Практическая работа 2

Проект маячок с нарастающей яркостью

В этом проекте научимся управлять яркостью светодиода.

Задание 1. Ответить на вопросы

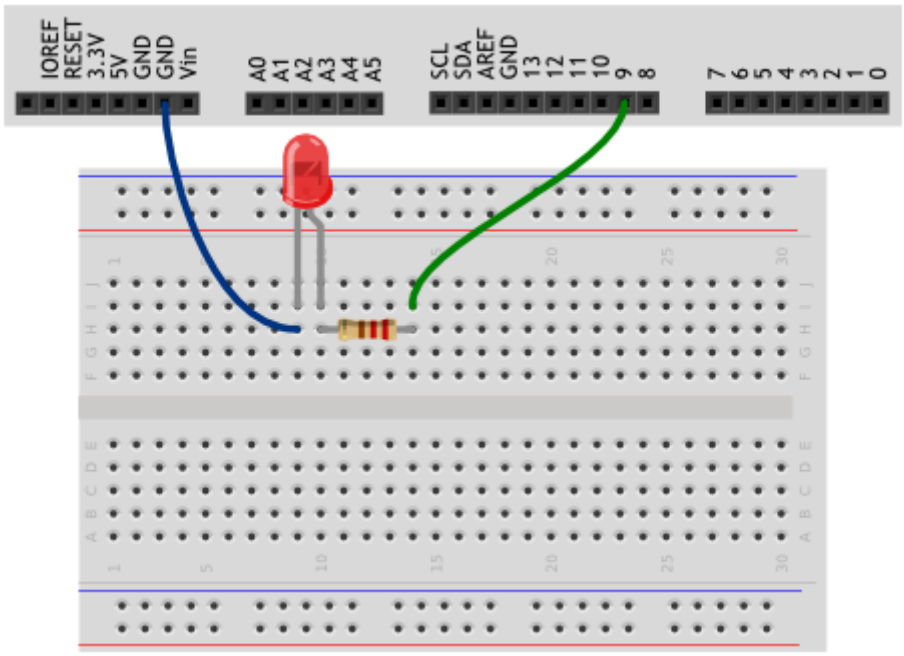
1. Что такое ШИМ?
2. Заполните таблицу при исходном напряжении 5В

Скважность, %	Напряжение, В
50	
20	
33	
40	
80	

Задание 2. Список деталей для эксперимента

1. 1 плата Arduino Uno
2. 1 беспаечная макетная плата
3. 1 светодиод
4. 1 резистор номиналом 220 Ом
5. 2 провода «папа-папа»

Схема на макетной плате:



1. Зарисуйте принципиальную схему установки.

Обратите внимание

✓ Не забудьте, как соединены рельсы в беспаечной макетной плате. Если на вашей макетке красная и синяя линии вдоль длинных рельс прерываются в середине, значит проводник внутри макетки тоже прерывается!

✓ Катод («минус») светодиода — короткая ножка, именно её нужно соединять с землёй (GND)

✓ Не пренебрегайте резистором, иначе светодиод выйдет из строя

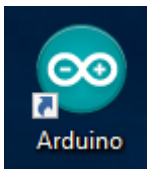
✓ Выбрать резистор нужного номинала можно с помощью таблицы маркировки или с помощью мультиметра в режиме измерения сопротивления

