Практическая работа 5

Проект Ночной светильник

В этом эксперименте светодиод должен включаться при падении уровня освещенности ниже порога, заданного потенциометром.

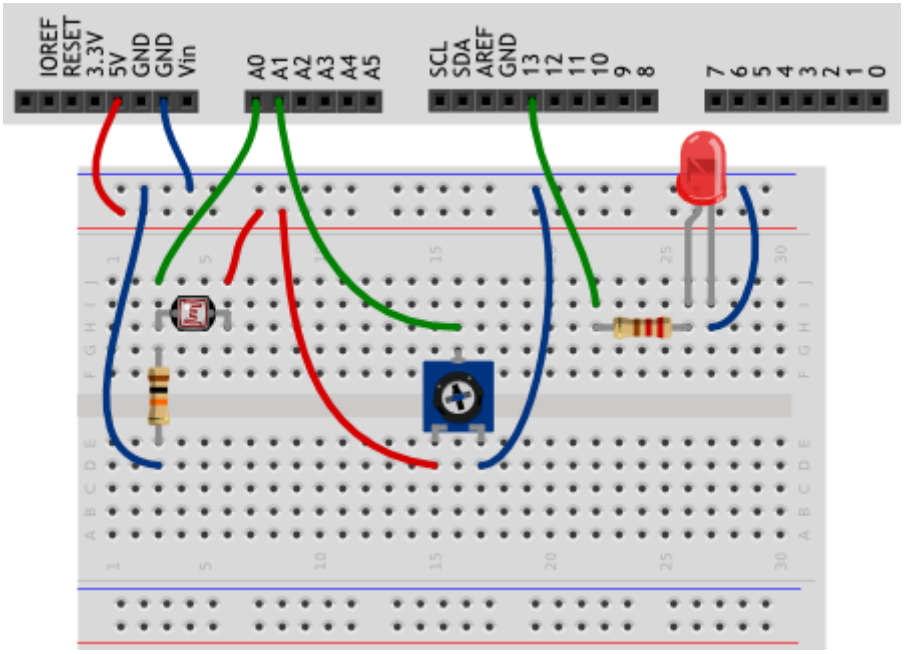
Задание 1. Ответить на вопросы

1. Что такое фоторезистор?
2. Как работает потенциометр?

Задание 2. Список деталей для эксперимента

1. 1 плата Arduino Uno
2. 1 беспаечная макетная плата
3. 1 светодиод
4. 1 резистор номиналом 10 Ком
5. 1 резистор 220 Ом
6. 10 проводов «папа-папа»
7. Фоторезистор

Схема на макетной плате:



1. Зарисуйте принципиальную схему установки.

Обратите внимание

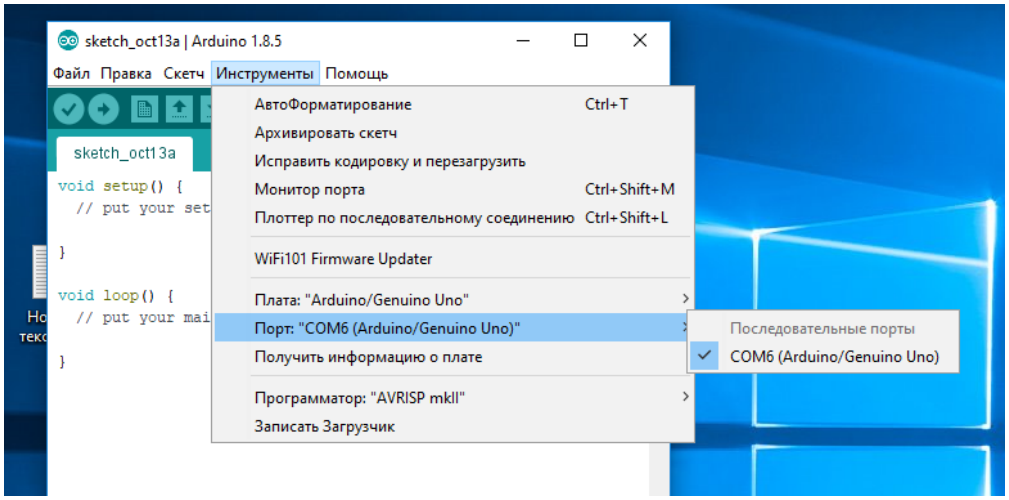
✓ В этом эксперименте мы устанавливаем фоторезистор между питанием и аналоговым входом, т.е. в позицию R1 в схеме делителя напряжения. Это нам нужно для того, чтобы при уменьшении освещенности мы получали меньшее напряжение на аналоговом входе.

