSingle + 3;
      break;
    case 4://4
      SwitchSumButton = sumSingle + 4;
      break;
    case 5://5
      SwitchSumButton = sumSingle + 5;
      break;
    case 6://6
      SwitchSumButton = sumSingle + 6;
      break;
```

```
    default:
      ;
   }
  }
  if ((oldSwitchSumButton != SwitchSumButton) && (SwitchSumButton < 21 || (SwitchSumButton > 26 && SwitchSumButton < 41)))
  {
   oldSwitchSumButton = SwitchSumButton;
   SwitchCore();
  }
  if ((oldSwitchSumButton != SwitchSumButton) && (SwitchSumButton > 20 && SwitchSumButton < 27) || (SwitchSumButton > 40 && SwitchSumButton < 47))
  {
   if (millis() - kdelay > period) //used to make non-blocking delay
   {
    kdelay = millis(); //capture time from millis function
    switch (keypad()) //switch used to specify which button
    {
     case 0://1
      SwitchSumButton = sumSingle + 1; // sum single+1-6 buttons to build a variable for case switch
      break;
     case 1://2
      SwitchSumButton = sumSingle + 2;
      break;
     case 2://3
      SwitchSumButton = sumSingle + 3;
      break;
```

```
        case 4://4
          SwitchSumButton = sumSingle + 4;
          break;
        case 5://5
          SwitchSumButton = sumSingle + 5;
          break;
        case 6://6
          SwitchSumButton = sumSingle + 6;
          break;
        default:
          ;
      }
    }
    SwitchCore();
    if (SwitchSumButton > 20 && SwitchSumButton < 27) {
      SwitchSumButton = 0;
    }
    if (SwitchSumButton > 40 && SwitchSumButton < 47) {
      SwitchSumButton = 0;
    }
  }
} //END void loop()
int checkButton() // check single button for single/double/hold
{
  event = 0;
  if (veryBeginning == 0)
  {
```

```
  buttonVal = digitalRead(Trigger_Button);    // Read the state of the button

  if (buttonVal == LOW && buttonLast == HIGH && (millis() - upTime) > debounce) {    // Button pressed down

    downTime = millis();

    ignoreUp = false;

    waitForUp = false;

    singleOK = true;

    holdEventPast = false;

    longHoldEventPast = false;

    ExtralongHoldEventPast = false;

    if ((millis() - upTime) < DCgap && DConUp == false && DCwaiting == true) DConUp = true;

    else DConUp = false;

    DCwaiting = false;

  }

  else if (buttonVal == HIGH && buttonLast == LOW && (millis() - downTime) > debounce) {    // Button released

    if (not ignoreUp) {

      upTime = millis();

      if (DConUp == false) DCwaiting = true;

      else {

        event = 2;

        DConUp = false;

        DCwaiting = false;

        singleOK = false;

      }

    }

  }
```

```
  if ( buttonVal == HIGH && (millis() - upTime) >= DCgap && DCwaiting == true && DConUp
== false && singleOK == true ) {     // Test for normal click event: DCgap expired
    event = 1;
    DCwaiting = false;
  }
  if (buttonVal == LOW && (millis() - downTime) >= holdTime) {    // Test for hold
    if (not holdEventPast) {      // Trigger "normal" hold
      event = 3;
      waitForUp = true;
      ignoreUp = true;
      DConUp = false;
      DCwaiting = false;
      //downTime = millis();
      holdEventPast = true;
    }
    if ((millis() - downTime) >= longHoldTime) {      // Trigger "long" hold
      if (not longHoldEventPast) {
        event = 4;
        longHoldEventPast = true;
      }
    }
    if ((millis() - downTime) >= ExtralongHoldTime) {      // Trigger "extralong" hold
      if (not ExtralongHoldEventPast) {
        event = 5;
        ExtralongHoldEventPast = true;
      }
    }
```

```
    }
    buttonLast = buttonVal;
    return event;
  }
  if (veryBeginning == 1) {
    event = 1;
    veryBeginning = 0;
    greenSingleledon();
    return event;
  }
}
/*void check_switches()
{
  static byte previousstate[NUMBUTTONS];
  static byte currentstate[NUMBUTTONS];
  static long lasttime;
  byte index;
  if (millis() < lasttime) {    // we wrapped around, lets just try again
    lasttime = millis();
  }
  if ((lasttime + DEBOUNCE) > millis()) {    // not enough time has passed to debounce
    return;
  }
  lasttime = millis();  // ok we have waited DEBOUNCE milliseconds, lets reset the timer
  for (index = 0; index < NUMBUTTONS; index++) {
    justpressed[index] = 0;       //when we start, we clear out the "just" indicators
    justreleased[index] = 0;
```

