**EMBEDDED SYSTEM PROGRAMMING**

**1ST ASSIGNMENT**

****

**INFORMATICS ENGINEERING**

**FACULTY OF COMMUNICATION AND INFORMATICS**

**MUHAMMADIYAH UNIVERSITY OF SURAKARTA**

**2017**

**BY**

**FILDZA NUR SABRINA**

**L200150145**

Arduino programs can be divided in three main parts: **structure**, **values** (variables and constants), and **functions**.

1. **STRUCTURE**
2. Setup()­­

When creating some code, the setup() is called. It use to define the variables, pin modes, start using libraries, etc. The function will only be runned once. After the start mode or reset of Arduino board.

//Saat membuat code, fungsi setup() dipanggil. Fungsi ini berfungsi untuk mendefinisikan vaiabel, mode pin, menggunakan library, dll. Fungsi ini hanya bisa di eksekusi sekali, setelah mode mulai atau reset dari Arduino.

1. Loop()

After the setup() function, that define and make the initial values, the loop() function does precisely what its name suggests, and loops consecutively, that will let your program to change and respond.

//Setelah fungsi setup(), yang mendefinisikan dan membuat inisial value, fungsi loop() melakukan persis dengan sugesti nya, dan membiarkan program anda untuk mengubah dan merespon.

* 1. CONTROL STRUCTURE(S)

1. If

Used in conjuction with comparison provider, tests whether a several condition has been achieved, such as an input being above a certain number.

//Digunakan sebagai operator perbandingan untuk mencapai suatu kondisi tertentu.

1. If/else

This statement will allow you to control over the flow of code than the regular **if** statement, by letting multiple tests to be grouped together.

//command ini akan mengizinkan pengguna untuk mengendalikan alur kode dibandingkan dengan fungsi **if** biasanya, dengan membiarkan beberapa test dikumpulkan menjadi kelompok tertentu.

1. For

Used to repeat the group of statements enclosed in curly braces. The **for** statement is useful for several operations, and its often used to combine with arrays on collection of data/pins.

//Digunakan untuk mengulangi kelompok statement dan ditutup dengan kurung kurawal. Fungsi ini digunakan pada beberapa operasi, dan kadang digunakan untuk menggabungkan array ke kelompok data/pin.

1. Switch/case

This statement has the same function to **if** statement. The **switch/case** is used to control the programs by let the programmers to specify different code that should be executed in several conditions.

//Statement ini mempunyai fungsi yang sama dengan statement **if**. Fungsi **switch/case** berfungsi untuk mengendalikan program dengan mengizinkan programmer untuk mengkategorikan beberapa kode yang bisa di eksekusi di kondisi yang berbeda.

1. While loops

While loops will loop continuously, or in other word: forever, infinitely, until the expression inside the parenthesis() become false.

//While loop akan berjalan secara terus-menerus, dengan kata lain: selamanya, infinit, sampai program didalam parentheses() dinyatakan salah.

1. Do-while

The **do** loop works in the same field with **while** loop, but with the exception that the condition is tested at the end of the loop. So, at least the  **do** loop will always run once in one program.

//Loop **do** bekerja di bidang yang sama dengan **while**, tapi dengan pengecualian bahwa kondisi ini akan ditest di akhir loop. Jadi **do** akan selalu bekerja setidaknya sekali dalam satu program.

1. Break

This **break** statement will let you to exit or stop the **do**, **for**, or **while loop** bypassing the normal loop condition.

//Statement **break** akan mengizinkan pengguna untuk keluar atau memberhentikan **do**, **for**, atau **while loop** dengan melompati kondisi loop normal.

1. Continue

This statement will skip the rest of the current iteration of a loop.

//Pernyataan ini melewatkan sisa dari iterasi saat ini loop (lakukan, untuk, atau saat). Dengan memeriksa ekspresi kondisional dari loop, dan melanjutkan dengan iterasi berikutnya.

1. Return

Terminate a function and return a value froma function to the calling function.

//Menghentikan fungsi dan mengembalikan nilai dari fungsi ke fungsi panggilan, jika diinginkan

1. Goto

Tranfers program flow to a labeled point in the program.

//Untuk keluar dari *loop* dan *nested loop* dengan lebih mudah dan ringkas.

* 1. FURTHER SYNTAX

1. ; (semicolon)

Used to end a statement.

// Digunakan setelah berakhirnya *statement*

1. {} (curly braces)

Used in several kind of different programs, outlined, and sometimes beginners confused to it.

//Mendefinisikan awal dan akhir dari sebuah blok fungsi

1. // (single line comment)

Used to inform programmers about the way the program works.

//Sama dengan *multi-line coment* hanya saja perbaris.

1. /\*\*/ (multi line comment)

Comments are lines in the program that are used to inform yourself or others about the way the program works. They are ignored by the compiler, and not exported to the processor, so they don't take up any space on the Atmega chip.

1. #define

Defined constants in arduino don't take up any program memory space on the chip. The compiler will replace references to these constants with the defined value at compile time.

//Komponen yang digunakan untuk memberi nama pada nilai konstanta sebelum program dikompilasi

1. #include

used to include outside libraries in your sketch. This gives the programmer access to a large group of standard C libraries (groups of pre-made functions), and also libraries written especially for Arduino.

// Memasukkan sketsa program ke dalam *libraries*.

* 1. ARITHMETIC OPERATORS

1. = (assignment operator)

Single equal sign

//persamaan

1. + (addition)

Penambahan

1. – (subtraction)

Pengurangan

1. \* (multiplication)

Perkalian

1. / (division)

Pembagian

1. % (modulo)

Hasil sisa bagi dua

* 1. COMPARISON OPERATORS

1. == (equal to)

Persamaan (sama dengan)

1. != (not equal to)

Tidak sama dengan

1. < (less then)

Kurang dari

1. > (greater than)

Lebih dari

1. <= (less than or equal to)

Lebih besar sama dengan

1. >= (greater than or equal to)

Lebih kecil sama dengan

* 1. BOOLEAN OPERATORS

1. && (and)

True only if both operands are true

//Ouput akan bernilai TRUE jika kedua operand bernilai TRUE.

1. || (or)

True if either operand is true

// Output akan bernilai FALSE jika kedua operand bernilai FALSE.

1. ! (not)

True if the operand is false

//Operator NOT menyatakan kebalikan nya. Contoh jika input bernilai FALSE output akan bernilai TRUE.

