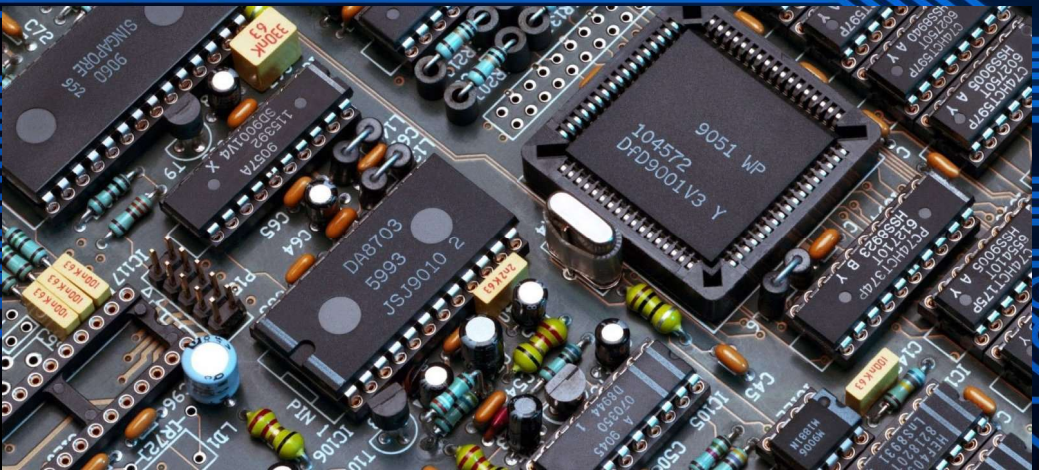
Безпека інтернет речей

Лекція №5



Лекцію проводить:

доц. Лимаренко Вячеслав Володимирович

к.т. 066-0708586

Операції з змінними і константами

Однією з основних функцій мікроконтроллера є виконання обчислень, як із числами напряму, так і зі значеннями змінних. Найпростіші математичні дії:

```
= привласнення
% залишок від ділення
* множення
/ / ділення
+ додавання
- віднімання
```

Простий приклад:

```
int a = 10;
int b = 20;
int c = a + b;  // c = 30
int d = a * b;  // d = 200
// так теж можна
d = d / a;      // d = 20
c = c * d;      // c = 600
```

Операції з змінними і константами
Складені оператори

Коли змінна бере участь у розрахунку свого значення: існують також складені оператори, що скорочують запис:

```
+= складене додавання:       a+= 10 рівносильно a = a + 10
-= складене віднімання:      a-= 10 рівносильно a = a - 10
*= складене множення:        a*= 10 рівносильно a = a * 10
/= складене ділення:         a/= 10 рівносильно a = a / 10
%= додати решту від поділу: a%= 10 рівносильно a = a + a % 10
```

Дуже часто в програмуванні використовується додавання або віднімання одиниці, для чого також є короткий запис:

```
++ (плюс плюс) інкремент:   a++ рівносильно a = a + 1
-- (мінус мінус) декремент: a-- рівносильно a = a - 1
```

Порядок запису інкременту відіграє дуже велику роль: пост-інкремент *var*++ повертає значення змінної *var* до виконання цього інкременту. Операція пре-інкремента ++*var* повертає значення зміненої змінної. Змінні бажано ініціалізувати, інакше вони можуть мати випадкове значення і в математичних операціях вийде непередбачуваний результат. Якщо змінна на момент початку виконання програми або після виклику функції (локальна змінна) повинна мати значення 0 – ініціалізуйте її як 0!

Операції з змінними і константами
Швидкість обчислень

Порядок обчислення виразів підпорядковується звичайним математичним правилам: спочатку виконуються дії у дужках, потім множення та ділення, і в кінці – додавання та віднімання.