```
   currentstate[index] = digitalRead(buttons[index]);   //read the button
   if (currentstate[index] == previousstate[index]) {
     if ((pressed[index] == LOW) && (currentstate[index] == LOW)) {       // just pressed
       justpressed[index] = 1;
     }
     else if ((pressed[index] == HIGH) && (currentstate[index] == HIGH)) {
       justreleased[index] = 1; // just released
     }
     pressed[index] = !currentstate[index];  //remember, digital HIGH means NOT pressed
   }
   previousstate[index] = currentstate[index]; //keep a running tally of the buttons
  }
}*/
/*byte thisSwitch_justPressed() {
  byte thisSwitch = 255;
  check_switches();  //check the switches &amp;amp; get the current state
  for (byte i = 0; i < NUMBUTTONS; i++) {
    current_keystate[i] = justpressed[i];
    if (current_keystate[i] != previous_keystate[i]) {
      if (current_keystate[i]) thisSwitch = i;
    }
    previous_keystate[i] = current_keystate[i];
  }
  return thisSwitch;
}*/
 void blinktriggerLed()
 {
```

```
  for (i = 0; i < 6; i++)
  {
    digitalWrite(Trigger_LED, HIGH);
    delay(Trigger_LED + triggerDelay);
    digitalWrite(Trigger_LED, LOW);
    delay(Trigger_LED + triggerDelay);
  }
}
void SwitchCore()
{
  int val = 0;
  switch (SwitchSumButton) // 1 or 2 or 3 = 1 Select Loop - 2 Build Loop - 3 Save loop - + 1 to 6 buttonpressed
  {
    /****************************** BYPASS MODE */
    case 41:
      bypassMode(0);
      break;
    case 42:
      bypassMode(1);
      break;
    case 43:
      bypassMode(2);
      break;
    case 44:
      bypassMode(3);
      break;
```

```
case 45:
  bypassMode(4);
  break;
case 46:
  bypassMode(5);
  break;
/****************************** BUILD PRESET MODE */
case 21:
  buildPresetMode(0);
  break;
case 22:
  buildPresetMode(1);
  break;
case 23:
  buildPresetMode(2);
  break;
case 24:
  buildPresetMode(3);
  break;
case 25:
  buildPresetMode(4);
  break;
case 26:
  buildPresetMode(5);
  break;
/****************************** STORE PRESET MODE */
case 31: //SaveLoop(int addr, int led)
```

```
  SaveLoop(11, ledPin[0]);
  break;
case 32:
  SaveLoop(21, ledPin[1]);
  break;
case 33:
  SaveLoop(31, ledPin[2]);
  break;
case 34:
  SaveLoop(41, ledPin[3]);
  break;
case 35:
  SaveLoop(51, ledPin[4]);
  break;
case 36:
  SaveLoop(61, ledPin[5]);
  break;
/****************************** READ PRESET MODE */
case 11: //(int addr, int pcNum, int led)
  LiveModePreset(11, 1, 0);
  EEPROM.write((150), 11);     //save preset in use, at startup this preset will be selected
  break;
case 12:
  LiveModePreset(21, 2, 1);
  EEPROM.write((150), 12);
  break;
case 13:
```

```
    LiveModePreset(31, 3, 2);
    EEPROM.write((150), 13);
    break;
   case 14:
    LiveModePreset(41, 4, 3);
    EEPROM.write((150), 14);
    break;
   case 15:
    LiveModePreset(51, 5, 4);
    EEPROM.write((150), 15);
    break;
   case 16:
    LiveModePreset(61, 6, 5);
    EEPROM.write((150), 16);
    break;
  }
}
void buildtempLoop() {
  mute();
  delay(Dmute);
  for (int q = 0 ; q < 6 ; q++)
  {
   if (digitalRead(relayPin[q]) == HIGH) {        // check if the relay is HIGH
    digitalWrite(ledPin[q], HIGH);  // turn LED ON
   } else {
    digitalWrite(ledPin[q], LOW);  // else turn LED OFF
   }
```

