Синтаксис и структура кода

Микроконтроллер - штука довольно глупая, не способная к мышлению и импровизации, умеет только выполнять конкретные инструкции от программиста. Общение с МК происходит в письменном виде на языке программирования, язык этот очень чёткий, строгий и имеет свой синтаксис и некоторые нормы оформления. И если синтаксическая ошибка приводит к ошибке компиляции кода или к неправильной работе устройства, то оформление кода служит для удобства его чтения программистом.

Синтаксис

Тела функций заключаются в фигурные скобки { }

Код внутри фигурных скобок иногда называют **блоком кода**.

* Каждая команда заканчивается точкой с запятой ;
* Метод применяется к объекту через точку. Пример: Serial.begin();
* Вызов функции или метода всегда заканчивается скобками, даже если функция не принимает параметров. Пример: loop()
* Разделитель десятичных дробей - **точка**. Пример: 0.25

 У запятой тут другое применение.

* Запятыми перечисляются аргументы функций и методов, члены массива, также через запятую можно выполнить несколько действий в одну строчку. Пример: digitalWrite(3, HIGH); массив - int myArray[] = {3, 4, 5 ,6};
* Одиночный символ заключается в одиночные кавычки 'а'
* Строка и массив символов заключается в двойные кавычки "строка"
* Имена переменных могут содержать **латинские буквы** в верхнем и нижнем регистре (большие и маленькие), **цифры** и **подчеркивание**. Пример: myVal\_35
* Имена переменных **не могут начинаться с цифры**. Только с буквы или подчёркивания.
* **Регистр имеет значение**, т.е. большая буква отличается от маленькой. Пример: имен val и Val - не одно и то же.

К синтаксису также можно отнести комментарии, т.к. в разных языках они выделяются по-разному. Комментарий это обычный текст, который игнорируется на этапе компиляции и не попадает в итоговую программу для МК. Комментарии нужны для пояснения кода, как себе самому, так и другим возможным его читателям. В C++ у нас два типа комментариев:

* Однострочный комментарий

// однострочный комментарий

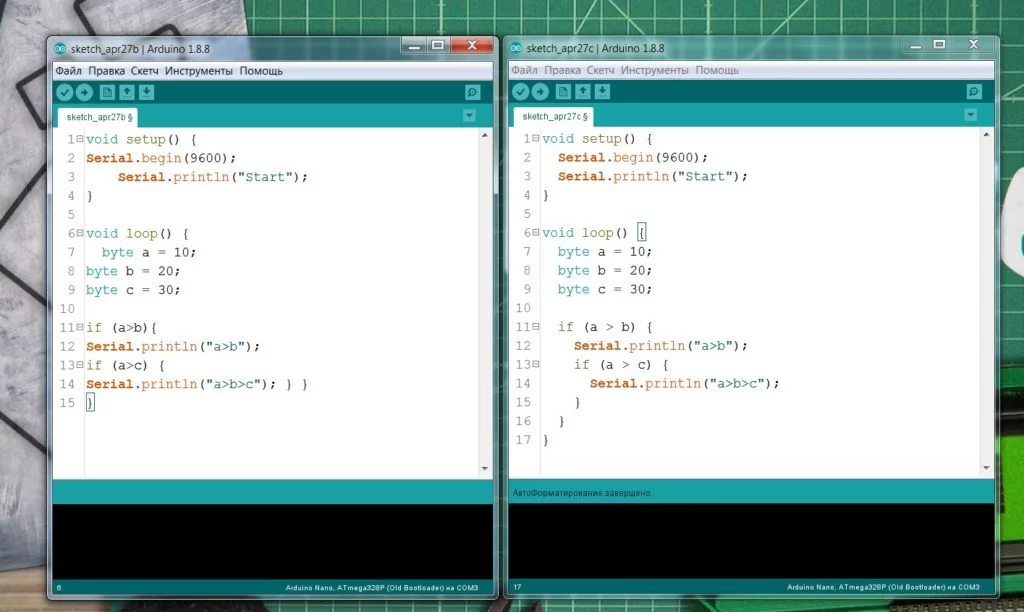
// компилятор меня игнорирует =(

* Многострочный комментарий

/\* Многострочный

комментарий \*/

Оформление



Не бойтесь, во всех серьезных средах разработки есть автоформатирование кода, оно работает как в процессе написания, так и "по кнопке". Arduino IDE - не исключение, в ней код форматируется комбинацией клавиш **Ctrl+T**:

* Между математическими действиями, знаками сравнения, присваивания и всем подобным ставится **пробел**.
* Как и в обычном тексте, пробел **ставится после** и **не ставится перед** запятой, двоеточием, точкой с запятой.
* Отступ от левого края экрана - знак табуляции, код сдвигается вправо и на одном расстоянии формируются команды из одного блока кода. В Arduino IDE одна табуляция равна двум пробелам. Можно использовать клавишу **Tab**.
* Каждое действие выполняется с новой строки (автоформатирование это не исправляет).
* Имена функций и переменных принято называть с маленькой буквы. Пример:

value

* Если имя состоит из двух и более слов, то их принято разделять. Есть два способа:
  + **camelCase** (верблюжий стиль): первая буква маленькая, каждая первая буква следующего слова - большая.
  + **under\_score** (подчёркивание): все буквы маленькие, разделитель - подчёркивание.



**Частый вопрос**: влияет ли длина названия переменной на занимаемую прошивкой память? На вес файла с программой на компьютере - влияет. На вес загруженной в микроконтроллер прошивки - **не влияет**, потому что код преобразуется в машинный, в котором нет имён.

* Имена типов данных и классов принято писать с большой буквы. Пример:

Signal, Servo

* Имена констант принято писать в верхнем регистре, разделение - подчеркивание. Пример:

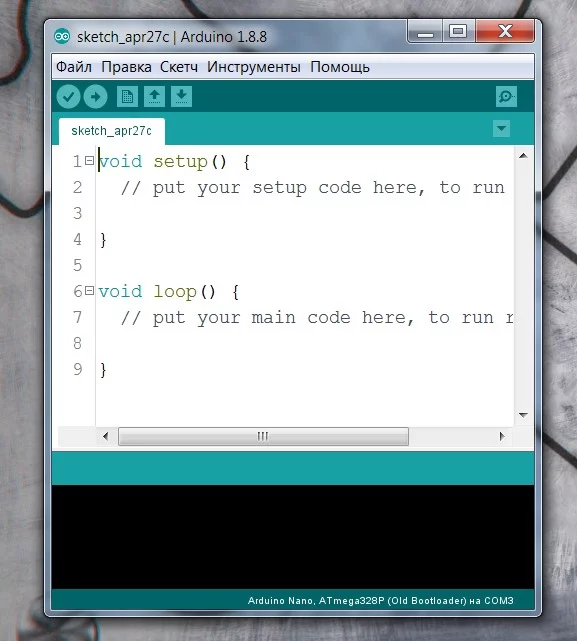
MOTOR\_SPEED

* При написании библиотек и классов, имена внутренних (приватных) переменных принято писать, начиная со знака подчёркивания. Пример: \_position
* Несколько общепринятых сокращений для названий переменных, вы часто будете встречать их в чужих прошивках и библиотеках:
  + **button** - btn, кнопка
  + **index** - idx - i, индекс
  + **buffer** - buf, буфер
  + **value** - val, значение
  + **variable** - var, переменная
  + **pointer** - ptr, указатель
* Имена функций и методов принято начинать с глагола, кратко описывающего действие функции. Вот те из них, которые вы будете встречать постоянно:
  + **get** - получить значение (getValue)
  + **set** - установить значение (setTime)
  + **print**, **show** - показать что-то
  + **read** - прочитать
  + **write** - записать
  + **change** - изменить
  + **clear** - очистить
  + **begin**, **start** - начать
  + **end**, **stop** - закончить, остановить

Структура кода

* Переменная любого типа должна вызываться только **после** своего объявления. Иначе будет ошибка
* Объявление и использование классов или типов данных из библиотеки/файла должно быть **после** подключения библиотеки/файла
* Функция может вызываться **как до, так и после** объявления, потому что C++ компилируемый язык, компиляция проходит в несколько этапов, и функции "выделяются" отдельно, поэтому могут вызываться в любом месте программы

При запуске Arduino IDE даёт нам заготовку в виде двух обязательных функций: **setup()** и **loop()**



Код в блоке setup()выполняется один раз при каждом запуске МК. Код в блоке loop()

выполняется "по кругу" на всём протяжении работы программы, начиная с момента завершения выполнения setup(). **Для любознательных**: если вы уже знакомы с языком C++, то вероятно спросите "а где же int main()и вообще файл **main.cpp**?". Всё очень просто: int main() за вас уже написали внутри файла main.cpp, который лежит глубоко в файлах "ядра", а setup() и loop() встроены в него следующим образом:

// main.cpp

// где-то в глубинах ядра Arduino

int main() {

setup();

for (;;) {

loop();

}

return 0;

}

На протяжении нескольких лет работы с Arduino я сформировал для себя следующую структуру **скетча**:

1. Описание прошивки, ссылки, заметки
2. Константы-настройки (define и обычные)
3. Служебные константы (которые следует менять только с полным осознанием дела)
4. Подключаемые библиотеки и внешние файлы, объявление соответствующих им типов данных и классов
5. Глобальные переменные
6. setup()
7. loop()
8. Свои функции

Пример

Данный скетч плавно крутит

сервопривод туда-обратно

между мин. и макс. углами

by AlexGyver

\*/

// -------- НАСТРОЙКИ ---------

#define SERVO\_PIN 13 // сюда подключена серво

#define SERVO\_SPEED 3 // скорость серво

#define MIN\_ANGLE 50 // мин. угол

#define MAX\_ANGLE 120 // макс. угол

// ------- БИБЛИОТЕКИ -------

#include <Servo.h>

Servo myservo;

// ------- ПЕРЕМЕННЫЕ -------

uint32\_t servoTimer;

boolean servoDirection;

int servoAngle;

// --------- SETUP ----------

void setup() {

myservo.attach(SERVO\_PIN);}

// ---------- LOOP ----------

void loop() {

turnServo();

}

// --------- ФУНКЦИИ --------

void turnServo() {

if (millis() - servoTimer >= 50) { // каждые 50 мс

servoTimer = millis();

if (servoDirection) {

servoAngle += SERVO\_SPEED;

if (servoAngle >= MAX\_ANGLE) {

servoAngle = MAX\_ANGLE;

servoDirection = **false**;

}

} else {

servoAngle -= SERVO\_SPEED;

if (servoAngle <= MIN\_ANGLE) {

servoAngle = MIN\_ANGLE;

servoDirection = **true**;

}

}myservo.write(servoAngle);

}

}

Подключение библиотек и файлов

В реальной работе вы очень часто будете использовать библиотеки или просто внешние файлы, они подключаются в главный файл (файлу прошивки) при помощи директивы

#include

. Данная команда сообщает компилятору, что нужно найти и добавить в программу указанный файл. Этот файл может содержать свои

#include

 и тянуть за собой и другие файлы, таким образом программа может быть разбита на множество независимых файлов. Рассмотрим пример:

#include <Servo.h> // подключает библиотеку Servo.h

#include “Servo.h” // тоже подключает библиотеку Servo.h

В чём отличие <> и ""? Когда указываем название "в кавычках", компилятор сначала ищет файл в папке со скетчем, а затем в папке с библиотеками. При использовании <галочек> компилятор ищет файл только в папке с библиотеками! К слову о папках с библиотеками: их две, в обеих будет производиться поиск библиотек.

* Пользовательская папка: *Документы/Arduino/libraries.* Сюда библиотеки попадают при добавлении их через "подключить .zip библиотеку" и при установке из менеджера библиотек.
* Папка с программой: *C:/Program Files (x86)/Arduino/libraries* (или *C:/Program Files/Arduino/libraries* для 32-разрядной Windows). Здесь хранятся встроенные стандартные библиотеки.

Не используйте мышку

* **Ctrl+←**,**Ctrl+→**– переместить курсор влево/вправо НА ОДНО СЛОВО
* **Home** , **End** – переместить курсор в начало/конец строки
* **Shift+←**,**Shift+→**– выделить символ слева/справа от курсора
* **Shift+Ctrl+←**,**Shift+Ctrl+→**– выделить слово слева/справа от курсора
* **Shift+Home**,**Shift+End**– выделить все символы от текущего положения курсора до начала/конца строки
* **Ctrl+Z** – отменить последнее действие
* **Ctrl+Y** – повторить отменённое действие
* **Ctrl+C** – копировать выделенный текст
* **Ctrl+X** – вырезать выделенный текст
* **Ctrl+V** – вставить текст из буфера обмена

Местные сочетания:

* **Ctrl+U** – загрузить прошивку в Arduino
* **Ctrl+R** – скомпилировать (проверить)
* **Ctrl+Shift+M** – открыть монитор порта
* **Ctrl+T** - автоформатирование

Для отодвигания комментариев в правую часть кода используйте **TAB**, а не **ПРОБЕЛ**. Нажатие TAB перемещает курсор по некоторой таблице, из-за чего ваши комментарии будут установлены на одинаковом расстоянии.