Виконання обчислень займають у процесора деякий час, він залежить від типу дії та від типу даних, з якими дія провадиться. Потрібно розуміти, що не завжди і не у всіх випадках на дії витрачається стільки часу, скільки буде розказано далі: компілятор намагається по можливості оптимізувати обчислення. Оптимізовані обчислення займають дуже мало часу порівняно з неоптимізованими. Складно сказати, чи буде оптимізовано окреме обчислення у вашому коді, тому потрібно завжди готуватися до гіршого та знати, як краще робити. А саме:

- Arduino (на AVR) не має «хардверної» підтримки обчислень з плаваючою точкою (float), і ці обчислення проводяться за допомогою окремих інструментів і займають набагато більше часу, ніж з цілими типами;
- Чим «масивніше» тип даних, тим довше виконуються обчислення, тобто. дії з 1-байтними змінними виконуються швидше, ніж з 4-х байтними;
- Ділення (і пошук залишку від ділення) проводиться окремими інструментами (як операції з float), тому ця операція займає більше часу, ніж додавання/віднімання/множення. Для оптимізації швидкості обчислень є сенс замінювати ділення множенням на зворотне число (навіть на float).

Операції з змінними і константами
Час не оптимізованих компілятором обчислень різних типів даних (мкс, частота 16 МГц)

Тип даних	Час виконання, мкс		
	додавання та віднімання	множення	ділення, залишок
int8_t	0.44	0.625	14.25
uint8_t	0.44	0.625	5.38
int16_t	0.89	1.375	14.25
uint16_t	0.89	1.375	13.12
int32_t	1.75	6.06	38.3
uint32_t	1.75	6.06	37.5
float	8.125	10	31.5

Ця інформація дана виключно для ознайомлення і хвилюватися про швидкість обчислень не потрібно, т.я. більшість із них буде оптимізовано

Операції з змінними і константами
Цілісний поділ

При цілісному розподілі результат не округляється за «математичними» правилами:

- дрібна частина просто відсікається (тобто має місце бути округлення вниз): і 9/10 і 1/10 дадуть 0.

При використанні float само собою вийде 0.9 і 0.1. Для округлення за нормальними математичними правилами можна використовувати функцію *round()*, але вона досить важка, т.я. працює з *float*. Багато завдань можна вирішити за допомогою цілочисельної математики, що дуже оптимально за швидкістю виконання коду.

Операції з змінними і константами
Особливість великих обчислень

Для додавання і віднімання за замовчуванням використовується осередок *long* (4 байти), але при множенні та діленні використовується *int* (2 байти), що може призвести до непередбачуваних результатів!

Якщо при множенні чисел результат перевищує 32 768, він буде порахований некоректно. Для виправлення ситуації потрібно писати (*тип даних*) перед множенням, що змусить МК виділити додаткову пам'ять для обчислення

*(long)35*1000*

Також існую модифікатори, які роблять приблизно те саме.

- U – переведення у формат *unsigned int* (0 … 65535). Приклад: 36000u
- L – переведення у формат *long* (-2147483648 ... 2147483647). Приклад: 325646L
- UL – переведення у формат *unsigned long* (0 … 4294967295). Приклад: 361341ul

Операції з змінними і константами
Особливість *float*

Arduino підтримує роботу з числами з плаваючою точкою (десяткові дроби). Цей тип даних не має апаратної підтримки, а реалізований програмно, тому обчислення з ним виконуються у кілька разів довше, ніж із цілим типом. Крім повільних обчислень, підтримка роботи з *float* займає пам'ять, т.к. вона реалізована як «бібліотеки». Використання математичних операцій з *float* (* / + -) додає приблизно 1000 байт у flash пам'ять, одноразово, просто підключається інструмент для виконання дій. Arduino підтримує три типи введення чисел з плаваючою точкою:

Тип запису	Приклад	Чому дорівнює
Десяткова дроб	20.5	20.5
Науковий	2.34E5	2.34*10^5 або 234000
Інженерний	67e-12	67*10^-12 або 0.000000000067

Операції з змінними і константами
Особливість *float*

З обчисленнями є така особливість: якщо у виразі немає *float* чисел, то обчислення матимуть цілий результат (дрібна частина відсікається). Для отримання правильного результату потрібно писати перетворення (*float*) перед дією, використовувати *float* числа або *float* змінні. Також є модифікатор *f*, який можна використовувати лише до цифр *float*. Сенсу в ньому немає, але такий запис можна зустріти.

```
float val;          // далі присвоюватимемо 100/3, очікуємо результат 33.3333
val = 100/3;        // рахує НЕПРАВИЛЬНО (результат 33.0)
int val1 = 100;     // цілісна змінна
val = val1/3;       // рахує НЕПРАВИЛЬНО (результат 33.0)
float val2 = 100;   // float змінна
val = val2/3;       // рахує правильно (є змінна float)
val=(float)100/3;   // рахує правильно (вказуємо (float) )
val = 100.0/3;      // рахує правильно (є число float)
val = 100/3.0f;     // рахує правильно (є число float і модифікатор)
```