* 1. POINTER ACCESS OPERATORS

1. \* dereference operator

Pointers are one of the more complicated subjects for beginners in learning C, and it is possible to write the vast majority of Arduino sketches without ever encountering pointers

//Address of atau alamat dari variable/fungsi tersebut di dalam memory cel

1. & reference operator

Value pointed by atau nilai dari variable yang direferensikan

* 1. BITWISE OPERATORS

1. & (bitwise and)

The bitwise operators perform their calculations at the bit level of variables. They help solve a wide range of common programming problems

1. | (bitwise or)

The bitwise OR operator in C++ is the vertical bar symbol, |. Like the & operator, | operates independently each bit in its two surrounding integer expressions, but what it does is different

1. ^ (bitwise xor)

There is a somewhat unusual operator in C++ called bitwise EXCLUSIVE OR, also known as bitwise XOR.

1. ~ (bitwise not)

The bitwise NOT operator in C++ is the tilde character ~

1. << (bitshift left)

Used to operate the change of binary code to the left.

Digunakan untuk operator pergeseran nilai biner ke kiri.

Contoh : int a = 00000111

int b = a << 3

int c = b << 3

artinya b = a digeser kekiri 3 bit atau b = 00111000

1. >> (bithshift right)

Used to operate the change of binary code to the right.

Digunakan untuk operator pergeseran nilai biner ke kanan.

Contoh : int a = 00000111

int b = a >> 2

int c = b >> 3

artinya b = a digeser kekanan 2 bit atau b = 11000001

* 1. COMPOUND OPERATORS

1. ++ (increment)

Kenaikan variable. Menambahkan suatu nilai variable.

1. – (decrement)

Penurunan variable. Mengurangi suatu nilai variable.  
Melakukan operasi matematika pada variabel dengan konstanta lain atau variabel

1. += (compound addition)

Group of addition

Penjumlahan gabungan

1. -= (compound substraction)

Group of substraction

Pengurangan gabungan

1. \*= (compound multiplication)

Group of multiplication

Perkalian gabungan

1. /= (compound division)

Group of division

Pembagian gabungan

1. %= (compound modulo)

Group of modulo

Gabungan hasil sisa bagi 2

1. &= (compound bitwise and)

Often used with variables and constanta to force bits in a variable in LOW (0) state.

Seringkali digunakan dengan variable dan konstanta untuk memaksa bit tertentu dalam variable untuk kondisi LOW (0).

1. |= (compound bitwise or)

Used with variable and constanta to control (set to 1) several bits in a variable.

//digunakan dengan variable dan konstanta untuk “mengatur” (set ke 1) bit tertentu dalam variable.

1. **VARIABLES**

2.1 CONSTANTS

* 1. HIGH | LOW

**HIGH**

When the configured pin used as input with pinMode(), and read with digitalRead(), Arduino(Atmega) will give a HIGH, if:

* Tension is over than 3.0V on pin (5V boards);
* Tension is over than 2.0V on pin (3.3V boards);

A pin could be configured as an INPUT in pinMode(), then make a HIGH with digitalWrite(). This condition will active the internal 20K resistor pullup, that will pull the pin input to read HIGH except the external circuit pulled to LOW state. This is how the INPUT\_PULLUP works. When pin is configured to OUTPUT with pinMode() function, and controled to be HIGH with digitalWrite(), pin will be in several states:

* 5Volts (5V boards);
* 3.3Volts (3.3V boards);

//Hanya ada dua kemungkinan untuk mengatur nilai *digital pin* yaitu : HIGH atau LOW

***HIGH***

Ketika pin dikonfigurasi sebagai INPUT dengan pinMode (), dan membaca dengan digitalRead (), Arduino (Atmega) akan memberikan nilai *HIGH* jika:

* tegangan lebih besar dari 3.0V pada pin (5V boards);
* tegangan lebih besar dari 2.0V pada pin (3.3V boards);

Sebuah pin juga dapat dikonfigurasi sebagai INPUT dengan pinMode (), dan kemudian membuat *HIGH* dengan digitalWrite (). Hal ini akan mengaktifkan internal 20K resistor pullup, yang akan menarik pin input untuk membaca *HIGH* kecuali ditarik menjadi *LOW* oleh sirkuit eksternal. Ini adalah bagaimana INPUT\_PULLUP bekerja. Ketika pin dikonfigurasi ke OUTPUT dengan fungsi pinMode (), dan diatur menjadi *HIGH* dengan digitalWrite (), pin berada pada tegangan:

* 5 volt (5V boards);
* 3,3 volt (3.3V boards);

**LOW**

LOW mode could be state as different kinds depending on what pin that controlled as an INPUT or OUTPUT. When pin configured as an INPUT with pinMode(), and read it with digitalRead() function, Arduino(Atmega) will be stated as LOW if:

* Tension is lower than 1.5V on pin (5V boards);
* Tension is lower than 1.0V (Approx) on pin (3.3V boards);

When pin configured to OUTPUT with pinMode(), and controlled to LOW with digitalWrite(), pin on the 0 volt (5V and 3.3V boards). This will happen when LED that was connected through resistor to 5 volt (or 3.3 volt).

// Nilai LOW dapat dinyatakan secara berbeda tergantung dari apakah pin diatur sebagai INPUT atau OUTPUT. Ketika pin dikonfigurasi sebagai INPUT dengan pinMode (), dan membaca dengan fungsi digitalRead (), Arduino (Atmega) akan menyatakan LOW jika:

* tegangan kurang dari 1.5V pada pin (5V boards);
* tegangan kurang dari 1.0V (Approx) pada pin (3.3V boards);

Ketika pin dikonfigurasi untuk OUTPUT dengan pinMode (), dan diatur untuk LOW dengan digitalWrite (), pin berada pada 0 volt (baik 5V dan 3.3V papan). Ini dapat terjadi misalnya cahaya LED yang terhubung melalui resistor seri untuk 5 volt (atau 3,3 volt).

* 1. INPUT | OUTPUT | INPUT\_PULLUP

**INPUT**

Arduino (Atmega) pins configured as **INPUT** with pinMode() are said to be in a high-impedance state. Pins configured as INPUT make extremely small demands on the circuit that they are sampling, equivalent to a series resistor of 100 Megohms in front of the pin. This makes them useful for reading a sensor.

// Arduino (Atmega) pin dikonfigurasi sebagai INPUT dengan pinMode () dinyatakan memiliki *high-impedance state*. Pin yang dikonfigurasi sebagai INPUT membuat *demands* yang sangat kecil pada sirkuit yang mereka coba, setara dengan resistor seri 100 megaohms di depan pin. Hal ini membuat mereka berguna untuk membaca sensor.