✓ Постарайтесь разместить компоненты так, чтобы светодиод не засвечивал фоторезистор.

Задание 3. Программирование микроконтроллера

1. Запустите приложение

2. Убедитесь, что выбран нужный порт



Логические выражения и ветвление

Для работы с целыми числами в C/C++ существует тип переменных int. Но операции над целыми числами — не единственный инструмент в арсенале разработчика. Одними из важнейших элементов в программировании являются логические операции, выражения и типы данных.

В то время как переменные типа int могут хранить произвольные целые числа, а операции над ними подчиняются законам целочисленной арифметики; логические переменные могут хранить лишь одно из двух значений: истину или ложь, а операции над ними подчиняются законам алгебры логики.

Алгебра логики — умное название для интуитивно понятных вещей: операций «и», «или», «не» над высказываниями, которые либо справедливы, либо нет. Например, рассмотрим такое логическое выражение: «в коридоре темно и по коридору идёт человек». Оно состоит из двух логических значений (утверждения про темноту и человека) и одного оператора (союз «и»). Его итоговым значением также будет логическое значение, которое можно использовать, скажем, как сигнал для включения света в коридоре гостиницы. В программировании переменные, которые хранят логические значения называются булевыми переменными, а операторы, которые производят над ними действия называют булевыми операторамиили просто логическими операторами.

Логические операции в C++

В C++ для булевых переменных существует тип bool. Для обозначения истины в C++ используется слово true, а для обозначения лжи — слово false. Таким образом, если для объявления целочисленных переменных мы используем:

```
int myValue = 42;
```

Для логических переменных, используется:

```
bool tooDark = false;

bool humanDetected = true;
```

Поскольку C++ для Arduino — это некоторая надстройка над голым C++, для обозначения логического типа наряду с bool существует ещё слово boolean. Это абсолютные синонимы. Можете использовать и то и другое. Просто из соображений единого стиля и повышения читаемости кода выберите один из терминов и используйте его всюду. Мы в статье будем использовать bool, просто потому что в нём меньше букв и он входит в стандарт языка.

Применение на практике: экологичный отель

Давайте разовьём тему с энергосберегающим освещением и сделаем на Arduino устройство, которое включает свет в коридоре только тогда, когда это действительно необходимо.

Допустим, к Arduino подключён аналоговый датчик уровня освещённости, пироэлектрический цифровой датчик движения тёплых объектов и экологичная светодиодная лампа.

Тогда скетч будет выглядеть следующим образом:

```
#define LIGHT_SENSOR_PIN    A0
#define MOTION_SENSOR_PIN   2
#define LED_LAMP_PIN        5

#define LIGHT_LEVEL_THRESHOLD   600

void setup()
{
    pinMode(LIGHT_SENSOR_PIN, INPUT);
    pinMode(MOTION_SENSOR_PIN, INPUT);
    pinMode(LED_LAMP_PIN, OUTPUT);
}

void loop()
{
    int lightLevel = analogRead(LIGHT_SENSOR_PIN);
    bool motionDetected = digitalRead(MOTION_SENSOR_PIN);
    bool tooDark = lightLevel < LIGHT_LEVEL_THRESHOLD;
    bool lightningRequired = tooDark && motionDetected;
    digitalWrite(LED_LAMP_PIN, lightningRequired ? HIGH : LOW);
}
```

Взглянем на loop и поймём что здесь происходит. С первой строкой всё понятно: мы считываем значение освещённости с аналогового сенсора с

помощью встроенной функции analogRead и присваиваем его переменной с именем lightLevel.