✓ Плата Arduino имеет три пина GND, используйте любой из них

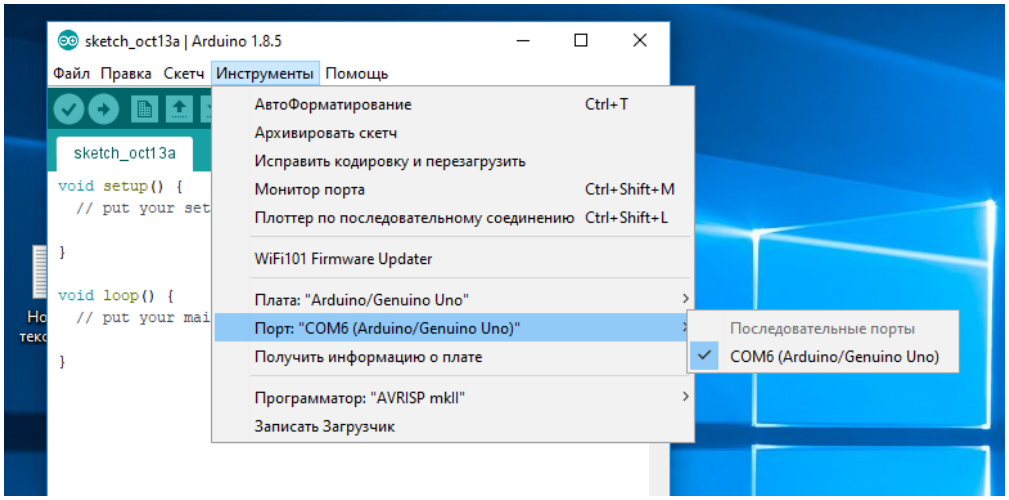
✓ Не любой порт Arduino поддерживает широтно-импульсную модуляцию, если вы хотите регулировать напряжение, вам подойдут пины, помеченные символом тильда «~». Для Arduino Uno это пины 3, 5, 6, 9, 10, 11

Задание 3. Программирование микроконтроллера

1. Запустите приложение



2. Убедитесь, что выбран нужный порт



В языке C++ пробелы, переносы строк, символы табуляции не имеют большого значения для компилятора. Там где стоит пробел, может быть перенос строки и наоборот. На самом деле 10 пробелов подряд, 2 переноса строки и ещё 5 пробелов — это всё эквивалент одного пробела.

Пустое пространство — это инструмент программиста, с помощью которого можно или сделать программу понятной и наглядной, или изуродовать до неузнаваемости.

Чтобы следовать негласному закону оформления программ, который уважается на форумах, при чтении другими людьми, легко воспринимается вами же, следуйте нескольким простым правилам:

1. Всегда, при начале нового блока между { и } увеличивайте отступ. Обычно используют 2 или 4 пробела. Выберите одно из значений и придерживайтесь его всюду.

2. Как и в естественном языке: ставьте пробел после запятых и не ставьте до.

3. Размещайте символ начала блока { на новой строке на текущем уровне отступа или в конце предыдущей. А символ конца блока } на отдельной строке на текущем уровне отступа.

4. Используйте пустые строки для разделения смысловых блоков:

Вы могли заинтересоваться: зачем в конце каждого выражения ставится точка с запятой? Таковы правила C++. Подобные правила называются синтаксисом языка. По символу ; компилятор понимает где заканчивается выражение.

Как уже говорилось, переносы строк для него — пустой звук, поэтому ориентируется он на этот знак препинания. Это позволяет записывать сразу несколько выражений в одной строке.

Одно из правил качественного программирования: «пишите код так, чтобы он был настолько понятным, что не нуждался бы в пояснениях». Это возможно, но не всегда. Для того, чтобы пояснить какие-то не очевидные моменты в коде его читателям: вашим коллегам или вам самому через месяц, существуют так называемые комментарии.

Это конструкции в программном коде, которые полностью игнорируются компилятором и имеют значение только для читателя.

Между символами /* и */ можно писать сколько угодно строк комментариев. А после последовательности // комментарием считается всё, что следует до конца строки.