**OUTPUT**

Pins configured as **OUTPUT** with pinMode() are said to be in a low-impedance state. This means that they can provide a substantial amount of current to other circuits. Atmega pins can source (provide current) or sink (absorb current) up to 40 mA (milliamps) of current to other devices/circuits. This makes them useful for powering LEDs because LEDs typically use less than 40 mA. Loads greater than 40 mA (e.g. motors) will require a transistor or other interface circuitry.

// Pin yang dikonfigurasi sebagai OUTPUT dengan pinMode () dinyatakan memilki *low-impedance state*. Ini berarti bahwa mereka dapat memberikan sejumlah besar arus untuk sirkuit lainnya. Atmega pin dapat memberikan arus menyerap arus hingga 40 mA (milliamps) dari arus untuk perangkat lain / sirkuit. Hal ini membuat mereka berguna untuk menyalakan LED karena LED biasanya menggunakan kurang dari 40 mA. Beban lebih dari 40 mA akan membutuhkan transistor atau sirkuit antarmuka lainnya.

**INPUT\_PULLUP**

The Atmega microcontroller on the Arduino has internal pull-up resistors (resistors that connect to power internally) that you can access. If you prefer to use these instead of external pull-up resistors, you can use the **INPUT\_PULLUP** argument in pinMode().

**//** Atmega mikrokontroler pada Arduino memiliki resistor pull-up internal (resistor yang terhubung ke daya internal) yang dapat Anda akses. Jika Anda memilih untuk menggunakan ini dibandingkan eksternal resistor pull-up, Anda dapat menggunakan argumen INPUT\_PULLUP di pinMode ().

* 1. LED\_BUILTIN

Most Arduino boards have a pin connected to an on-board LED in series with a resistor. The constant LED\_BUILTIN is the number of the pin to which the on-board LED is connected. Most boards have this LED connected to digital pin 13.

// Kebanyakan Arduino boards memilki sebuah pin yang terhubung ke LED secara seri dengan sebuah resistor. Konstanta LED\_BUILTIN adalah angka pin yang menunjukan LED mana yang terhubung. Kebanyakan *boards* memiliki LED yang terhubung dengan *digital pin* 13

* 1. True | false

**FALSE**

false is the easier of the two to define. false is defined as 0 (zero).

//false lebih mudah didefinisikan. False didefinisikan sebagai 0.

**TRUE**

true is often said to be defined as 1, which is correct, but true has a wider definition. Any integer which is non-zero is true, in a Boolean sense. So -1, 2 and -200 are all defined as true, too, in a Boolean sense.

//true lebih sering didefinisikan sebagai 1, yang artinya benar, namun ‘true’ mempunyai arti yang lebih luas. Integer apapun yang bukan 0 adalah benar, di dalam Boolean. Jadi, -1,2 dan -200 merupakan true di dalam Boolean.

* 1. Integer constants

Integer constants are numbers used directly in a sketch, like 123. Normally, integer constants are treated as base 10 (decimal) integers, but special notation (formatters) may be used to enter numbers in other bases.

//Konstanta integer merupakan bilangan yang digunakan untuk menggambarkan sesuatu, seperti 123. Normalnya, konstanta integer digunakan sebagai dasar 10 integer, tapi notasi special (formatters) bisa digunakan untuk memasukkan angka di dasar lain.

* 1. Floating point constants

Similar to integer constants, floating point constants are used to make code more readable. Floating point constants are swapped at compile time for the value to which the expression evaluates.

//sama seperti konstanta integer, konstanta floating points digunakan untuk membuat kode yang lebih mudah untuk dibaca. Konstanta floating point akan dilewati saat compiling untuk membuat value yang bisa dikerjakan.

2.2 DATA TYPES

1. Void

used only in function declarations. It indicates that the function is expected to return no information to the function from which it was called.

//digunakan untuk mendeklarasikan fungsi. Hal ini mengindikasikan bahwa fungsi akan di ekspektasikan untuk tidak mengembalikan informasi apapun kedalam fungsi dari apa yang sudah dipanggil.

1. Boolean

A **boolean** holds one of two values, [true](https://www.arduino.cc/en/Reference/Constants) or [false](https://www.arduino.cc/en/Reference/Constants).

//Boolean memiliki dua value, true atau false.

1. Char

A data type that takes up 1 byte of memory that stores a character value. Character literals are written in single quotes, like this: 'A' (for multiple characters - strings - use double quotes: "ABC").

// tipe data untuk menyatakan suatu karakter seperti “A”, “B”, “C” , “+” , “.” dan sebagainya. Karakter ini disimpan dalam bentuk angka. Untuk mengkonversikan bentuk angka ke bentuk karakter dapat menggunakan tabel ASCII

1. Unsigned char

Same as char type but without a negative value so unsigned char has more than 0 to 255 value or equal to byte data.

//serupa dengan tipe char tetapi tanpa nilai negatif sehingga unsigned char memiliki nilai dari 0 hingga 255 atau sama dengan tipe data byte.

1. Byte

Has 8 bit data (0-255). Data byte type does not have a negative value.

//Memiliki 8 bit data (0 – 255). Tipe data byte tidak memiliki nilai negatif

1. Int

Used to state integer data type (decimal). Integer data type is about -32768 to 32768 (-215 to (215-1)). This value need 2 byte microcontroller memory.

//Digunakan untuk menyatakan tipe data integer (bilangan bulat). Tipe data integer berkisar antara -32768 sampai 32768 (-215 hingga (215-1)). Nilai bilangan ini membutuhkan 2 byte memori mikrokontroller.

1. Unsigned int

Used to define positive decimal data type or more than 0 value to 65535. Same as integer, this data type also needs 2 byte microcontroller memory.

//Digunakan untuk mendeklarasikan tipe data bilangan bulat positif atau bernilai 0 hingga 65535. Sama dengan integer, tipe data ini juga membutuhkan 2 byte memori mikrokontroller.

1. Word

Same as unsigned int data type. The data word has 16 bit or needs 2 byte microcontroller memory. This data type is rarely used.

//Sama dengan tipe data unsigned int. Besar data word adalah 16 bit atau membutuhkan 2 byte memori mikrokontroller. Tipe data ini jarang digunakan.

1. Long

A data type to collect decimal numbers in range -2.147.438.648 to 2.147.483.647. A long typed variables written with an L at the end of it. For example 456789980L.

//Tipe data untuk menampung bilangan bulat yang berkisar antara -2.147.483.648 hingga 2.147.483.647. variabel bertipe long ditulis dengan akhiran L atau l. Contoh 456789980L.

1. Unsigned long

Same as long data type, but counted from 0or has more than 0 to 4.292.967.295 value. Variables written with an UL at the end of the constants.