Далее мы объявляем логическую переменную motionDetected в качестве значения которой присваиваем результат вызова встроенной функции digitalRead. Функция digitalRead похожа по своей сути на analogRead, но может возвращать лишь одно из двух значений: либо истину (true), либо ложь (false). То есть это функция, которая возвращает логическое значение. Она подходит для считывания показаний разнообразных бинарных цифровых датчиков. В нашем примере мы как раз использовали такой: пироэлектрический сенсор, который выдаёт 0 вольт, пока движения в его радиусе видимости нет и 5 вольт, когда замечено перемещение тёплого объекта: человека, кошки или кого-то ещё. Нулю вольт микроконтроллер ставит в соответствие значение false, а пяти вольтам — true.

Итак, по итогам исполнения второй строки, в переменной motionDetected будет храниться либо истина, либо ложь в зависимости от того замечено ли движение в коридоре.

Далее мы видим определение булевой переменной tooDark, которой в качестве значения присваивается значение выражения lightLevel < LIGHT_LEVEL_THRESHOLD. Символ <, как можно догадаться, в C++ означает оператор «меньше, чем». Из двух численных операндов, этот оператор делает один логический результат. Значение всего выражения считается истинным, если то, что записано слева от знака (lightLevel в нашем случае) меньше, чем то, что записано справа от знака (LIGHT_LEVEL_THRESHOLD). Довольно логично и интуитивно понятно, не правда ли?!

Таким образом в переменной tooDark окажется значение true, только если значение уровня освещённости lightLevel, полученное ранее, окажется меньше 600. В противном случае, в переменной окажется false.

Конкретное значение, вроде 600 в нашем случае, в подобных случаях часто получают экспериментально: в зависимости от используемого сенсора и конфигурации помещения, где стоит устройство.

Идём дальше. Мы видим объявление булевой переменной lightningRequired, в качестве значения которой присваивается значение выражения tooDark && motionDetected. Символ && в C++ означает оператор логического «и». Как можно догадаться, всё выражение считается истинным тогда и только тогда, когда и то, что справа, и то что слева от оператора истинно. Таким образом, переменная lightningRequired примет значение true, если в коридоре одновременно: и слишком темно, и замечено движение человека. Если не выполнено хоть одно из условий, результатом будет false. Как раз то, что нужно.

И наконец, последним выражением идёт вызов функции digitalWrite для пина Arduino, к которому подключена лампа. Нам нужно включить лампу, если переменная lightningRequired содержит истинное значение и выключить, если в ней хранится ложь. Чтобы сделать это, мы используем тернарный условный оператор ? :. Где в качестве условия используем просто значение переменнойlightningRequired. Помните? В тернарном операторе условие считается выполненным, если его значение — не ноль; и не выполненным если его значение — ноль.

На самом деле для процессора не существует понятия логических значений. Всё что он умеет — оперировать над целыми числами. Поэтому в C++ существует автоматическое преобразование типов. Там, где ожидается int, а мы используем bool, за кадром происходит автоматическое преобразование:true превращается в целое число 1, а false — в целое число 0.

Так что, булевы выражения и переменные — ни что иное, как удобство для программистов, синтаксический сахар, который позволяет писать программы более понятно и выразительно.

Возвращаясь к нашему примеру, в выражении:

```
digitalWrite(LED_LAMP_PIN, lightningRequired ? HIGH : LOW);
```

второй аргумент примет значение HIGH, если lightningRequired — это true; и LOW, если lightningRequired — это false. Таким образом, мы добились чего хотели: включения света, если он нужен и выключения, если он не уместен.

О краткости записи

Если вспомнить об автоматическом преобразовании переменных разных типов, а ещё о том, что HIGH — это ничто иное, как макроопределение числа 1, а LOW — макроопределение числа 0, последнюю строку в нашем примере можно сократить до лаконичного выражения:

```
digitalWrite(LED_LAMP_PIN, lightningRequired);
```

Мы обошлись без тернарного оператора: ведь всё равно функция digitalWrite получит:

- ✓ единицу: то же самое, что и HIGH, если lightningRequired будет true
- ✓ ноль: то же самое, что и LOW, если lightningRequired будет false

