



多言語 プログラミング

～プログラミングにおける骨格～



配列（Cの例）

～データをまとめる～

構造化プログラミングでの配列

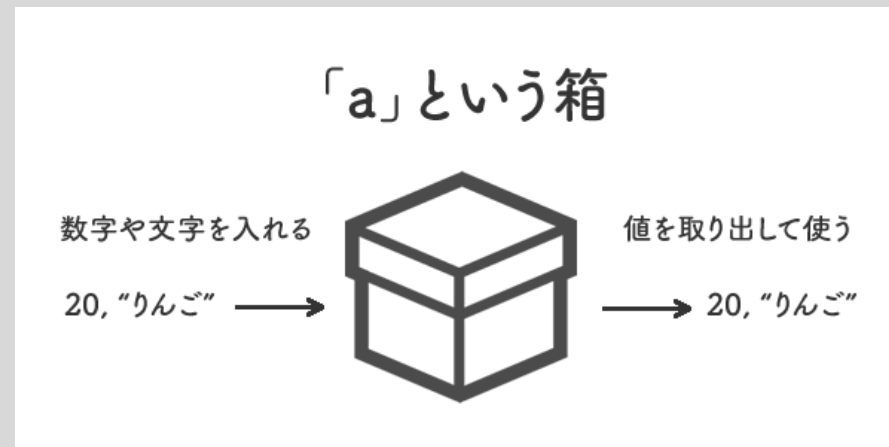
- 同じデータ型の変数をひとつに**まとめたもの**

- オブジェクト指向言語では配列は**オブジェクト**として扱う

➡ 後述

- この章では代表的な**C言語**での配列を紹介

- 文字列はchar型の配列として扱う



基本要素

- 宣言

配列を**作成**すること

- 初期化

配列の作成とともに**数値を割り当てる**こと

- 代入

配列の値を**上書き**すること

宣言

```
int x[5];
```

- 配列の宣言は言語によって様々

➡ C言語の方法を紹介

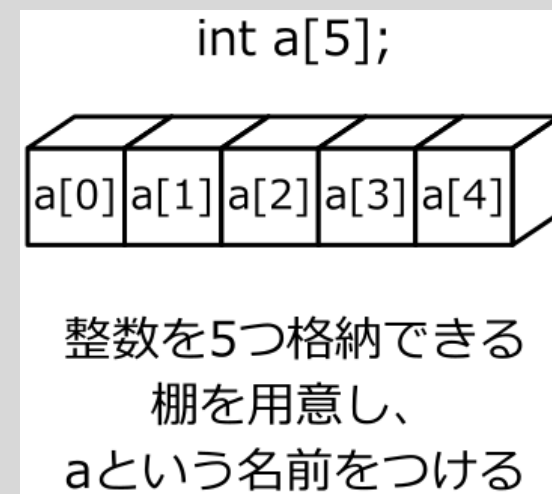
- 要素型 配列名[要素数] で宣言

要素数は定数、要素型は要素に入れるデータ型

- 添字演算子

配列名 + [先頭から何個後ろか] として配列にアクセス

[]を添字演算子と呼ぶ



初期化

```
int x[5] = {0, 1, 2, 3, 4};  
//先頭から順に0, 1, 2, 3, 4
```

```
int y[5] = {1, 2};  
//先頭から順に1, 2, 0, 0, 0
```

- 宣言と共に値を割り当てる
- 要素型 配列名[要素数] = {初期値}
で初期値を先頭から順番に割り当てることができる
- 要素数を超えての初期化はエラー
- 要素数より少ない数を割り当てると、それ以外は0になる

代入

```
x[0] = 3;  
//先頭から0番目(先頭含め1つ目)に3を代入  
x[4] = 2;  
//先頭から4番目(先頭含め5つ目)に2を代入
```

- 配列に配列を代入することは不可
- 配列名[先頭からのカウント] = 値
として添字演算子で配列の要素にアクセスし、代入する
- 配列を丸ごとコピーするには、制御構文を用いた工夫が必要