//Tipe data yang sama dengan long, tetapi dihitung dari angka 0 atau mempunyai nilai berkisar 0 hingga 4.292.967.295. Nilai variabel ini ditulis dengan kode UL diakhir konstanta

1. Short

Arduinos with Atmega and ARM based, contains a short with 16-bit(2 byte) value.

//Semua Arduino berbasis ATMega dan ARM, menyimpan sebuah short bernilai 16-bit (2-byte)

1. Float

Uses to save some real number. That could be saved from -3,4028235 x 1038 to 3,4028235 x 1038. A float typed number is huge, so it rarely to used or it will slow down the microcontroller processor’s work.

//Untuk menyimpan bilangan real. Angka yang bisa disimpan dari -3,4028235 x 1038 hingga 3,4028235 x 1038. Angka dengan tipe float sangat besar sekali, sehingga sangat jarang digunakan karena akan memperlambat kerja prosessor mikrokontroller.

1. Double

A data type that has no difference with float.

//tidak ada perbedaan antara double dan float.

1. String – char array

Arduino system is support a string data type in a group of character.

//Sistem Arduino mendukung tipe data string dalam bentuk kumpulan karakter.

1. String – object

It can used to use a dynamic memory that could occure some debug.

// Penggunaan string-object ini dapat menghabiskan *dynamic memory* yang dapat memunculkan *debug.*

1. Array

A group of variables that could be accessd with indexing.

// kumpulan variabel yang diakses dengan nomor indeks.

* 1. **CONVERSION**

1. Char()

Change a type value to char data

// merubah suatu nilai ke tipe data char

1. Byte()

Change a type value to byte data

// merubah suatu nilai ke tipe data byte

1. Int()

Change a type data to int data type

// merubah suatu nilai ke tipe data int

1. Word()

Change some value indo a word data type or make some word from 2 bytes

// merubah suatu nilai ke tipe data word atau membuat suatu *word* dari 2bytes

1. Long()

Change some values into a long data type

// merubah suatu nilai ke tipe data long

1. Float()

Change some values into a float data type

// merubah suatu nilai ke tipe data float

**2.4 VARIABLE SCOPE & QUALIFIERS**

1. Variable scope

Variables in the C programming language, which Arduino uses, have a property called scope. This is in contrast to early versions of languages such as BASIC where every variable is a global variable.

A global variable is one that can be seen by every function in a program. Local variables are only visible to the function in which they are declared. In the Arduino environment, any variable declared outside of a function (e.g. setup(), loop(), etc. ), is a global variable.

When programs start to get larger and more complex, local variables are a useful way to insure that only one function has access to its own variables. This prevents programming errors when one function inadvertently modifies variables used by another function.

It is also sometimes handy to declare and initialize a variable inside a for loop. This creates a variable that can only be accessed from inside the for-loop brackets.

//Variabel dalam bahasa pemrograman C, yang digunakan Arduino, memiliki sifat yang disebut lingkup. Hal ini berbeda dengan versi awal bahasa seperti BASIC dimana setiap variabel adalah variabel global.

Sebuah variabel global adalah salah satu yang dapat dilihat oleh setiap fungsi dalam sebuah program. Variabel lokal hanya dapat dilihat didalam fungsi. Dalam lingkungan Arduino, setiap variabel dinyatakan di luar fungsi (misalnya setup (), lingkaran (), dll), adalah variabel global.

Ketika program menjadi lebih kompleks, variabel lokal adalah cara yang berguna untuk memastikan bahwa hanya satu fungsi yang memiliki akses ke variabelnya sendiri. Hal ini untuk mencegah kesalahan pemrograman ketika salah satu fungsi secara tidak sengaja memodifikasi variabel yang digunakan oleh fungsi lain.

Hal ini juga kadang-kadang berguna untuk mendeklarasikan dan menginisialisasi variabel untuk loop. Hal ini menciptakan sebuah variabel yang hanya dapat diakses dari dalam kurung loop.

1. Static

The static keyword is used to create variables that are visible to only one function. However unlike local variables that get created and destroyed every time a function is called, static variables persist beyond the function call, preserving their data between function calls. Variables declared as static will only be created and initialized the first time a function is called

//Digunakan untuk membuat variabel yang hanya terlihat oleh satu fungsi. Namun tidak seperti variabel lokal yang bisa diciptakan dan dihancurkan setiap kali fungsi dipanggil, variabel static bertahan melampaui panggilan fungsi, melestarikan data mereka diantara fungsi yang dipanggil.

Variabel yang dideklarasikan sebagai static hanya akan dibuat dan diinisialisasi pada saat fungsi pertama kali dipanggil.

1. Volatile

volatile is a keyword known as a variable qualifier, it is usually used before the datatype of a variable, to modify the way in which the compiler and subsequent program treats the variable.

// volatile merupakan kata kunci yang dikenal sebagai pengkualifikasi variabel, biasanya digunakan sebelum tipe data dari suatu variabel, untuk memodifikasi cara memperlakukan variabel pada compiler dan program selanjutnya.

1. Const

The **const** keyword stands for constant. It is a variable qualifier that modifies the behavior of the variable, making a variable "read-only". This means that the variable can be used just as any other variable of its type, but its value cannot be changed. You will get a compiler error if you try to assign a value to a **const** variable.

//ini adalah pengkualifikasi variabel yang mengubah perilaku dari variabel, membuat variabel menjadi "read-only"yang artinya nilai variabel tidak dapat diubah. Anda akan mendapatkan kesalahan kompilator jika Anda mencoba untuk memberikan nilai pada variabel const

**2.5 UTILITIES**

1. Sizeof()

The size of operator returns the number of bytes in a variable type, or the number of bytes occupied by an array.

// mengembalikan jumlah byte dalam jenis variabel, atau jumlah byte yang ditempati oleh sebuah array.

1. PROGMEM

Store data in flash (program) memory instead of SRAM. The PROGMEM keyword is a variable modifier, it should be used only with the datatypes defined in pgmspace.h. It tells the compiler "put this information into flash memory", instead of into SRAM, where it would normally go.

//menyimpan data dalam flash (program) memori bukan SRAM. PROGMEM adalah pengubah variabel, harus digunakan hanya dengan tipe data yang didefinisikan dalam pgmspace.h. Ini memberitahu compiler "menaruh informasi ini ke dalam flash memory", bukan ke SRAM, di mana ia biasanya akan pergi. PROGMEM merupakan bagian dalam library pgmspace.h yang tersedia dalam arsitektur AVR saja.

1. **FUNCTION**

**3.1 DIGITAL I/O**

* 1. pinMode()

Configures the specified pin to behave either as an input or an output. See the description of digital pins for details on the functionality of the pins.