Кроме того, логические выражения — это самые обычные выражения в C++, к которым применяются общие правила встраивания. Поэтому весь код loop в примере с экологичным освещением на самом деле мог бы быть записан в одну строку:

```
void loop()
{
    digitalWrite(LED_LAMP_PIN, analogRead(LIGHT_SENSOR_PIN) < LIGHT_LEVEL_THRESHOLD &&
digitalRead(MOTION_SENSOR_PIN));
}
```

Да, код теперь находится на грани читаемости, но это всего лишь демонстрация возможностей. В реальной жизни лучше использовать пару промежуточных переменных:

```
void loop()
{
    bool tooDark = analogRead(LIGHT_SENSOR_PIN) < LIGHT_LEVEL_THRESHOLD;
    bool motionDetected = digitalRead(MOTION_SENSOR_PIN);
    digitalWrite(LED_LAMP_PIN, tooDark && motionDetected);
```

```
}
```

Или хотя бы использовать перенос строк для наглядности:

```
void loop()
{
    digitalWrite(LED_LAMP_PIN,
        analogRead(LIGHT_SENSOR_PIN) < LIGHT_LEVEL_THRESHOLD &&
        digitalRead(MOTION_SENSOR_PIN)
    );
}
```

А если свет таки нужен: условное выражение if

Внимательный читатель мог заметить, что приведённый скетч едва ли может работать в реальных условиях. Если мы включим свет, когда слишком темно и зафиксировано движение, при следующем же прогоне loop датчик «увидит» свет от нашей же лампы, программа посчитает, что и так достаточно светло и выключит лампу. Процесс постоянного выключения и включения так и будет продолжаться пока пироэлектрический датчик будет фиксировать движение. В лучшем случае мы получим свечение лампы в пол силы, в худшем — раздражающее мерцание. Как быть?

Можно после включения лампы усыплять программу на, скажем, 30 секунд. Вполне достаточно, чтобы дать постояльцу отеля пройти коридор и не так уж расточительно с точки зрения экономии электроэнергии.

Как сделать так, чтобы задержка на 30 секунд производилась только после включения лампы, но не после выключения? Для этого в C++ существует условное выражение «if». Используя его, скетч может выглядеть следующим образом:

```
#define LIGHT_SENSOR_PIN    A0
#define MOTION_SENSOR_PIN   2
#define LED_LAMP_PIN        5

#define LIGHT_LEVEL_THRESHOLD   600
```

```
void setup()
{
    pinMode(LIGHT_SENSOR_PIN, INPUT);
    pinMode(MOTION_SENSOR_PIN, INPUT);
    pinMode(LED_LAMP_PIN, OUTPUT);
}

void loop()
{
    bool tooDark = analogRead(LIGHT_SENSOR_PIN) < LIGHT_LEVEL_THRESHOLD;
    bool motionDetected = digitalRead(MOTION_SENSOR_PIN);

    if (tooDark && motionDetected) {
        digitalWrite(LED_LAMP_PIN, HIGH);
        delay(30000); // спать 30 секунд
    } else {
        digitalWrite(LED_LAMP_PIN, LOW);
    }
}
```

Начало программы прежнее, но в loop появляется новое составное выражение, обозначаемое словами if, else и фигурными скобками. Давайте поймём в чём его суть.

Сразу после слова if компилятор C++ ожидает в круглых скобках увидеть логическое выражение, которое в этом случае называется условием. Оно имеет тот же смысл, что и для тернарного оператора. Если его значение — не ноль или истинно, выполняется блок кода, который следует сразу после условия в фигурных скобках. В нашем случае, если tooDark и motionDetected истины, будет выполнен блок кода из двух строк:

```
digitalWrite(LED_LAMP_PIN, HIGH);
delay(30000); // спать 30 секунд
```

Если же условие было нулём, или что то же самое — ложно, блок кода следующий за условием пропускается и не выполняется вовсе. Зато, если после после этого пропущенного блока следует слово else, выполняется блок кода в фигурных скобках, следующий после этого слова. В нашем случае, если либо tooDark, либо motionDetected были false, выполнится блок кода из одной строки:

```
digitalWrite(LED_LAMP_PIN, LOW);
```

Стоит отметить, что в случае выполнения условия, блок кода следующий после else не выполняется, а пропускается. То есть, на самом деле, выражение if — это отображение в языке программирования простого и понятного утверждения: «если что-то, делай то-то, а иначе делай сё-то».