Операції з змінними і константами
Особливість *float*

Важливий момент: через особливості самої моделі «чисел з плаваючою точкою» - обчислення іноді виконуються з невеликою похибкою. Дивіться (значення виведені через порт):

```
float val2 = 1.1 - 1.0;   // val2 == 0.100000023 !!!
float val4 = 1.5 - 1.0;   // val4 == 0.500000000
```

Здавалося б, *val2* має дорівнювати 0.1 після віднімання, але у 8-му знаку вилізла похибка! Будьте дуже уважними при порівнянні *float* чисел, особливо зі строгими операціями <= : результат може бути некоректним та нелогічним.

Типи даних

Змінні оголошуються таким чином: <*тип даних*> <*ім'я*>;

```
int my_val; // оголосити змінну my_val
```

Також можна відразу надати значення: <*тип даних*> <*ім'я*> = <*значення*>;

```
int my_val = 2300; // оголосити змінну my_val і присвоїти їй число 2300
```

Також можна оголосити кілька змінних одного типу одразу

```
int my_val = 2300, my_val2, my_val4, lolkek = 5; // оголосити змінні
```

Типи даних

Назва	Альтернативна назва	Вага	Діапазон	Особливість
boolean		1 байт	0 або 1	Логічна змінна, може приймати значення true (1) і false (0)
char	int8_t	1 байт	-128 … 127	Зберігає номер символу з таблиці символів ASCII
byte	uint8_t	1 байт	0 … 255	
int	int16_t	2 байти	-32 768 … 32 767	
unsigned int	uint16_t	2 байти	0 … 65 535	
word		2 байти	0 … 65 535	Теж саме, що і unsigned int
long	int32_t	4 байти	-2 147 483 648 … 2 147 483 647	- 2 мільярди … 2 мільярди
unsigned long	uint32_t	4 байти	0 … 4 294 967 295	0 … 4 мільярди
float		4 байти	- 3.4028235E+38 … 3.4028235E+38	Зберігає числа з плаваючою крапкою (десяткові дроби). Точність 6-7 знаків
double		4 байти		Теж саме, що і float

Типи даних
Особливості використання змінних

- Уважно слідкуйте за значенням, яке приймає змінна. Якщо значення перевищить максимальне або принизить мінімальне (вийде з діапазону) для цього типу даних змінна скинеться в 0, а потім продовжить збільшення або зменшення в тому ж напрямку (наприклад, якщо присвоїти *byte* значення 300, то вона прийме 300-255=45). Таку помилку потім буде важко відстежити;
- Тип даних вказується при оголошенні змінної ТІЛЬКИ ОДИН РАЗ, далі змінна використовується чисто за ім’ям (звернення до змінної). При спробі змінити тип змінної (переоголосити змінну) ви отримаєте помилку. Але тільки в тому випадку, якщо змінна глобальна, або коли локальна переоголошується всередині функції, у якій вона вже була оголошена.

Типи даних
Особливості float

- Надавати тільки значення з точкою, навіть якщо воно ціле (10.0)
- Ділити також лише на числа з точкою, навіть якщо вони цілі (змінна / 2.0)
- При діленні цілого типу з метою отримати число з плаваючою точкою, писати (*float*) перед обчисленням!
- Операції з числами типу *float* займають набагато більше часу, ніж із цілими! Якщо потрібна висока швидкість обчислень, краще застосовувати всякі хитрощі, у стилі виконання всіх обчислень типом *long*, і результат уже переводити на *float*. Наприклад замість 3.25 обчислювати у 100 разів більші числа, тобто 325, а потім ділити на 100.