// Mengkonfigurasi pin yang ditentukan untuk berperilaku baik sebagai input atau output. Lihat deskripsi pin digital untuk rincian tentang fungsi pin

* 1. digitalWrite()

Write a HIGH or a LOW value to a digital pin.

// Tulis HIGH atau nilai LOW ke pin digital

* 1. digitalRead()

Reads the value from a specified digital pin, either HIGH or LOW.

// Membaca nilai dari pin digital ditentukan, baik HIGH atau LOW.

**3.2 ANALOG I/O**

1. analogReference()

Configures the reference voltage used for analog input (i.e. the value used as the top of the input range). The options are:

* DEFAULT: the default analog reference of 5 volts (on 5V Arduino boards) or 3.3 volts (on 3.3V Arduino boards)
* INTERNAL: an built-in reference, equal to 1.1 volts on the ATmega168 or ATmega328 and 2.56 volts on the ATmega8 (*not available on the Arduino Mega*)
* INTERNAL1V1: a built-in 1.1V reference (*Arduino Mega only*)
* INTERNAL2V56: a built-in 2.56V reference (*Arduino Mega only*)
* EXTERNAL: the voltage applied to the AREF pin (**0 to 5V only**) is used as the reference.

//Mengkonfigurasi tegangan referensi yang digunakan untuk input analog (yaitu nilai yang digunakan sebagai bagian atas berbagai masukan). Pilihannya adalah:

DEFAULT: referensi analog default 5 volt (pada papan 5V Arduino) atau 3,3 volt (di papan 3.3V Arduino)

* INTERNAL: built-in referensi, sama dengan 1,1 volt pada ATmega168 atau ATmega328 dan 2,56 volt pada ATmega8 (tidak tersedia pada Mega Arduino)
* INTERNAL1V1: built-in referensi 1.1V (Arduino mega saja)

INTERNAL2V56: built-in referensi 2.56V (Arduino mega saja)  
EKSTERNAL: tegangan diterapkan pada pin AREF (0 5V hanya) digunakan sebagai referensi.

1. analogRead()

Reads the value from the specified analog pin. The Arduino board contains a 6 channel (8 channels on the Mini and Nano, 16 on the Mega), 10-bit analog to digital converter. This means that it will map input voltages between 0 and 5 volts into integer values between 0 and 1023. This yields a resolution between readings of: 5 volts / 1024 units or, .0049 volts (4.9 mV) per unit. The input range and resolution can be changed using analogReference().

It takes about 100 microseconds (0.0001 s) to read an analog input, so the maximum reading rate is about 10,000 times a second.

// Membaca nilai dari pin analog yang ditentukan. Papan Arduino berisi 6 channel (8 saluran pada Mini dan Nano, 16 di Mega), 10-bit analog ke digital. Ini berarti bahwa itu akan memetakan tegangan masukan antara 0 dan 5 volt menjadi nilai integer antara 0 dan 1023. Ini menghasilkan resolusi antara pembacaan dari: 5 volt / 1024 unit atau, 0,0049 volt (4,9 mV) per unit. Berbagai masukan dan resolusi dapat diubah dengan menggunakan analogReference ().  
  
Dibutuhkan sekitar 100 mikrodetik (0,0001 s) untuk membaca input analog, sehingga tingkat membaca maksimum adalah sekitar 10.000 kali per detik.

1. analogWrite()

Writes an analog value (PWM wave) to a pin. Can be used to light a LED at varying brightnesses or drive a motor at various speeds. After a call to **analogWrite()**, the pin will generate a steady square wave of the specified duty cycle until the next call to **analogWrite()** (or a call to **digitalRead()** or **digitalWrite()** on the same pin). The frequency of the PWM signal on most pins is approximately 490 Hz. On the Uno and similar boards, pins 5 and 6 have a frequency of approximately 980 Hz. Pins 3 and 11 on the Leonardo also run at 980 Hz.

//Menulis nilai analog (gelombang PWM) ke pin. Dapat digunakan untuk menyalakan LED di berbagai brightnesses atau mengendarai motor pada berbagai kecepatan. Setelah panggilan untuk analogWrite (), pin akan menghasilkan gelombang persegi stabil siklus tertentu sampai panggilan berikutnya ke analogWrite () (atau panggilan untuk digitalRead () atau digitalWrite () pada pin yang sama). Frekuensi sinyal PWM pada kebanyakan pin adalah sekitar 490 Hz. Di Uno dan mirip papan, pin 5 dan 6 memiliki frekuensi sekitar 980 Hz. Pin 3 dan 11 pada Leonardo juga berjalan pada 980 Hz.  
Fungsi analogWrite tidak ada hubungannya dengan pin analog atau fungsi analogRead.

**3.3 DUE AND ZERO ONLY**

1. analogReadResolution()

An extension of the Analog API for the Arduino Due and Zero.

Sets the size (in bits) of the value returned by analogRead(). It defaults to 10 bits (returns values between 0-1023) for backward compatibility with AVR based boards.

The **Due and the Zero have 12-bit ADC capabilities** that can be accessed by changing the resolution to 12. This will return values from analogRead() between 0 and 4095.

//Merupakan perluasan dari API Analog untuk Arduino Due and Zero.  
Menetapkan ukuran (dalam bit) dari nilai yang dikembalikan oleh analogRead (). Standarnya 10 bit (mengembalikan nilai antara 0-1023) untuk kompatibilitas dengan papan berbasis AVR.  
The due and the Zero memiliki kemampuan ADC 12-bit yang dapat diakses dengan mengubah resolusi ke 12. Ini akan mengembalikan nilai dari analogRead () antara 0 dan 4095.

1. analogWriteResolution()

An extension of the Analog API for the Arduino Due, Genuino and Arduino Zero and MKR1000.

analogWriteResolution() sets the resolution of the analogWrite() function. It defaults to 8 bits (values between 0-255) for backward compatibility with AVR based boards.

// Merupakan perluasan dari API Analog untuk Arduino Due, Genuino dan Arduino Zero dan MKR1000.  
AnalogWriteResolution () menetapkan resolusi fungsi analogWrite (). Standarnya 8 bit (nilai antara 0-255) untuk kompatibilitas dengan papan berbasis AVR.

**3.4 ADVANCED I/O**

1. tone()

Generates a square wave of the specified frequency (and 50% duty cycle) on a pin. A duration can be specified, otherwise the wave continues until a call to noTone(). The pin can be connected to a piezo buzzer or other speaker to play tones.

Only one tone can be generated at a time. If a tone is already playing on a different pin, the call to tone() will have no effect. If the tone is playing on the same pin, the call will set its frequency.

Use of the tone() function will interfere with PWM output on pins 3 and 11 (on boards other than the Mega).