Возвращаясь к нашему примеру, код можно интерпретировать так: «если требуется освещение, включить свет и уснуть на 30 секунд, а если свет не нужен — выключить его». Довольно просто и логично.

Использование выражений, которые меняют ход программы в зависимости от каких-то условий называется ветвлением программы, а блоки кода после if и else называются ветками.

Краткая версия if

При использовании выражения if совершенно не обязательно использовать ветку else. Если что-то должно произойти при выполнении условия, а при невыполнении не должно происходить ничего, ветку else можно просто опустить. Например, мы можем изменить loop нашей программы следующим образом:

```
void loop()

{

    bool tooDark = analogRead(LIGHT_SENSOR_PIN) < LIGHT_LEVEL_THRESHOLD;

    bool motionDetected = digitalRead(MOTION_SENSOR_PIN);

    bool lightningRequired = tooDark && motionDetected;



    digitalWrite(LED_LAMP_PIN, lightningRequired);
```

```
    if (lightningRequired) {
        delay(30000); // спать 30 секунд
    }
}
```

Эта вариация кода делает абсолютно то же, что и раньше. Просто теперь код организован иначе. Мы в любом случае делаем указание лампе на включение или выключение, вызывая digitalWrite с булевым значением lightningRequired, но засыпаем на 30 секунд только если мы включаем свет, т.е. если переменная lightningRequired истинна. Если мы только что выключали свет, спать не нужно: нужно пропустить delay и сразу оказаться в конце функции loop. Поэтому мы не писали ветку else вовсе.

Более того, если в блоке кода после if или else содержится всего одно выражение, фигурные скобки можно не писать:

```
if (lightningRequired)
    delay(30000); // спать 30 секунд
```

Вложенные выражения if

Блоки кода, используемые в условных выражениях — это самые обычные блоки, которые следуют общим правилам C++. Поэтому в ветку одного if можно запросто вкладывать другой if.

Для примера рассмотрим скетч устройства, которое контролирует открывание двери холодильника, автомобиля или чего-то такого. Мы хотим, чтобы при открытии двери включалась подсветка, а если дверь остаётся открытой более 20 секунд, раздавался бы предупреждающий писк. При этом, хочется, чтобы писк можно было отключить переключателем, на случай когда производится длительная загрузка/разгрузка: так он не будет нервировать.

Допустим на дверце установлен постоянный магнит, а на раме бинарный цифровой датчик магнитного поля. Если магнитное поле фиксируется — дверца закрыта; иначе магнит отходит от датчика слишком далеко, магнитного поля нет — можно понять, что дверца открыта. Также мы подключим к Arduino лампу подсветки, пьезо-пищалку (buzzer) и рокерный выключатель.

В этом случае код работающего устройства может выглядеть так:

```
#define DOOR_SENSOR_PIN    2
#define BUZZER_PIN         3
#define SWITCH_PIN         4
#define LAMP_PIN           5

#define BUZZ_TIMEOUT       20
#define BUZZ_FREQUENCY     4000

void setup()
{
    pinMode(DOOR_SENSOR_PIN, INPUT);
    pinMode(SWITCH_PIN, INPUT);
    pinMode(BUZZER_PIN, OUTPUT);
    pinMode(LAMP_PIN, OUTPUT);
}

void loop()
{
    bool doorOpened = !digitalRead(DOOR_SENSOR_PIN);

    if (doorOpened) {
        digitalWrite(LAMP_PIN, HIGH);
        delay(BUZZ_TIMEOUT * 1000);
        bool buzzEnabled = digitalRead(SWITCH_PIN);
        if (buzzEnabled) {
            tone(BUZZER_PIN, BUZZ_FREQUENCY);
        }
    } else {
        digitalWrite(LAMP_PIN, LOW);
        noTone(BUZZER_PIN);
    }
```

```
}
```

Давайте разберём происходящее в loop. Первым делом мы определяем переменную doorOpened и присваиваем ей значение выражения !digitalRead(DOOR_SENSOR_PIN). Как уже говорилось, дверь стоит считать открытой, если магнитного поля нет, т.е. digitalRead для нашего сенсора возвращает false. Символ ! перед логическим выражением в C++ означает оператор логического «не» или просто оператор отрицания. Он действует на значение, записанное после него, и из true делает false, а изfalse делает true. Как раз то, что нам нужно в этом случае.