Макроопределения

Мы можем единожды указать, что левый светодиод — это пин 13, правый — пин 12, а переключать состояния нужно каждые X миллисекунд. Для этого каждому значению назначается понятное имя, которое затем и используется для обращения:

```
#define LEFT_LED      13
#define RIGHT_LED     12
#define SWITCH_TIME   2000

void setup()
{
    pinMode(LEFT_LED, OUTPUT);
    pinMode(RIGHT_LED, OUTPUT);
}

void loop()
{
    digitalWrite(LEFT_LED, HIGH);
    digitalWrite(RIGHT_LED, LOW);
    delay(SWITCH_TIME);

    digitalWrite(LEFT_LED, LOW);
    digitalWrite(RIGHT_LED, HIGH);
    delay(SWITCH_TIME);
}
```

теперь для изменения параметров устройства достаточно изменить нужные значения в начале программы и не думать об изменениях в самой логике. Кроме того выражение вроде digitalWrite(RIGHT_LED, LOW) гораздо более информативно нежели digitalWrite(12, LOW) и даёт чёткое понимание того, что имел в виду автор.

Конструкция **#define** называется макроопределением. Она говорит компилятору о том, что всякий раз, когда он видит указанное имя, стоит использовать на этом месте указанное значение.

Обратите внимание: макроопределения #define не завершаются точкой с запятой в конце строки. Дело в том, что #define — это не обычное выражение, а так называемая препроцессорная директива. Подстановка конкретных значений вместо имён происходит ещё до компиляции, на стадии предварительной обработки исходного файла. На этой стадии, по сути, компилятор проделывает операцию как в текстовом редакторе: «Найти все и заменить». Просто результат этой операции не сохраняется в файл, а тут же им используется для непосредственной компиляции.

Если бы вы поставили ; после #define, в результате обработки компилятор увидел бы такой код:

```
// Если завершать директивы точками с запятой…
#define LEFT_LED      13;
#define RIGHT_LED     12;
#define SWITCH_TIME   2000;

// …они появятся в самом неподходящем месте
void setup()
{
    pinMode(12;, OUTPUT);
    pinMode(13;, OUTPUT);
}

void loop()
{
    digitalWrite(12;, HIGH);
    digitalWrite(13;, LOW);
    delay(2000;);

    digitalWrite(12;, LOW);
    digitalWrite(13;, HIGH);
    delay(2000;);
}
```

Попытка скомпилировать такой скетч приведёт к ошибке.

Встроенные макроопределения

Вы могли догадаться, что уже знакомые нам значения HIGH, LOW, OUTPUT — это также не что иное как макроопределения. Просто **#define** для них написан в некотором другом месте и мы можем просто сразу ими пользоваться.

На самом деле код мигания светодиодом:

```
void setup()
{
    pinMode(13, OUTPUT);
}

void loop()
{
    digitalWrite(13, HIGH);
    delay(100);
    digitalWrite(13, LOW);
    delay(900);
}
```

С точки зрения компилятора:

```
void setup()
{
    pinMode(13, 1);
}

void loop()
{
    digitalWrite(13, 1);
    delay(100);
    digitalWrite(13, 0);
    delay(900);
}
```

Если вы откроете файл hardware/arduino/cores/arduino/Arduino.h в вашем дистрибутиве Arduino IDE, вы сможете увидеть, что HIGH — это макроопределение 1, LOW — 0, OUTPUT — 1 и т.д. Эти значения используются настолько часто, что они встроены таким образом.

Использование понятных имён вместо магических чисел — это один из признаков профессионализма. Это делает код более понятным и простым для изменений, что всегда уважается.

Об именах макроопределений

По негласному соглашению все макроопределения должны иметь имена, написанные заглавными буквами с символом нижнего прочерка _ на месте пробелов.