配列と制御構文

- 配列は制御構文を用いて操作をする
 ➡ 構造化プログラミングの手法
- 代表的なコピーと反転について紹介
- 大抵はfor文の繰り返し処理と組み合わせることが重要

配列のコピー

- for文で要素数分インクリメントして繰り返す
- コピーする側とされる側両方に同じ要素を指定して代入
- 0からインクリメントすると先頭から順番にコピーされていく

```
int x[5] = {1,2,3,4,5};  
int y[5] = {0}  
  
for(int i = 0, i < 5; i++){  
    y[i] = x[i];  
}
```

配列の反転

- 要素数の半分だけ繰り返す
- 入れ替えるときに塗り替えられる値はあらかじめ適当な変数に格納して保存しておく

```
int x[5] = {1,2,3,4,5};
```

```
for(int i = 0, i < 2; i++){  
    int temp = x[i];  
    x[i] = x[5-i];  
    x[5-i] = temp;  
}
```

x[0]をtempに保存 ➡ x[0]にx[5]代入 ➡ x[5]にtemp代入

x[1]をtempに保存 ➡ x[1]にx[4]代入 ➡ x[4]にtemp代入

多次元配列

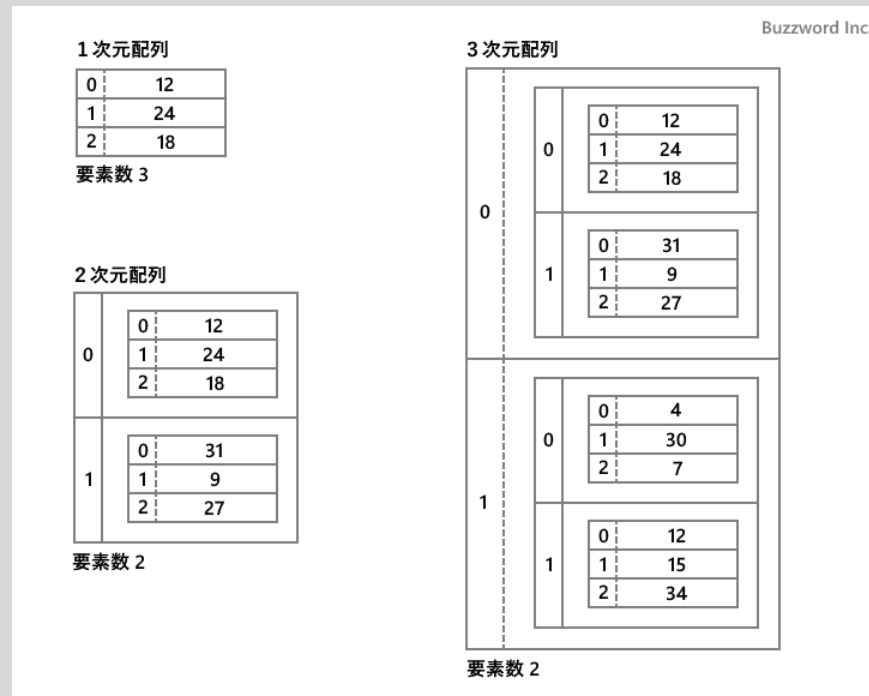
- 配列を配列として縦に並べると...

次元の多い配列が出来る



- 配列を要素とした配列と捉える ← 多次元配列

- 添字演算子が1つ増えるだけの違い




多次元配列基本要素

```
int x[2][3]
    = {{12,24,18},{31,9,27}};
x[1][2] = 5;
```

- 要素型 配列名[行要素][列要素] = {{初期値},{初期値}...}
とすることで宣言 + 初期化が可能
- 配列名[行][列]
で配列の要素にアクセスでき、代入が可能
- 多次元配列は行列的なので2重ループを使って処理をすることが殆ど

