```
#include "OLED_I2C_128x64_Monochrome.h"
#include "OLED_I2C_128x64_Monochrome_Font.h"
#include <Wire.h>
#include <avr/pgmspace.h>

#define OLED_I2C_128x64_Monochrome Display

void Display::init(boolean regulator)
{
      Wire.begin();
      // upgrade to 400KHz! (only use it when all other i2c devices support that
speed)
      if (I2C_400KHZ)
      {
            byte twbrbackup = TWBR;
            TWBR = 12;
      }

      if (regulator)
      {
      sendCommand(COMMAND_CHARGE_PUMP_SETTING);
            sendCommand(COMMAND_CHARGE_PUMP_ENABLE);
      }

      font = 0;
      fontSize = 1;
      inv = 0;
      scroll = false;

    setDisplayOff();
    setBlackBackground();
      setPageMode();
      clear();
    setDisplayOn();
}

void Display::sendCommand(byte command)
{
      Wire.beginTransmission(OLED_ADDRESS); // begin transmitting
      Wire.write(COMMAND_MODE); // data mode
      Wire.write(command); // send command
      Wire.endTransmission(); // stop transmitting
}

void Display::sendData(byte data)
{
      Wire.beginTransmission(OLED_ADDRESS);
      Wire.write(OLED_DATA_MODE);
      Wire.write(data);
      Wire.endTransmission();
}

void Display::printChar(char pChar, byte posX, byte posY)
{
      /*
            Changes in printChar are made by untergeekDE, and are only adapted
by me: https://github.com/untergeekDE/OLED_I2C_128x64_Monochrome_Library
            All kudos to him!
      */

      // Ignore unused ASCCII characters
      if (pChar < 32 || pChar > 127)
    {
            pChar = '?'; // ?: characters that can't be displayed
```

```
    }

      if (posX < OLED_Max_X)
      {
            setCursor(posX, posY);
      }

    for(byte i = 0; i < 8; i++)
    {
      switch(font)
      {
            case 0:
                  sendData(pgm_read_byte(&basicFont[pChar - 32][i]));
                  break;
            case 1:
                  sendData(pgm_read_byte(&boldFont[pChar][i]));
                  break;
            default:
                  sendData(pgm_read_byte(&basicFont[pChar - 32][i]));
      }
    }// end for

      return;

      px = posX % OLED_Max_X;
      py = posY % OLED_Max_Y;
      setCursor(px, py);

      if(pChar == '\n')
      {
            px  = 0; // Reset x to zero,
            py += fontSize; // advance y one line
      }
      else
      {
        // Scale x1: 8x8
            setCursor(px, py);
            for(byte i = 0; i < 8; i++)
                  {
                        sendData(pgm_read_byte(&basicFont[pChar][i]) ^ inv);
                  }


            px += fontSize;
            if (px >= OLED_Max_X)
            {
                  px = 0;
                  py += fontSize;
                  if (py >= OLED_Max_Y)
                  {
                        if (scroll)
                              py = OLED_Max_Y-fontSize;
                        else
                              py = 0;
                  }
            }
      } // End else
} //End function

void Display::printString(const char *data, byte posX, byte posY)
{
      byte i = 0;

      if (posX < OLED_Max_X)
```

```
      {
            setCursor(posX, posY);
      }

    while(data[i] && i < OLED_Max_X)
    {
            printChar(data[i++]);
      }
}


void Display::drawBitmap(const byte *bitmapArray, byte posX, byte posY, byte
width, byte height)
{
      // max width = 16
      // max height = 8
      setCursor(posX, posY);

      byte Column = 0;
      for(int i = 0; i < width * 8 * height; i++)
      {
            sendData(pgm_read_byte(&bitmapArray[i]));
            if (++Column == width * 8)
            {
                  Column = 0;
                  setCursor(posX, ++posY);
            }
      }
}

void Display::setCursor(byte posX, byte posY)
{
      // Y - 1 unit = 1 page (8 pixel rows)
      // X - 1 unit = 8 pixel columns
    sendCommand(0x00 + (8 * posX & 0x0F)); // set column lower address
    sendCommand(0x10 + ((8 * posX >> 4) & 0x0F)); // set column higher address
      sendCommand(0xB0 + posY); // set page address
}

void Display::clear()
{
      for(byte i = 0; i < 8; i++)
      {
            setCursor(0, i);
            for(byte j = 0; j < 128; j++) // clear all columns
            {
                  sendData(0);
            }
      }
      setCursor(0, 0);
}

void Display::setWhiteBackground()
{
      sendCommand(COMMAND_WHITE_BACKGROUND);
}

void Display::setBlackBackground()
{
      sendCommand(COMMAND_BLACK_BACKGROUND);
}

void Display::setDisplayOff()
{
```

```
	sendCommand(COMMAND_DISPLAY_OFF);
}

void Display::setDisplayOn()
{
	sendCommand(COMMAND_DISPLAY_ON);
}

void Display::setBrightness(byte brightness)
{
	sendCommand(COMMAND_SET_BRIGHTNESS);
	sendCommand(brightness);
}

void Display::setPageMode()
{
	addressingMode = PAGE_ADDRESSING;
	sendCommand(0x20); //set addressing mode
	sendCommand(PAGE_ADDRESSING); //set page addressing mode
}

void Display::setHorizontalMode()
{
	addressingMode = HORIZONTAL_ADDRESSING;
	sendCommand(0x20); // set addressing mode
	sendCommand(HORIZONTAL_ADDRESSING); // set page addressing mode
}

void Display::setFont(byte type)
{
	font = type;
}

void Display::setFontSize(byte size)
{
	// fontSize adapted from untergeekDE:
https://github.com/untergeekDE/OLED_I2C_128x64_Monochrome_Library
	// All kudos to him!
	fontSize = size;
}


Display lcd;
```