Типи змінних

- Глобальна змінна – оголошується поза функціями, наприклад на самому початку скетчу або між функціями. Звертатися до глобальної змінної (використовувати її значення) можна СКРІЗЬ.
- Локальна змінна – оголошується ВСЕРЕДИНІ функції, і звертатися до неї можна тільки всередині цієї функції.
- Локальних змінних може бути кілька з однаковим ім'ям, але різними значеннями. Це пов'язано з тим, що локальна змінна вивантажується з оперативної пам'яті мікроконтролера при виході із функції.

Константи

```
const <тип> <ім'я> = <значення>; - оголосити константу
const int my_val = 2300; // оголосити константу my_val і присвоїти їй число 2300

#define <ім'я> <значення> - оголосити константу через define (крапка з комою НЕ ПОТРІБНА)
#define my_val 2300 // визначити константу my_val і присвоїти їй число 2300
```

Константа, оголошена через #define, працює трохи інакше: на етапі компіляції коду вказана назва ЗАМІНЯЄТЬСЯ на вказане значення, і зберігається у флеш-пам'яті МК.

При спробі змінити значення константи ПІСЛЯ її оголошення ви отримаєте помилку!

Послідовний порт

На платах Arduino стоїть USB-TTL конвертер, що дозволяє мікроконтролеру в текстовому режимі «консолі» спілкуватися з комп'ютером за послідовним інтерфейсом, Serial. На комп'ютері створюється віртуальний COM порт, до якого можна підключитися за допомогою програм-терміналів порту, і приймати-надсилати текстові дані. Через цей порт завантажується прошивка, т.я. підтримка Serial є вбудованою в мікроконтролер на «залізному» рівні, і USB-TTL перетворювач підключений саме до цих виходів мікроконтролера. На платі Arduino Nano/Uno це піни D0 і D1.

Саме тому піни D0 та D1 на платах Nano/Uno не можна займати датчиками або підтягувати до живлення/землі на момент прошивки: мікроконтролер не зможе отримати дані і ви отримаєте помилку завантаження.

До цих пінів можна підключатися за допомогою окремих плат «програматорів», наприклад на чіпах CP2102 або тому ж CH340 з метою завантаження прошивки або просто спілкування з платою. У самій Arduino IDE також є вбудована консоль - монітор порту, кнопка з іконкою лупи в правому верхньому кутку програми. Натиснувши на цю кнопку, ми відкриємо сам монітор порту.

Послідовний порт. Объект Serial

Serial – це об'єкт класу *Stream*, що дозволяє як просто приймати/надсилати дані через послідовний порт, так і успадковує з класу Stream багато можливостей.

```
Serial.begin(speed)
```

Запустити зв'язок по *Serial* на швидкості *speed* (baud rate, біт за секунду). Швидкість можна поставити будь-яку, але є кілька стандартних. Список швидкостей для монітора порту Arduino IDE:

- 300
- 1200
- 2400
- 4800
- **9600** найчастіше використовується, можна назвати стандартною для більшості пристроїв зі зв'язком через TTL
- 19200
- 38400
- 57600
- 115200 теж часто зустрічається
- 230400
- 250000
- 500000
- 1000000
- 2000000

Послідовний порт. Объект Serial

Serial.end() – припинити зв'язок із Serial. Для UNO/NANO (ATmega328) це дозволяє звільнити піни 0 та 1 для звичайних цілей (читання/запис).

Serial.available() – повертає кількість байт, що зберігаються в буфері (обсяг буфера 64 байта) та доступні для читання.

Serial.availableForWrite() – повертає кількість байт, які можна записати до буфера послідовного порту, не блокуючи при цьому функцію запису.

Serial.write(val), Serial.write(buf, len) – відправляє до порту val чисельне значення або рядок, або відправляє кількість len байт із буфера buf. Важливо! Відправляє дані як байт (див. таблицю ASCII), тобто надіславши 88, ви отримаєте літеру X: Serial.write(88); виведе літеру X.