配列のまとめ

- 配列にも変数と同じように宣言、初期化、代入の操作が存在
- 配列丸ごと操作することは出来ず、制御構文を用いて操作する
- 配列を要素とすることで多次元配列を生成可能に



関数（Cの例）

～繰り返し処理をひとつに～

関数

処理をひとつにまとめて名前を付けたもの

- 関数はプログラム中のどこでも呼び出すことが可能
- メインループもサブルーチンも関数であり、
プログラムは全て関数でできている

基本要素

- 定義

関数の宣言を行う。関数の処理などを決定する。

- 呼び出し

関数の処理を実際に行う。

- 実引数

呼び出し元が関数に与える値。関数側では**仮引数**という。

- 戻り値

関数の呼び出し後、関数から戻ってくる値。

関数の定義

```
int sum(int a, int b){  
    int result = a +  
    b;  
    return result;  
}
```

- C言語は以下のように宣言する

戻り値の型 関数名(仮引数の宣言){処理}

- **return**命令で呼び出し元に返す戻り値を指定する
- 処理中は**return**で**強制終了**できる
- 戻り値や引数を指定しない場合、**void**型を使用する

オブジェクト指向言語における関数

- オブジェクト指向で関数をメソッドと呼ぶ
- メソッドの定義方法は様々あり、オブジェクト指向構文の章で後述
- JavaScriptでの基本的な宣言はfunction命令を使う

関数の呼び出し

```
int val = sum(2, 5);  
printf("%d", val);
```

関数名(実引数1, 実引数2…)

として関数の処理を実行し値を取りだすことが可能

- 。実引数を与えて処理を実行し
return文で指定した値が返却される

関数のオーバーロード

- C++などでは同じ名前で処理の違う関数を複数個宣言できる
- 引数の個数やデータ型によって振り分けが行われる
- 戻り値のみ異なる関数はエラーになる

```
int val1 = sum(2, 5);  
int val2 = sum(3, 4, 5);
```

```
int sum(int a, int b){  
    int result = a + b;  
    return result;  
}  
int sum(int a, int b, int c){  
    int result = a + b + c;  
    return result;  
}
```

変数の有効範囲

- **ブロック有効範囲**

関数の**ブロック**({から}まで)での有効範囲

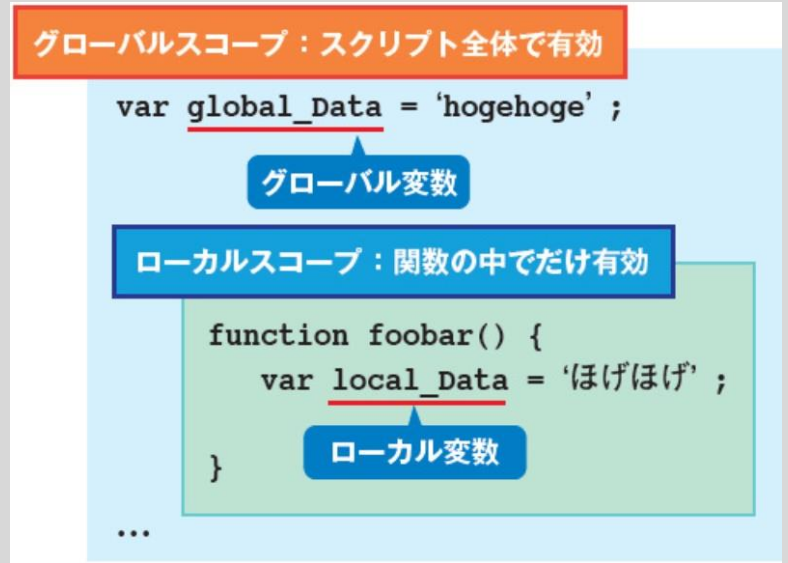
宣言した変数(**ローカル変数**)は外から扱えない

- **ファイル有効範囲**

どのブロックにも属さない一番外側での有効範囲

宣言した変数(**グローバル変数**)はファイル内のどこからでも扱える

➡ 変更されるタイミングが不明なので**可読性**は×




ヘッダファイル

- ~.hというファイルには、ライブラリにて使用される関数の宣言をしている
- #include命令を使用することでライブラリ関数が使用可能に
- Arduinoではセンサーなどを扱うときに使用されることがしばしば