Далее следует выражение if, проверяющее открыта ли дверь. Если да, начинается исполнение блока кода, следующего непосредственно за условием. В нём мы первым делом включаем подсветку. Затем засыпаем на 20 секунд (20×1000 мс).

Далее мы должны включить пищалку если только она не была умышленно выключена. Для этого мы проверяем состояние переключателя, который за это отвечает. Мы объявляем переменную buzzEnabled, которой присваиваем логическое значение, считанное с выключателя.

Обратите внимание, переменная buzzEnabled объявлена прямо внутри блока кода, следующего за if. Так делать можно и нужно: хорошей практикой является объявление переменных как можно ближе к тому месту, где они впервые используются.

Напомним о понятии области видимости переменных: переменные доступны для использования только внутри того блока, где они объявлены. В нашем случае buzzEnabled может быть использована в выражениях внутри блока кода, следующего за if (doorOpened), но попытка обращения к ней откуда-то ещё: например, из блока кода ветки else или непосредственно в loop вне ветки if, приведёт к ошибке на этапе компиляции программы. И это хорошо: нам не стоит впутывать переменную, которая нужна сию секунду в другие происходящие процессы; это делает программу чище и нагляднее.

Вслед за чтением переключателя следует вложенное условное выражение. Его суть абсолютно та же, что и ранее: в зависимости от условия выполнить или не выполнить код. Одно лишь отличие: оно расположено прямо в блоке кода ветки другого if. Это распространённая практика, которая встречается в программировании довольно часто. На самом деле, во вложенный if можно вкладывать другие if, в них ещё одни и т.д. до бесконечности: язык C++ вас в этом не ограничивает.

Возвращаясь к примеру, если переключатель находится в положении «включён», т.е. buzzEnabledсодержит true, мы включаем писк. Это делается с помощью встроенной функции tone. Она принимает 2 аргумента: номер пина Arduino, куда подключён пьезоизлучатель и частоту писка. В данном случае, мы выбрали частоту 4000 Гц, т.е. 4 КГц.

Наконец, если дверь была закрыта, т.е. doorOpened содержит ложь, мы убеждаемся, что подсветка и пищалка выключены. Это делается в ветке else первого условия if. Как выключить лампу вы понимаете, а функция noTone, как можно догадаться, отключает писк на указанном пине.

Вспоминая о правилах короткой записи и встраивании выражений, мы можем чуть упростить код loop. Он может выглядеть так:

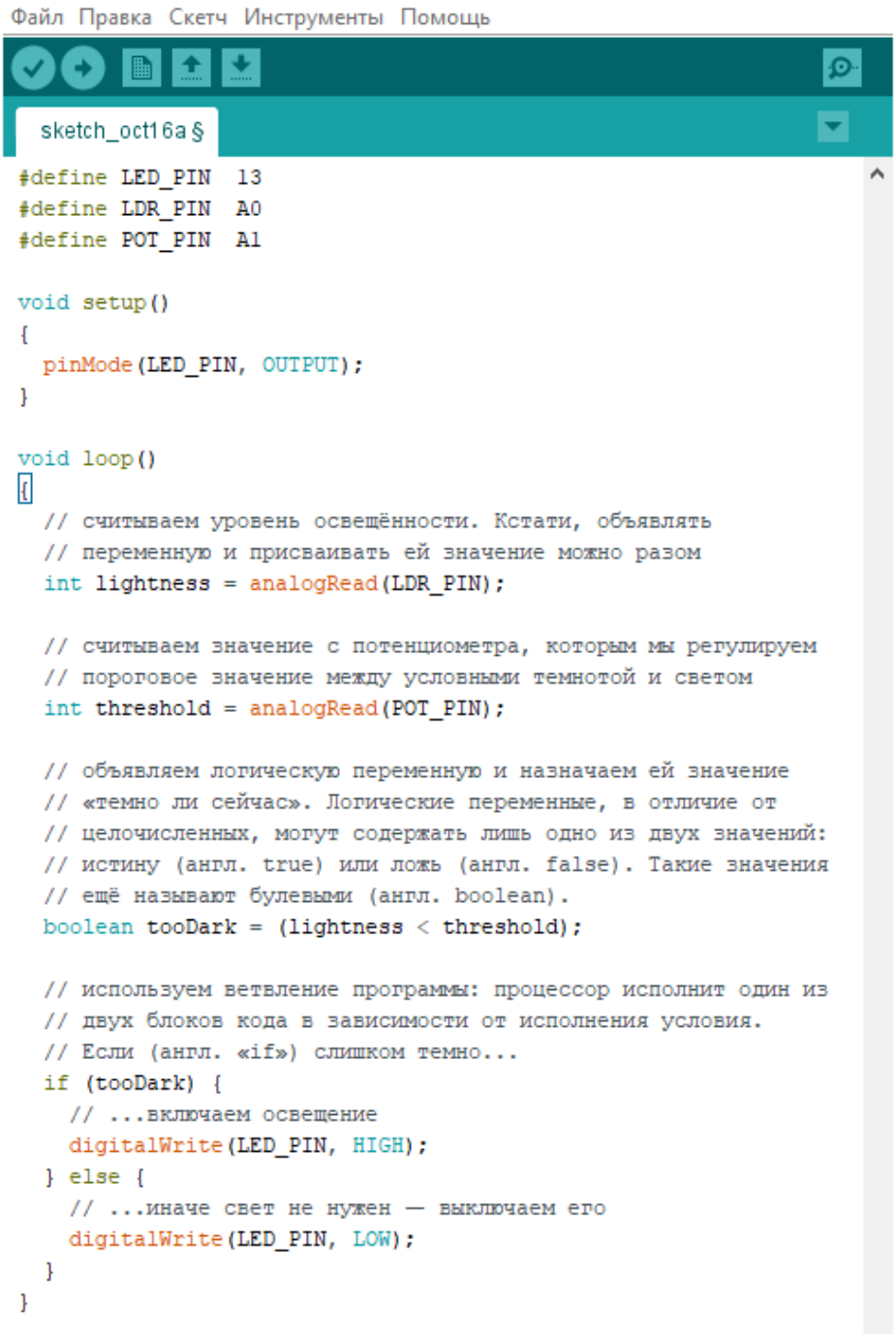
```
void loop()
{
    if (!digitalRead(DOOR_SENSOR_PIN)) {
        digitalWrite(LAMP_PIN, HIGH);
        delay(BUZZ_TIMEOUT * 1000);
        if (digitalRead(SWITCH_PIN))
            tone(BUZZER_PIN, BUZZ_FREQUENCY);
    } else {
        digitalWrite(LAMP_PIN, LOW);
        noTone(BUZZER_PIN);
    }
}
```

Мы убрали фигурные скобки для внутреннего if и встроили вызовы digitalRead прямо в условные выражения.

Обратите внимание, как правильное использование отступов в блоках кода, помогает легко понять к какому if относится else и что за чем и при каких условиях следует. Никогда не пишите без отступов несмотря на то, что компилятору всё равно. Это делает код не читаемым и для других людей, и для вас самих:

```
void loop()
{
if (!digitalRead(DOOR_SENSOR_PIN)) {
digitalWrite(LAMP_PIN, HIGH);
delay(BUZZ_TIMEOUT * 1000);
if (digitalRead(SWITCH_PIN))
tone(BUZZER_PIN, BUZZ_FREQUENCY);
} else {
digitalWrite(LAMP_PIN, LOW);
noTone(BUZZER_PIN);
}
}
```

3. Наберите в редакторе кода следующий код программы:



✓ Мы используем новый тип переменных — boolean, которые хранят только значения true (истина, 1) или false (ложь, 0). Эти значения являются результатом вычисления логических выражений. В данном примере логическое выражение — это lightness < threshold. На человеческом языке это звучит как: «освещенность ниже порогового уровня». Такое высказывание будет истинным, когда освещенность ниже порогового уровня. Микроконтроллер может сравнить значения

переменных lightness и threshold, которые, в свою очередь, являются результатами измерений, и вычислить истинность логического выражения.