```
  }
  unmute();
  delay(Dmute);
}
void BuildLoop(int relay) //old version of switch 21/26 - not used anymore left here "just in case"
{
  digitalWrite(relayPin[relay], ~state);
  digitalWrite(ledPin[relay], ~state);
  //  delay(Dmute);
  digitalWrite(relayPin[relay], state);
  digitalWrite(ledPin[relay], state);
}
void LiveModePreset(int addr, int pcNum, int led) //k-o
{
  mute();
  delay(Dmute);
  for (int i = 0; i < 6; i++)
  {
    digitalWrite(relayPin[i], EEPROM.read((addr) + i));
    digitalWrite(ledPin[i], LOW);
    digitalWrite(ledPin[led], HIGH);
    delay(10);
  }
  delay(Dmute);
  unmute();
}
```

```
void SaveLoop(int addr, int led)
{
  //please contact the author to get these part of the sketch. Sorry ;)
}
void AllPresetLedsOn() {
  for (int i = 0 ; i < 6 ; i ++)
  {
    digitalWrite(ledPin[i], HIGH);
  }
}
void AllPresetLedsOff() {
  for (int i = 0 ; i < 6 ; i ++)
  {
    digitalWrite(ledPin[i], LOW);
  }
}
void AllPresetRelaysOff() {
  for (int i = 0 ; i < 6 ; i ++)
  {
    digitalWrite(relayPin[i], LOW);
  }
}
void AllPresetRelaysOn() {
  for (int i = 0 ; i < 6 ; i ++)
  {
    digitalWrite(relayPin[i], HIGH);
  }
```

```
}
void startupBlinking() {
  mute();
  delay(Dmute);
  for (int q = 0 ; q < 2 ; q++)
  {
    for (int i = 6 ; i > 0 ; i --)  {
      digitalWrite(ledPin[i - 1], HIGH);
      delay(100);
    }
    for (int i = 0 ; i < 6 ; i ++)  {
      digitalWrite(ledPin[i], LOW);
      delay(100);
    }
  }
  for (int i = 0 ; i < 6; i++)
  {
    digitalWrite(relayPin[i], !digitalRead(relayPin[i]));
    delay(200);
  }
  for (int i = 0; i < 4; i++)
  {
    AllPresetLedsOn();
    delay(100);
    AllPresetLedsOff();
    delay(100);
  }
```

```
 AllPresetRelaysOff();
 for (int i = 0 ; i < 2; i++)
 {
  digitalWrite(Trigger_LED, !digitalRead(Trigger_LED));
  delay(200);
 }
 unmute();
 delay(Dmute);
 delay(600);
}
void writetempEprom() { //delete before re lease final product
 for (int i = 11 ; i < 67; i++)
 {
  EEPROM.write((i), 0);
 }
 EEPROM.write((11), 0);
 EEPROM.write((12), 1);
 EEPROM.write((13), 0);
 EEPROM.write((14), 0);
 EEPROM.write((15), 1);
 EEPROM.write((16), 0);

 EEPROM.write((21), 0);
 EEPROM.write((22), 0);
 EEPROM.write((23), 1);
 EEPROM.write((24), 1);
 EEPROM.write((25), 0);
```

```
EEPROM.write((26), 0);


EEPROM.write((31), 0);
EEPROM.write((32), 0);
EEPROM.write((33), 0);
EEPROM.write((34), 0);
EEPROM.write((35), 1);
EEPROM.write((36), 0);


EEPROM.write((41), 0);
EEPROM.write((42), 0);
EEPROM.write((43), 1);
EEPROM.write((44), 0);
EEPROM.write((45), 0);
EEPROM.write((46), 0);


EEPROM.write((51), 0);
EEPROM.write((52), 1);
EEPROM.write((53), 1);
EEPROM.write((54), 1);
EEPROM.write((55), 0);
EEPROM.write((56), 0);


EEPROM.write((61), 0);
EEPROM.write((62), 0);
EEPROM.write((63), 0);
EEPROM.write((64), 1);
```

```
  EEPROM.write((65), 0);
  EEPROM.write((66), 0);
}
void greenSingleledon() {
  pinMode(Trigger_LED, OUTPUT);
  digitalWrite(Trigger_LED, HIGH);   // turn the LED on (HIGH is the voltage level)
}
void greenSingleledoff() {
  pinMode(Trigger_LED, INPUT);    // turn the LED off by making the voltage LOW
  delay(100);           // wait for a 100th second
}
void redSingleledon() {
  pinMode(Trigger_LED, OUTPUT);
  digitalWrite(Trigger_LED, LOW);   // turn the LED on (HIGH is the voltage level)
}
void redSingleledoff() {
  pinMode(Trigger_LED, INPUT);    // turn the LED off by making the voltage LOW
  delay(100);           // wait for a 100th second
}
void redSingleledblink() {
  for (i = 0 ; i < 6; i++)
  {
    redSingleledon();
    delay(triggerDelay);
    redSingleledoff();
  }
}
```