関数のまとめ

- 関数の操作には定義と呼び出しがある
- 関数内では仮引数として受け取り戻り値をreturnする
- 呼び出し元では実引数を与えて戻り値を受け取る
- 変数の有効範囲は関数のブロックにより決定する

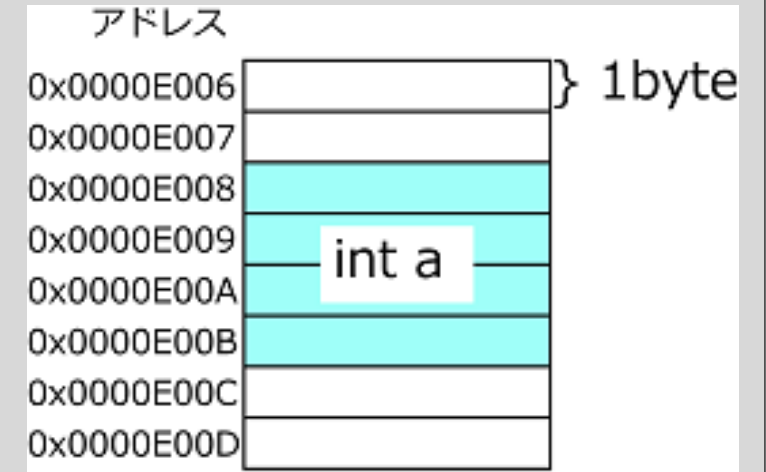


ポインタ（Cの例）

～アドレスからプログラムを操作しよう～

変数の住所

- 。コンピュータにて変数はメモリ上に確保される領域
- 。メモリ上のどこに変数があるかを指すもの ➡ アドレス
- 。アドレスを見ると時々の変数の値が分かる



ポインタ

決まったデータ型のアドレスを指す特殊なデータ型

- 変数と同じように扱う
- そのまま代入をするとアドレス自体が変更される

ポインタ基本要素

- 宣言

ポインタ型の変数を宣言

- 初期化

ポインタ型の変数に初期値のアドレスを与える

- 代入

何もつけないとアドレス自体を上書きする

間接演算子を用いることで変数本体を上書きできる

宣言と初期化

- データ型 *ポインタ名 = &変数名
としてポインタ変数を宣言する
- アドレス演算子(単項&演算子)
指定された変数名のアドレスを返す
- ポインタとはアドレスを指す変数のこと！

```
int x = 2;  
int y = 3;  
int *p = &x;
```

2種類の代入

```
p = &y;  
*p = 5;  
//yが5に変更される
```

ポインタ名 = &変数名

とすることでポインタ自体の値（アドレス）を変更できる

*ポインタ名 = 値

とすると、ポインタの指すアドレスの変数が上書きされる

➡ 参照による代入という

この*を間接演算子という

関数への参照渡し

- 。仮引数にポインタを宣言すると...

戻り値無しで変数そのものの値を変更



有効範囲外からでも操作が可能に

➡ 変数そのものを渡す、参照渡し

```
void changeval(int *p){  
    *p *= 5;  
}  
int main(void){  
    int x = 2;  
    changeval(&x);  
    //xの値が5倍に  
    return 0;  
}
```

配列の受け渡し

- 参照渡しを使えば**配列**の受け渡しも可能に
- 配列のポインタを**加算**すると
加算した分後ろの要素を指す
- 繰り返し処理でも扱いやすく

```
void changeval(int *a, int n){  
    for(int i = 0; i < n; i++){  
        *(a + i) *= 5;  
    }  
}  
  
int main(void){  
    int x[3] = {1,2,3};  
    changeval(x,3);  
    //単独の配列名は  
    //配列の先頭の変数のアドレス  
    return 0;  
}
```