Serial.print(val), Serial.print(val, format) – відправляє в порт значення val – число чи рядок. На відміну від write виводить саме символи, тобто. відправивши 88 ви отримаєте 88: Serial.print(88); виведе 88. Також метод print/println має кілька налаштувань для різних даних, що робить його дуже зручним інструментом налагодження:

```
Serial.print(78); // виведе 78
Serial.print(1.23456); // 1.23 (за замовчуванням. 2 знаки)
Serial.print('N'); // виведе N
Serial.print("Hello world."); // Hello world.

// можна зробити форматований вивід у стилі
Serial.print("i have" + String(50) + "apples");
// виведе рядок i have 50 apples

// замість чисел можна вставляти змінні
byte appls = 50;
Serial.print("i have" + String(appls) + "apples");
// виведе те саме
```

Послідовний порт. Объект Serial

format дозволяє налаштовувати виведення даних: BIN, OCT, DEC, HEX виведуть число у відповідній системі обчислення, а цифра після виведення float дозволяє налаштовувати кількість знаків, що виводиться після точки

```
Serial.print(78, BIN); // виведе на екран "1001110"
Serial.print(78, OCT); // виведе на екран "116"
Serial.print(78, DEC); // виведе на екран "78"
Serial.print(78, HEX); // виведе на екран "4E"
Serial.print(1.23456, 0); // виведе на екран "1"
Serial.print(1.23456, 2); // виведе на екран "1.23"
Serial.print(1.23456, 4); // виведе на екран "1.2345"
```

Serial.println(), Serial.println(val), Serial.println(val, format) – повний аналог print(), але автоматично переводить рядок після виводу. Дозволяє також викликатися без аргументів (з порожніми дужками) просто для переведення рядка

Serial.flush() – чекає на закінчення передачі даних

Serial.peek() – повертає поточний байт із краю буфера, не прибираючи його з буфера. При виклику Serial.read() буде прочитано той же байт, але з буфера вже забереться

Serial.read() – читає та повертає байт як код символу з таблиці ASCII. Якщо потрібно повернути цифру, робимо Serial.read() - '0'

Serial.setTimeout(time) – встановлює time (мілісекунди) таймаут очікування прийому даних для наступних функцій. За промовчанням дорівнює 1000 мс (1 секунда)

Serial.find(target), Serial.find(target, length) – читає дані з буфера та шукає набір символів target (тип char), опціонально можна вказати довжину length. Повертає true, якщо знаходить вказані символи. Чекає на передачу по таймуту

Послідовний порт. Объект Serial

```
// будемо шукати слово KhNUE
char target[] = "KhNUE";

void setup() {
  Serial.begin(9600);
}

void loop() {
  if (Serial.available() > 0) {
    if (Serial.find(target))
      Serial.println("found");
    // вивести found, якщо було надіслано
  }
}
```

Serial.findUntil(target, terminal) – читає дані з буфера та шукає набір символів target (тип char) або термінальний рядок terminal. Очікує закінчення передачі по таймауту або завершує прийом після читання terminal

Serial.readBytes(buffer, length) – читає дані з порту та закидає їх у буфер buffer (масив char [] або byte []), також вказується кількість байт, яку потрібно записати – length (щоб не переповнити буфер)

Serial.readBytesUntil(character, buffer, length) – читає дані з порту та закидає їх у буфер buffer (масив char [] або byte []), також вказується кількість байт, які потрібно записати – length (щоб не переповнити буфер) та термінальний символ character. Закінчення прийому в buffer відбувається при досягненні заданої кількості length, при прийомі термінального символу character (він у буфер не йде) або по таймауту

Serial.readString() – читає буфер порту та формує з даних рядок String, який повертає. Закінчує роботу з таймауту

Serial.readStringUntil(terminator) – читає буфер порту та формує з даних рядок String, який повертає. Закінчує роботу з таймауту або після прийому символу terminator (символ char)

Плоттер

Крім монітора послідовного порту, в Arduino IDE є плоттер – будівник графіків у реальному часі за даними послідовного порту. Достатньо відправити значення за допомогою команди Serial.println (значення) і відкрити плоттер по послідовному з'єднанню.