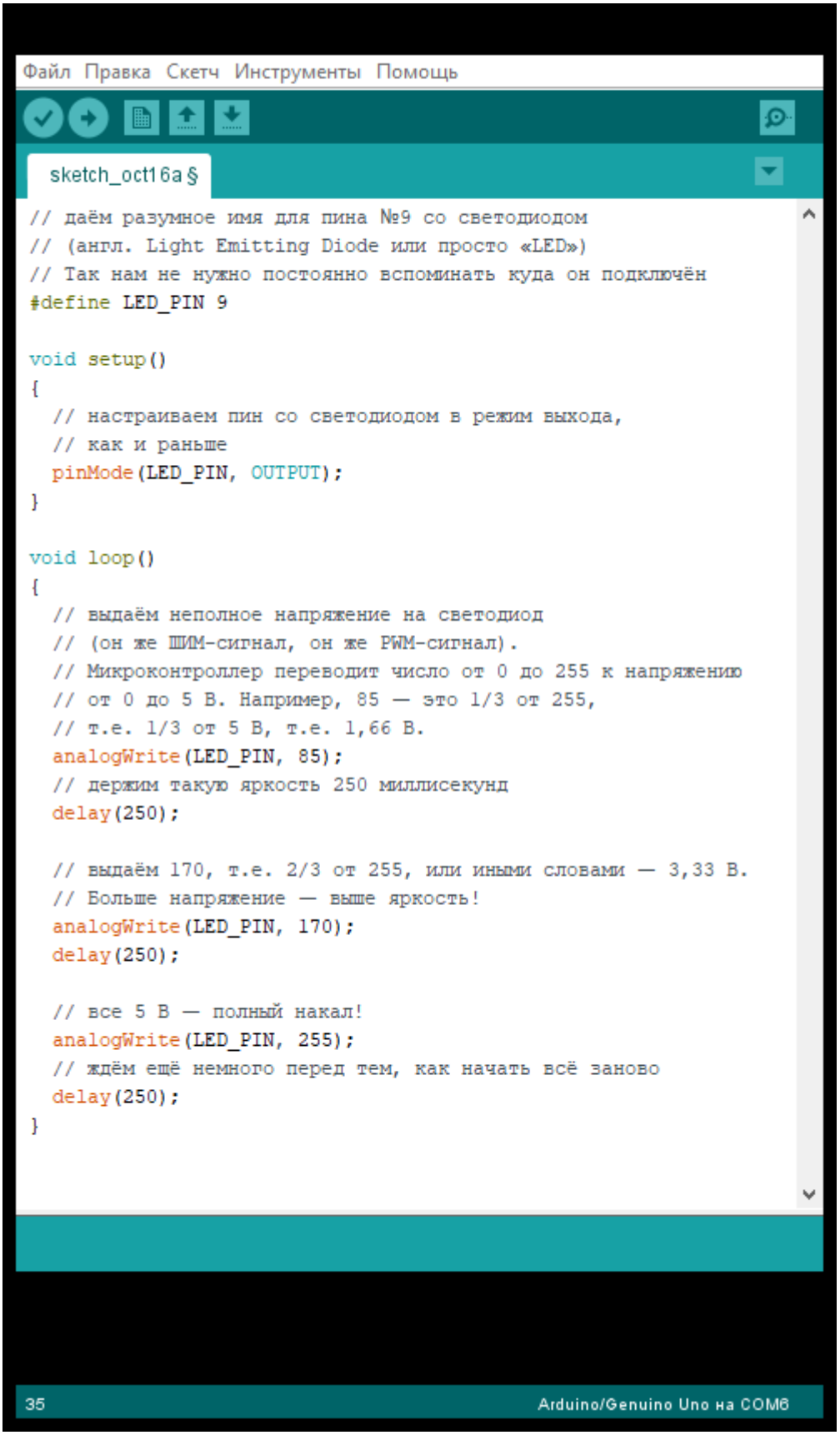
```
// плохо
#define ledpin  13
#define led_pin 13
#define LedPin  13

// хорошо
#define LED_PIN 13
```

Это настолько привычное правило, что вы можете ввести в заблуждение других, если не будете его придерживаться. Опять же, следование общепринятым канонам — признак профи.