ポインタのまとめ

- ポインタは変数のアドレスを指す変数
- 間接演算子を用いて参照による代入が可能
- 関数に参照渡しを行うことで配列の操作も可能に



コンピュータの基礎

～マイコンを使ってみよう～

ノイマンコンピュータの大原則～ 5 大装置～

- CPU

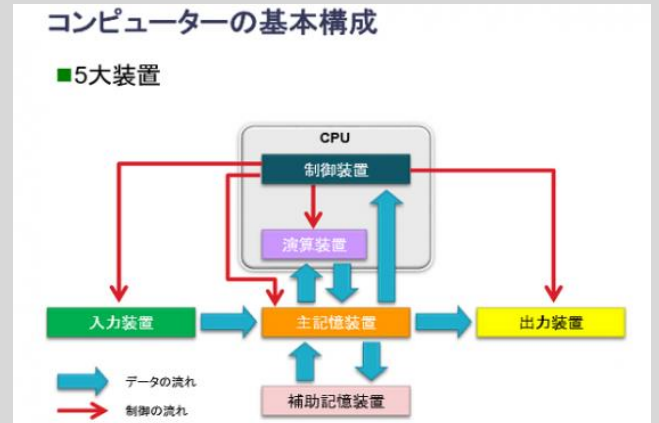
プログラムを実行する**制御**装置、算術演算を行う**演算**装置

- インターフェース

人とコンピュータを繋ぐ装置。人からの**入力**、人への**出力**を行う。

- メモリ

プログラムやデータを**記憶**する装置。



CPU

保存されたプログラムを実行し、コンピュータ全体を制御する**制御**装置



算術演算や論理演算を実行し、データ処理を行う**演算**装置



CPU : Central Processing Unit
中央処理装置



メモリ

- 主記憶装置

一時的にデータを保存しておく装置。**RAM**と呼ばれる
SDRやDDR、レジスタやキャッシュ（CPU内部メモリ）など

- 補助記憶装置

永続的にデータを保存しておく装置。**ROM**と呼ばれる。
HDDやSSDなど、様々な物理的機構を用いて保管



インターフェース

- 入力装置

人からコンピュータへデータを送る装置。

マウスやキーボード、マイクやカメラに加えてペンタブなども

- 出力装置

コンピュータから人へデータを送る装置。

ディスプレイやスピーカー、プリンタなど

コンピュータとプログラム

フラッシュメモリにプログラムを**保存**



電源がつくとコンピュータがプログラムを**実行**



電源がついている限りプログラムの処理を**ループ**し続ける⇐**無限ループ**

スリープ状態はボタンの**入力**を**待機**するプログラムが…

プログラミング言語とアセンブリ

- 高水準言語

C言語～の言語。人が理解しやすいプログラム

- 低水準言語

～アセンブリの言語。機械が理解しやすいプログラム

コンパイルとアセンブル

- コンパイル

高水準言語から低水準言語に翻訳すること


- アセンブル

アセンブリ言語を機械語（バイナリ）に翻訳すること

コンパイルとアセンブルはほとんどの場合セット

コンピュータの基礎のまとめ

- コンピュータにはCPU、メモリ、インターフェースの3つに
重要な5つの機能が含まれている
- プログラムはコンピュータの電源を付けたときから無限ループで開始
- プログラミング言語をコンパイルして機械語がコンピュータに指令を