```
void greenSingleledblink() {
  for (i = 0 ; i < 6; i++)
  {
    greenSingleledon();
    delay(triggerDelay);
    greenSingleledoff();
  }
}
void mute()
{
  digitalWrite(mutePin, HIGH);
}
void unmute()
{
  digitalWrite(mutePin, LOW);
}
void byPassAll()
{
  delay(Dmute);
  mute();
  for (i = 0; i < 6; i++)
  {
    WPreseton = digitalRead(ledPin[i]);
    if (WPreseton == HIGH)
    {
      Preseton = i ;
      digitalWrite(ledPin[i], LOW);
```

```
    }
  }
  AllPresetRelaysOn();
  AllPresetRelaysOff();
  unmute();
  delay(Dmute);
  delay(500);
}
byte keypad() // function used to detect which button is used
{
  static bool no_press_flag = 0; //static flag used to ensure no button is pressed
  for (byte x = 0; x < columns; x++) // for loop used to read all inputs of keypad to ensure no button is pressed
  {
    if (digitalRead(Input[x]) == HIGH); //read evry input if high continue else break;
    else
      break;
    if (x == (columns - 1)) //if no button is pressed
    {
      no_press_flag = 1;
      h = 0;
      v = 0;
    }
  }
  if (no_press_flag == 1) //if no button is pressed
  {
    for (byte r = 0; r < rows; r++) //for loop used to make all output as low
```

```
    digitalWrite(Output[r], LOW);
  for (h = 0; h < columns; h++) // for loop to check if one of inputs is low
  {
    if (digitalRead(Input[h]) == HIGH) //if specific input is remain high (no press on it)
continue
      continue;
    else //if one of inputs is low
    {
      for (v = 0; v < rows; v++) //for loop used to specify the number of row
      {
        digitalWrite(Output[v], HIGH); //make specified output as HIGH
        if (digitalRead(Input[h]) == HIGH) //if the input that selected from first sor loop is
change to high
        {
          no_press_flag = 0; //reset the no press flag;
          for (byte w = 0; w < rows; w++) // make all outputs as low
            digitalWrite(Output[w], LOW);
          return v * 4 + h; //return number of button
        }
      }
    }
  }
 }
 return 50;
}
void bypassMode(int led)
{
 mute();
```

```
  delay(Dmute);
  digitalWrite(relayPin[led], !digitalRead(relayPin[led]));
  digitalWrite(ledPin[led], !digitalRead(ledPin[led]));
  delay(Dmute);
  unmute();
}
void buildPresetMode(int led)
{
  mute();
  delay(Dmute);
  digitalWrite(relayPin[led], !digitalRead(relayPin[led]));
  digitalWrite(ledPin[led], !digitalRead(ledPin[led]));
  delay(Dmute);
  unmute();
}
void blinkledinByPassMode() {
  static byte heartState = 1;
  static unsigned long previousBeat = millis();
  unsigned long currentMillis = millis();
  if (currentMillis - previousBeat >= 1200) {
    previousBeat = currentMillis;
    heartState ^= 1;
    //                digitalWrite(13, heartState);
//    pinMode(Trigger_LED, OUTPUT);
    digitalWrite(Trigger_LED, heartState);   // turn the LED on (HIGH is the voltage level)
  }
}
```

```
void blinkLedfortesting()
{
 for (int q = 0 ; q < 2 ; q++)
 {
  for (int i = 6 ; i > 0 ; i --)  {
   digitalWrite(ledPin[i - 1], HIGH);
   delay(100);
  }
  for (int i = 0 ; i < 6 ; i ++)  {
   digitalWrite(ledPin[i], LOW);
   delay(100);
  }
 }
}
void blinkledpresetmod(int Pled) { // blinks led of preset selected to be modified
 static byte heartState = 1;
 static unsigned long previousBeat = millis();
 unsigned long currentMillis = millis();
 if (currentMillis - previousBeat >= beatdelay) {
  previousBeat = currentMillis;
  heartState ^= 1;
  //                digitalWrite(13, heartState);
  //   pinMode(Trigger_LED, OUTPUT);
  digitalWrite(ledPin[Pled], heartState);   // turn the LED on (HIGH is the voltage level)
 }
}
```