// Menghasilkan gelombang persegi frekuensi yang ditentukan (dan 50% siklus) pada pin. Sebuah durasi dapat ditentukan, dinyatakan gelombang terus sampai panggilan untuk noTone (). pin dapat dihubungkan ke buzzer piezo atau pembicara lain untuk bermain nada.

Penggunaan fungsi tone () akan mengganggu output PWM pada pin 3 dan 11 (pada papan selain Mega).

1. noTone()

Stops the generation of a square wave triggered by [tone](https://www.arduino.cc/en/Reference/Tone)(). Has no effect if no tone is being generated.

// Menghentikan generasi gelombang persegi dipicu oleh tone(). Tidak berpengaruh jika ada nada yang dihasilkan.

1. shiftOut()

Shifts out a byte of data one bit at a time. Starts from either the most (i.e. the leftmost) or least (rightmost) significant bit. Each bit is written in turn to a data pin, after which a clock pin is pulsed (taken high, then low) to indicate that the bit is available.

// Bergeser keluar byte data satu bit pada satu waktu. Mulai dari salah sebagian (yaitu paling kiri) atau setidaknya (paling kanan) bit signifikan. Setiap bit ditulis pada gilirannya ke pin data, setelah mana pin jam berdenyut (diambil tinggi, maka rendah) untuk menunjukkan bahwa bit yang tersedia.

1. shiftIn()

Shifts in a byte of data one bit at a time. Starts from either the most (i.e. the leftmost) or least (rightmost) significant bit. For each bit, the clock pin is pulled high, the next bit is read from the data line, and then the clock pin is taken low. If you're interfacing with a device that's clocked by rising edges, you'll need to make sure that the clock pin is low before the first call to shiftIn(), e.g. with a call to digitalWrite(clockPin, LOW).

// Pergeseran dalam byte data satu bit pada satu waktu. Mulai dari salah sebagian (yaitu paling kiri) atau setidaknya (paling kanan) bit signifikan. Untuk setiap bit, jam pin ditarik tinggi, bit berikutnya dibaca dari baris data, dan kemudian jam pin diambil rendah.  
Jika Anda berinteraksi dengan perangkat yang clock oleh meningkatnya tepi, Anda harus memastikan bahwa jam pin rendah sebelum panggilan pertama yang shiftIn (), misalnya dengan panggilan untuk digitalWrite (clockPin, LOW).

1. pulseIn()

Reads a pulse (either HIGH or LOW) on a pin. For example, if **value** is **HIGH**, **pulseIn()** waits for the pin to go **HIGH**, starts timing, then waits for the pin to go **LOW** and stops timing. Returns the length of the pulse in microseconds or 0 if no complete pulse was received within the timeout. The timing of this function has been determined empirically and will probably show errors in shorter pulses. Works on pulses from 10 microseconds to 3 minutes in length. Please also note that if the pin is already high when the function is called, it will wait for the pin to go **LOW** and then **HIGH** before it starts counting. This routine can be used only if interrupts are activated. Furthermore the highest resolution is obtained with short intervals.

//Membaca pulsa (baik high atau low) pada pin. Misalnya, jika nilai adalah HIGH, pulseIn () menunggu untuk pin untuk pergi HIGH, dimulai waktu, kemudian menunggu untuk pin untuk pergi LOW dan berhenti waktu. Mengembalikan panjang dari pulsa di mikrodetik atau 0 jika tidak ada pulsa yang lengkap diterima dalam batas waktu.  
Waktu fungsi ini telah ditentukan secara empiris dan mungkin akan menunjukkan kesalahan dalam pulsa pendek. Bekerja pada pulsa dari 10 mikrodetik sampai 3 menit. Perlu diketahui juga bahwa jika pin sudah tinggi saat fungsi dipanggil, ia akan menunggu pin untuk pergi LOW dan kemudian HIGH sebelum mulai menghitung. rutin ini dapat digunakan hanya jika interupsi diaktifkan. Selain resolusi tertinggi diperoleh dengan interval pendek

**3.5 TIME**

1. milis()

Returns the number of milliseconds since the Arduino board began running the current program. This number will overflow (go back to zero), after approximately 50 days.

// Mengembalikan jumlah milidetik sejak papan Arduino mulai menjalankan program saat ini. Jumlah ini akan naik (kembali ke nol), setelah sekitar 50 hari.

1. Mikro()

Returns the number of microseconds since the Arduino board began running the current program. This number will overflow (go back to zero), after approximately 70 minutes. On 16 MHz Arduino boards (e.g. Duemilanove and Nano), this function has a resolution of four microseconds (i.e. the value returned is always a multiple of four). On 8 MHz Arduino boards (e.g. the LilyPad), this function has a resolution of eight microseconds.

// Mengembalikan jumlah mikrodetik sejak papan Arduino mulai menjalankan program saat ini. Jumlah ini akan naik (kembali ke nol), setelah sekitar 70 menit. Pada 16 MHz papan Arduino (seperti Duemilanove dan Nano), fungsi ini memiliki resolusi empat mikrodetik (yaitu nilai yang dikembalikan selalu kelipatan empat). Pada 8 MHz papan Arduino (seperti LilyPad), fungsi ini memiliki resolusi delapan mikrodetik.

Catatan: ada 1.000 mikrodetik dalam 1 milidetik dan 1.000.000 mikrodetik dalam 1 detik.

1. Delay()

Pauses the program for the amount of time (in miliseconds) specified as parameter. (There are 1000 milliseconds in a second.)

// Jeda program untuk jumlah waktu (dalam milidetik) ditetapkan sebagai parameter. (Ada 1000 milidetik dalam 1 detik.)

1. delayMicroseconds()

Pauses the program for the amount of time (in microseconds) specified as parameter. There are a thousand microseconds in a millisecond, and a million microseconds in a second. Currently, the largest value that will produce an accurate delay is 16383. This could change in future Arduino releases. For delays longer than a few thousand microseconds, you should use delay() instead.

// Jeda program untuk jumlah waktu (dalam mikrodetik) ditetapkan sebagai parameter. Ada 1000 mikrodetik dalam 1 milidetik, dan 1 juta mikrodetik dalam 1 detik. Saat ini, nilai terbesar yang akan menghasilkan delay yang akurat adalah 16383. Ini bisa berubah di rilis masa Arduino. Untuk keterlambatan lebih dari beberapa ribu mikrodetik, anda seharusnya menggunakan delay () sebagai gantinya.

**3.6 MATH**

1. min (x,y)

Calculates the minimum of two numbers.

// Menghitung minimal dari 2 angka.

1. Max(x,y)

Calculates the maximum of two numbers.

//Menghitung maksimal dari 2 angka.