マイコンプログラミング

～マイコンを使ってみよう～

マイクロコンピュータ

- 。マイクロコンピュータ（コントローラ）、マイコン
フラッシュメモリにプログラムした通りに動く電子部品

中に保存されたプログラム（ソフトウェア）が
接続された回路（ハードウェア）の脳となり動かす

マイコンの種類

- **AVR**マイコン

アトメル社のマイコン、PICマイコンのライバル

- **PIC**マイコン

マイクロチップ社のマイコン、AVRマイコンのライバル

- **ARM**マイコン

arm社のアーキテクチャを搭載したマイコン、非常に高性能



マイコンプログラミングの基礎

- **グローバル変数**はなるべく控える

フラッシュメモリの容量を圧迫してしまうため

- **メインループ**が電源を切断するまで回り続ける

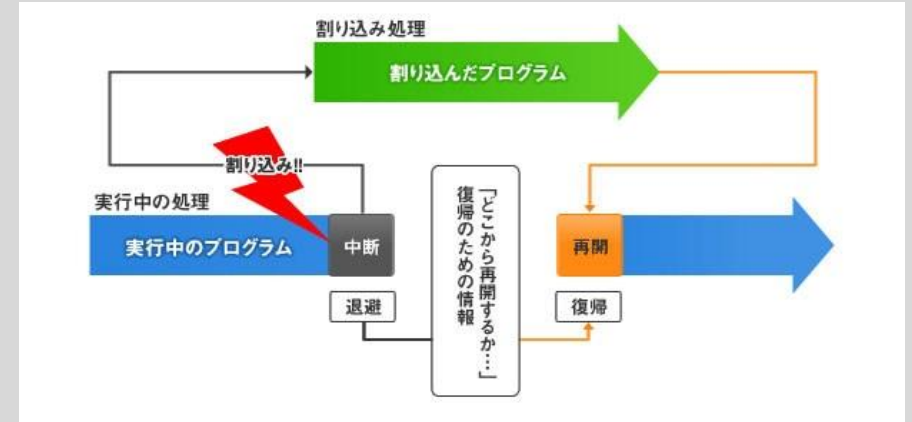
前章参照、変数設定などの前処理はメインループ外で

- **デバッグ**はシリアル通信で動的解析

動的解析は後述、**Serial.println**を使用して値を表示する

割り込み

- 外部の**入力**や予め決めた**時間**に発生する
- メインループの処理が**ストップ**し定義しておいた処理が発生
- 割り込み処理が**終了**すると、元の位置からメイン処理が**再開**