- ✓ Мы взяли это логическое выражение в скобки только для наглядности. Всегда лучше писать читабельный код. В других случаях скобки могут влиять на порядок действий, как в обычной арифметике.
- ✓ В нашем эксперименте логическое выражение будет истинным, когда значение lightness меньше значения threshold, потому что мы использовали оператор <. Мы можем использовать операторы >, <=, >=, ==, !=, которые значат «больше», «меньше или равно», «больше или равно», «равно», «не равно» соответственно.
- ✓ Будьте особенно внимательны с логическим оператором == и не путайте его с оператором присваивания =. В первом случае мы сравниваем значения выражений и получаем логическое значение (истина или ложь), а во втором случае присваиваем левому операнду значение правого. Компилятор не знает наших намерений и ошибку не выдаст, а мы можем нечаянно изменить значение какой-нибудь переменной и затем долго разыскивать ошибку.
- ✓ Условный оператор if («если») — один из ключевых в большинстве языков программирования. С его помощью мы можем выполнять не только жестко заданную последовательность действий, но принимать решения, по какой ветви алгоритма идти, в зависимости от неких условий.
- ✓ У логического выражения lightness < threshold есть значение: true или false. Мы вычислили его и поместили в булеву переменную tooDark («слишком темно»). Таким образом мы как бы говорим «если слишком темно, то включить светодиод»
- ✓ С таким же успехом мы могли бы сказать «если освещенность меньше порогового уровня, то включить светодиод», т.е. передать в if всё логическое выражение:

```
if (lightness < threshold) {
// ...
}
```

- ✓ За условным оператором if обязательно следует блок кода, который выполняется в случае истинности логического выражения. Не забывайте про обе фигурные скобки {}!
- ✓ Если в случае истинности выражения нам нужно выполнить только одну инструкцию, ее можно написать сразу после if (…) без фигурных скобок:

```
if (lightness < threshold)
digitalWrite(LED_PIN, HIGH);
```

- ✓ Оператор if может быть расширен конструкцией else («иначе»). Блок кода или единственная инструкция, следующий за ней, будет выполнен только если логическое выражение в if имеет значение false, «ложь». Правила, касающиеся фигурных скобок, такие же. В нашем эксперименте мы написали «если слишком темно, включить светодиод, иначе выключить светодиод».

Задание 4. Ответьте на следующие вопросы

1. Если мы установим фоторезистор между аналоговым входом и землей, наше устройство будет работать наоборот: светодиод будет включаться при увеличении количества света. Почему?
2. Какой результат работы устройства мы получим, если свет от светодиода будет падать на фоторезистор?
3. Если мы все же установили фоторезистор так, как сказано в предыдущем вопросе, как нам нужно изменить программу, чтобы устройство работало верно?
4. Допустим, у нас есть код if (условие) {действие;}. В каких случаях будет выполнено действие?
5. При каких значениях y выражение x + y > 0 будет истинным, если x > 0?

6. Обязательно ли указывать, какие инструкции выполнять, если условие в операторе if ложно?
7. Чем отличается оператор == от оператора =?
8. Если мы используем конструкцию if (условие) действие1; else действие2;, может ли быть ситуация, когда ни одно из действий не выполнится? Почему?

Задание 5. Самостоятельно измените существующую программу и схему

1. Перепишите программу без использования переменной `tooDark` с сохранением функционала устройства.
2. Добавьте в схему еще один светодиод. Дополните программу так, чтобы при падении освещенности ниже порогового значения включался один светодиод, а при падении освещенности ниже половины от порогового значения включались оба светодиода.
3. Измените схему и программу так, чтобы светодиоды включались по прежнему принципу, но светились тем сильнее, чем меньше света падает на фоторезистор.