1. Abs(x,y)

Computes the absolute value of a number.

// Menghitung nilai pasti dari sebuah angka.

1. Constrain(x,a,b)

Constrains a number to be within a range.

// Membatasi sejumlah angka yang berada dalam jangkauan.

1. Map(value, fromLow, fromHigh, toLow, toHigh)

Re-maps a number from one range to another. That is, a **value** of **fromLow** would get mapped to **toLow**, a value of **fromHigh** to **toHigh**, values in-between to values in-between, etc.

Does not constrain values to within the range, because out-of-range values are sometimes intended and useful. The constrain() function may be used either before or after this function, if limits to the ranges are desired.

//Memetakan kembali nomor dari satu rentang ke yang lain. Artinya, **nilai** **fromLow** akan mendapatkan dipetakan ke **toLow**, nilai dari **fromHigh** untuk **toHigh**, nilai di antara ke nilai diantara, dll

Tidak membatasi nilai-nilai ke dalam jangkauan, karena nilai out-of-range kadang-kadang yang dimaksudkan dan berguna. Fungsi constrain() dapat digunakan baik sebelum atau setelah fungsi ini, jika batas-batas rentang yang diinginkan.

1. Pow(base, exponent)

Calculates the value of a number raised to a power. Pow() can be used to raise a number to a fractional power. This is useful for generating exponential mapping of values or curves.

// Menghitung nilai dari sejumlah dinaikkan ke power. Pow() dapat digunakan untuk meningkatkan angka ke power pecahan. Hal ini berguna untuk menghasilkan pemetaan eksponensial dari nilai-nilai atau kurva.

1. Sqrt(x)

Calculates the square root of a number.

// Menghitung akar kuadrat dari angka.

**3.7 TRIGONOMETRY**

1. Sin(rad)

Calculates the sine of an angle (in radians). The result will be between -1 and 1.

// Menghitung sinus dari sudut (dalam radian). Hasilnya akan berada di antara -1 dan 1.

1. Cos(rad)

Calculates the cos of an angle (in radians). The result will be between -1 and 1.

// Menghitung cos dari sudut (dalam radian). Hasilnya akan berada di antara -1 dan 1.

1. Tan(rad)

Calculates the tangent of an angle (in radians). The result will be between negative infinity and infinity.

// Menghitung tangen dari sudut (dalam radian). Hasilnya akan antara infinity negatif dan tak terhingga.

**3.8 CHARACTER**

1. IsAlphaNumeric(thisChar)

Analyse if a char is alphanumeric.

// menganalisa jika character berupa alphanumeric parameter thisChar: berupa character yang akan dianalisa hasilnya berupa true atau false

1. isAlphaNumeric(thisChar)

Analyse if a char is alphanumeric.

// menganalisa apakah char berupa Alpha

1. isAscii(thisChar)

Analyse if a char is ASCII.

// menganalisa apakah char berupa ASCII

1. isWhitespace(thisChar)

Analyse if a char is a white space.

//menganalisa apakah char berupa White space

1. isControl(thisChar)

Analyse if a char is a control character.

//menganalisa apakah char dapat mengendalikan char

1. isDigit(thisChar)

Analyse if a char is a digit.

//menganalisa apakah char berupa digit.

1. isGraph(thisChar)

Analyse if a char is a printable character.

// menganalisa apakah char dapat diprint.

1. isLowerCase(thischar)

Analyse if a char is a lower case character.

// menganalisa apakah char berupa lower case.

1. isPrintable(thisChar)

Analyse if a char is a printable character.

// menganalisa apakah char dapat diprint.

1. isPunct(thisChar)

Analyse if a char is punctuation character.

// menganalisa apakah char berupa tanda baca.

1. isSpace(thisChar)

Analyse if a char is a space character.

// menganalisa apakah char berupa spasi.

1. isUpperCase(thisChar)

Analyse if a char is an upper case character.

// menganalisa apakah char berupa uppercase.

1. isHexadecimal(thisCase)

Analyse if a char is a valid hexadecimal digit.

// menganalisa apakah char berupa hexadecimal.

**3.9 RANDOM NUMBER**

1. randomSeed()

Initializes the pseudo-random number generator, causing it to start at an arbitrary point in its random sequence. This sequence, while very long, and random, is always the same.

If it is important for a sequence of values generated by random() to differ, on subsequent executions of a sketch, use randomSeed() to initialize the random number generator with a fairly random input, such as analogRead() on an unconnected pin.

// menginisialisasi generator pseudo-random nomor, yang menyebabkan memulai dari titik sembarang dalam urutan random. Urutan ini sangat panjang, dan random, tetapi selalu sama.

Ketika itu penting untuk urutan nilai yang dihasilkan oleh random() ke berbeda, pada eksekusi berikutnya dari sketsa, menggunakan randomSeed() untuk menginisialisasi nomor generator acak dengan inputan yang cukup random, seperti analogRead() pada pin yang tidak terhubung.

Sebaliknya, kadang-kadang dapat berguna untuk menggunakan urutan pseudo-random yang berulang-ulang. Hal ini dapat dilakukan dengan memanggil randomSeed() dengan nomor tetap, sebelum memulai urutan acak.

1. Random()

The random function generates pseudo-random numbers.

// fungsi random yang menghasilkan pseudo-random number

**3.10 BITS AND BYTES**

1. LowByte()

Extracts the low-order (rightmost) byte of a variable (e.g. a word).

// Mengekstrak low-order (paling kanan) byte dari dari variable (seperti kata).

1. highByte()

Extracts the high-order (leftmost) byte of a word (or the second lowest byte of a larger data type).

// Mengeksrak high-order (paling kiri) dari variable (atau byte terendah kedua dari tipe data yang lebih besar).

1. bitRead()

Reads a bit of a number.

// Membaca bit dari sebuah nomor.

1. bitWrite()

Writes a bit of a numeric variable.

// Menulis bit dari sebuah variable angka.

1. bitSet()

Sets (writes a 1 to) a bit of a numeric variable.

// Menetapkan (menulis 1 untuk) satu bit dari variable angka.

1. bitClear()

Clears (writes a 0 to) a bit of a numeric variable.

// Menghapus (menulis 0 untuk) satu bit dari variable angka.

1. Bit()

Computes the value of the specified bit (bit 0 is 1, bit 1 is 2, bit 2 is 4, etc.).

// Menghitung nilai bit tertentu (bit 0 adalah 1, bit 1 adalah 2, bit 2 adalah 4, dll).