ファームウェアとソフトウェア

- ファームウェア

ハードウェアを**直接的**に制御するプログラムをファームウェアという

- ソフトウェア

ハードウェアをファームウェアを介して**間接的に**制御するプログラム

ハードウェアやファームウェアは回路やチップで決定する = **固い**

ソフトウェアはプログラムで柔軟に変化する = **柔らかい**

ハードウェア

- **ハードウェア**

マイコンを含める**回路全体**、駆動部までを含めることもある

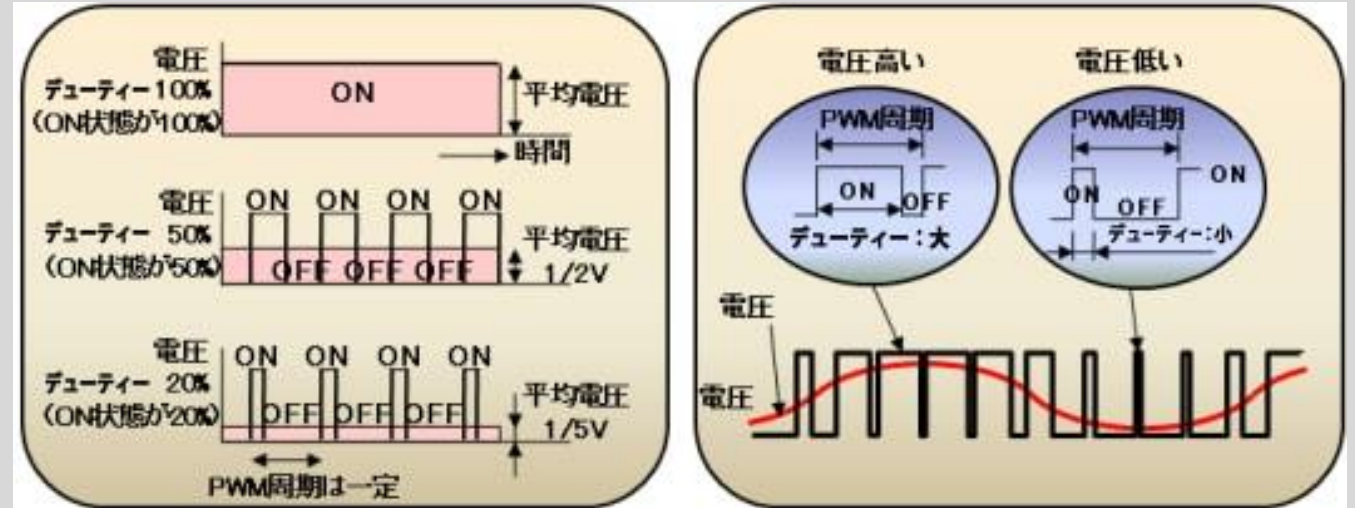
- ハードウェアをマイコンで制御する様々な制御方法を紹介

GPIO

- 汎用IO（デジタル入出力ピン）のこと
- 0又は1の2種類の電圧をLow又はHighで表現して入出力を行う
- 一般的なマイコンの入出力ポート

PWM

- パルス幅変調のこと



- デジタルで表現される矩形波のOn/Off時間を制御することで
疑似的なアナログ電圧を作ることが可能に
- AC/DCコンバータやモータの制御など様々なものに活用されている

DACとADC

- **DAC** (デジタルアナログコンバータ)
PWMによる制御で**デジタルな信号から**アナログ電圧を生成する
 - **ADC** (アナログデジタルコンバータ)
PWMによって**アナログの入力値**をデジタル信号に変換する
- マイコンにもよるがGPIOのうち使用できるピンが限られている

UART

- パラレルをシリアルに変換して行う高効率の通信のこと
- マイコンはUARTでPCとシリアル通信を行っている
- ArduinoではSerial.printlnという関数を使用して出力できる

SPIとI2C

- SPI通信

同期式のシリアル通信で主にセンサ類と行う通信方式
通信するセンサの数の配線が必要

- I2C通信

SPIと同じく同期式のシリアル通信でセンサ類と行う通信
配線が共通で、複数個のセンサをまとめて扱えるという利点がある

マイコンプログラミングのまとめ

- マイコンはプログラム通りに動く**電子部品**
- 様々な種類があり、プログラミングするときには**注意**が必要
- マイコンは**ハードウェア**を様々な技術を使って制御している

参考文献

- <https://eng-entrance.com/linux-shellscript-variable>
- <https://xtech.nikkei.com/it/atcl/column/14/091700069/091700002/>
- <https://wa3.i-3-i.info/diff446programming.html>
- <http://www.b.s.osakafu-u.ac.jp/~hezoe/pro/chapter5.html>
- https://kanda-it-school-kensyu.com/php-super-intro-contents/psi_ch09/psi_0905/
- https://kanda-it-school-kensyu.com/java-basic-intro-contents/jbi_ch06/jbi_0606/
- <https://itmanabi.com/structured-objectoriented-prog/>
- https://kanda-it-school-kensyu.com/java-basic-intro-contents/jbi_ch07/jbi_0704/
- https://kanda-it-school-kensyu.com/java-basic-intro-contents/jbi_ch07/jbi_0703/
- https://kanda-it-school-kensyu.com/java-basic-intro-contents/jbi_ch07/jbi_0702/
- <https://www.crucial.jp/memory/ddr4/ct2k16g4dfd824a>
- <https://brain.cc.kogakuin.ac.jp/~kanamaru/lecture/prog1/06-01.html>
- <https://www.javadrive.jp/start/array/index7.html>
- <https://www.renesas.com/jp/ja/support/engineer-school/mcu-programming-peripherals-04-interrupts>
- <https://toshiba.semicon-storage.com/jp/semiconductor/knowledge/e-learning/brushless-motor/chapter3/what-pwm.html>