Стоит отметить, что язык C++, как и многие другие считает строчные и заглавные буквы различными, поэтому если к макроопределению LED_PIN вы затем попытаетесь обратиться, написав led_pin будет выдана ошибка компилятора о том, что он не понимает что такое led_pin. То же самое касается имён функций и всего остального.

3. Наберите в редакторе кода следующий код программы:



✓ Идентификаторы переменных, констант, функций (в этом примере идентификатор LED_PIN) являются одним словом (т.е. нельзя создать идентификатор LED PIN).

✓ Идентификаторы могут состоять из латинских букв, цифр и символов подчеркивания _. При этом идентификатор не может начинаться с цифры.

PRINT // верно

PRINT_3D // верно

MY_PRINT_3D // верно

_PRINT_3D // верно

3D_PRINT // ошибка

ПЕЧАТЬ_3Д // ошибка

PRINT:3D // ошибка

✓ Регистр букв в идентификаторе имеет значение. Т.е. LED_PIN, LED_pin и led_pin с точки зрения компилятора — различные идентификаторы

✓ Идентификаторы, создаваемые пользователем, не должны совпадать с предопределенными идентификаторами и стандартными конструкциями языка; если среда разработки подсветила введенный идентификтор каким-либо цветом, замените его на другой

✓ Директива #**define** просто говорит компилятору заменить все вхождения заданного идентификатора на значение, заданное после пробела (здесь 9), эти директивы помещают в начало кода. В конце данной директивы точка с запятой ; не допустима

✓ Названия идентификаторов всегда нужно делать осмысленными, чтобы при возвращении к ранее написанному коду вам было ясно, зачем нужен каждый из них

✓ Также полезно снабжать код программы комментариями: в примерах мы видим однострочные комментарии, которые начинаются с двух прямых слэшей // и многострочные, заключённые между /* */

// однострочный комментарий следует после двойного слеша до конца строки

/* многострочный комментарий

помещается между парой слеш-звездочка и звездочка-слеш */

✓ комментарии игнорируются компилятором, зато полезны людям при чтении давно написанного, а особенно чужого, кода

✓ Функция **analogWrite(pin, value)** не возвращает никакого значения и принимает два параметра:

pin — номер порта, на который мы отправляем сигнал

value — значение скважности ШИМ, которое мы отправляем на порт. Он может принимать целочисленное значение от 0 до 255, где 0 — это 0%, а 255 — это 100%

Задание 4. Ответьте на следующие вопросы

1. Какие из следующих идентификаторов корректны и не вызовут ошибку?

13pin

MOTOR_1

контакт_светодиода

sensor value

leftServo

my-var

distance_eval2

2. Что произойдет, если создать директиву **#define HIGH LOW**?
3. Почему мы не сможем регулировать яркость светодиода, подключенного к порту 7?
4. Какое усреднённое напряжение мы получим на пине 6, если вызовем функцию **analogWrite(6, 153)**?
5. Какое значение параметра **value** нужно передать функции **analogWrite**, чтобы получить усреднённое напряжение 2 В?

Задание 5. Самостоятельно измените существующую программу и схему

1. Отключите питание, отключите светодиод от 9-го порта и подключите к 11-му. Измените программу так, чтобы схема снова заработала

2. Измените код программы так, чтобы в течение секунды на светодиод последовательно подавалось усреднённое напряжение 0, 1, 2, 3, 4, 5 В
3. Возьмите еще один светодиод, резистор на 220 Ом и соберите аналогичную схему на этой же макетке, подключив светодиод к пину номер 3 и другому входу GND, измените программу так, чтобы светодиоды мигали в противофазу: первый выключен, второй горит максимально ярко и до противоположного состояния
4. Для генерации случайных чисел используется функция Random(min, max). Измените программу так, чтобы напряжение на оба светодиода подавалось случайным образом
5. Возьмите 3 светодиода и создайте программу «иллюзия огня», которая будет менять яркость светодиодов таким образом, что со стороны это будет выглядеть как горящий огонь.