**3.11 EXTERNAL INTERRUPTS**

1. AttachInterrupt()

Adds interrupts into variables

//menambahkan interrupt pada variable

1. detachInterrupt()

terminates interrupts in a variable

//mematikan atau menonatifkan interrupt di dalam variable

**3.12 INTERRUPTS**

1. interrupt()

Memungkinkan kembali interupsi (setelah mereka telah dinonaktifkan oleh noInterrupts ()). Interupsi memungkinkan tugas-tugas penting tertentu terjadi di latar belakang dan diaktifkan secara default. Beberapa fungsi tidak akan bekerja saat interupsi dinonaktifkan, dan komunikasi yang masuk dapat diabaikan. Interupsi dapat sedikit mengganggu waktu kode, bagaimanapun, dan mungkin dinonaktifkan karena terutama bagian penting dari kode.

1. noInterrupts()

Menonaktifkan interupsi (dapat diaktifkan kembali dengan interupsi ()). Interupsi memungkinkan tugas-tugas penting tertentu terjadi di latar belakang dan diaktifkan secara default. Beberapa fungsi tidak akan bekerja saat interupsi dinonaktifkan, dan komunikasi yang masuk dapat diabaikan. Interupsi dapat sedikit mengganggu waktu kode, bagaimanapun, dan mungkin dinonaktifkan karena terutama bagian penting dari kode

**3.13 COMMUNICATION**

1. serial

komunikasi serial pada pin TX / RX menggunakan tingkat logika TTL (5V atau 3.3V tergantung yang tertera pada papan). Jangan menghubungkan pin ini secara langsung ke port serial RS232; mereka akan beroperasi +/- 12V dan dapat merusak papan Arduino Anda.

Serial digunakan untuk berkomunikasi antara papan Arduino dan sebuah komputer atau perangkat lain. Semua papan Arduino memiliki setidaknya satu serial port (dikenal sebagai UART atau USART): Serial. Berkomunikasi pada digital pin 0 (RX) dan 1 (TX) serta dengan komputer melalui USB. Jadi, jika Anda menggunakan fungsi-fungsi ini, Anda tidak dapat menggunakan pin 0 dan 1 untuk input atau output digital.

Anda dapat menggunakan built-in monitor seri lingkungan Arduino untuk berkomunikasi dengan papan Arduino. Klik tombol Monitor seri pada toolbar dan pilih baud rate yang sama digunakan dalam panggilan untuk begin().

Arduino mega memiliki tiga port serial tambahan: serial1 pada pin 19 (RX) dan 18 (TX), Serial2 pada pin 17 (RX) dan 16 (TX), Serial3 pada pin 15 (RX) dan 14 (TX). Untuk menggunakan pin ini untuk berkomunikasi dengan komputer pribadi Anda, Anda akan memerlukan sebuah adaptor USB-to-serial tambahan, karena mereka tidak terhubung ke adaptor Mega USB-to-serial. Untuk menggunakannya untuk berkomunikasi dengan TTL perangkat serial eksternal, menghubungkan pin TX untuk perangkat RX pin, RX untuk perangkat TX pin, dan ground Mega ke ground perangkat Anda.

Arduino memiliki tiga tambahan 3.3V TTL serial port: serial1 pada pin 19 (RX) dan 18 (TX); Serial2 pada pin 17 (RX) dan 16 (TX), Serial3 pada pin 15 (RX) dan 14 (TX). Pin 0 dan 1 juga terhubung ke yang sesuai pin dari ATmega16U2 USB-to-TTL Serial chip, yang terhubung ke port USB debug. Selain itu, ada port USB-serial asli pada chip SAM3X, SerialUSB '.

Papan Arduino Leonardo menggunakan serial1 untuk berkomunikasi melalui TTL (5V) serial pada pin 0 (RX) dan 1 (TX). Serial dicadangkan untuk USB CDC communication.

1. Stream

kelas dasar untuk karakter dan dasar biner stream. Hal ini tidak dipanggil secara langsung, tapi dipanggil setiap kali Anda menggunakan fungsi yang bergantung pada itu.

Stream mendefinisikan fungsi membaca di Arduino. Ketika menggunakan fungsionalitas inti yang menggunakan read() atau metode yang sama, Anda dapat dengan aman berasumsi itu dipanggil pada kelas Stream. Untuk fungsi seperti print(), Turunan stream dari Print class.

**3.14 FUNCTIONS**

1. available()

Mendapat jumlah byte yang tersedia pada stream. Ini hanya untuk byte yang telah tiba.

1. read()

Membaca karakter dari stream yang masuk ke buffer

1. flush()

Membersihkan buffer ketika semua karakter telah dikirim

1. find()

Membaca data dari stream sampai target string ditemukan dengan waktu yang ditentukan Fungsi returns true jika target string ditemukan, false jika timed out

1. findUntil()

Membaca data dari stream sampai target string ditemukan dengan waktu yang ditentukan terminator string ditemukan

1. peek()

Membaca byte dari file tanpa melanjutkan ke tahap berikutnya. Artinya, panggilan berturut-turut untuk peek() akan mengembalikan nilai yang sama, seperti yang akan panggilan berikutnya untuk read().

1. readBytes()

Membaca karakter dari stream ke dalam buffer. Fungsi berakhir jika panjang bytes telah terbaca, atau time out (lihat setTimeout()). mengembalikan jumlah byte ditempatkan dalam buffer. angka 0 berarti tidak ada data valid yang ditemukan

1. readBytesUntil()

Membaca karakter dari stream ke dalam buffer. Fungsi berakhir jika karakter terminator terdeteksi, panjang bytes telah terbaca, atau time out (lihat setTimeout()). readBytesUntil() mengembalikan jumlah byte ditempatkan dalam buffer. Angka 0 berarti tidak ada data valid yang ditemukan.

1. readString()

Membaca karakter dari stream ke dalam string. fungsi berakhir jika time out (lihat setTimeout()).

1. readStringUntil()

Membaca karakter dari stream ke dalam string. Fungsi berakhir jika karakter terminator terdeteksi atau time out (lihat setTimeout()).

1. parseInt()

Mengembalikan valid pertama (panjang) nomor integer dari serial buffer. Karakter yang tidak integer (atau tanda minus) yang dilewati.

1. parseFloat()

Mengembalikan valid pertama nomor floating point dari posisi saat ini. karakter awal yang tidak integer (atau tanda minus) dilewati. parseFloat() diakhiri oleh karakter pertama yang bukan angka floating point.

1. setTimeout()

Menetapkan milidetik maksimum untuk menunggu stream data, standarnya ke 1000 milidetik. Fungsi ini merupakan bagian dari Stream class, dan dipanggil oleh kelas yang mewarisi dari itu (Wire, Serial, dll). Lihat halaman utama kelas Streaming untuk informasi lebih lanjut.