Побудуємо графік значення з аналогового піна A0:

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  Serial.println(analogRead(0));
  delay(10);
}
```

Плоттер підтримує кілька ліній графіків одночасно, для їх відображення потрібно дотримуватися наступного протоколу відправки даних: значення1 пробіл_або_кома значення2 пробіл_або_кома значення3 пробіл_або_кома значенняN перенос_рядка, тобто значення виводяться в один рядок, одне за іншим по порядку, поділяються пробілом або комою, і в кінці обов’язково перенос рядку.

Плоттер

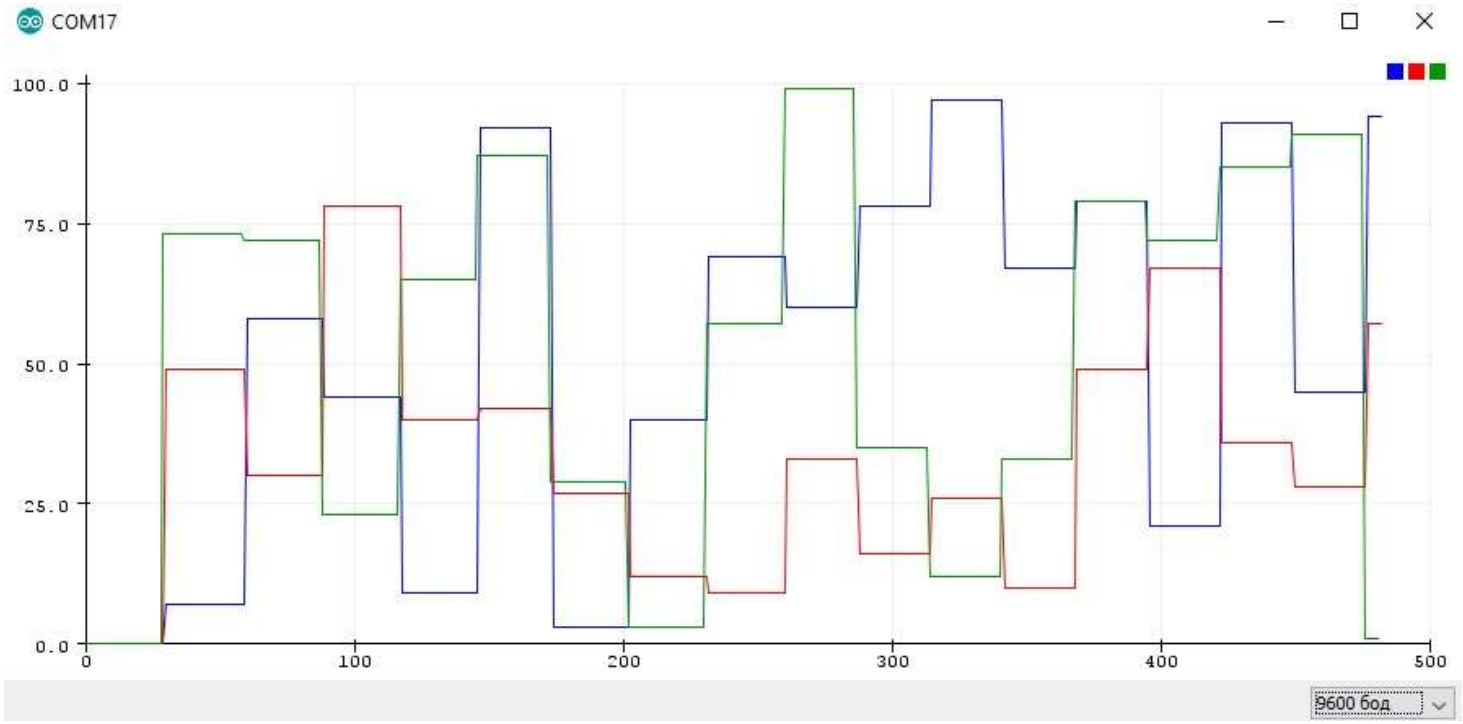
```
void setup() {
  Serial.begin(9600);
}

byte val1, val2, val3;
uint32_t timer;

void loop() {
  // кожні 300 мс
  if (millis() - timer >= 300) {
    timer = millis();
    val1 = random(100);
    val2 = random(100);
    val3 = random(100);
  }

  // вивід кожні 10 мс
  Serial.print(val1);
  Serial.print(' ');
  Serial.print(val2);
  Serial.print(' ');
  Serial.print(val3);
  Serial.println(' ');
  delay(10);
}
```

Виведення значень відбувається кожні 10 мілісекунд, а кожні 300 мілісекунд значення оновлюються. Отримуємо такий графік:



Лекцію закінчено
Дякую за увагу