Arduino をはじめよう@KYOTO-U

2016/03/26

京都大学 学術情報メディアセンター

喜多　一

目次

1. この資料で学ぶこと

この資料では Arduino を使った電子工作について,

- 電子部品の使い方などのハードウェアの取扱い
- Arduino のプログラミング

について学びます。

2 章以下は次のような構成になっています。

表 1

章	ハードウェア	プログラミング
2 章	● Arduino の PC への接続とセットアップ	● Arduino のプログラムの基本構成 ● コンパイルと実行 ● エラーとその対処 ● 逐次実行
3 章	● ブレッドボードの使い方 ● LED, ● 抵抗器, ● コンタクトスイッチ	● デジタル入出力 ● 条件分岐 ● 変数の利用
4 章	● 圧電スピーカ	● 配列 ● 関数 ● ２つのプログラムを組み合わせること
5 章	ここまでの力試し	
6 章	● 光センサ(CdS) ● 可変抵抗器 ● PWM	● アナログ入出力 ● PC との通信 ● 繰り返し処理
7～10 章	付録	

2. Arduino の使い方

2.1 今回の学習目標

- ArduinoUno を PC に接続する方法を覚える.
- Arduino の統合開発環境(IDE) を起動する方法を覚える.
- Arduino IDE で簡単な例を用いて，ベリファイ，アップロード，実行する方法を覚える.
- 簡単なエラーを経験する.
- プログラムでの命令の逐次実行を知る.

2.2 作業の心得

この授業ではプログラムを書いたり，電子回路を作成したりします．人が行う作業には必ず誤りが伴います．これを極力，無くすことで安全で効率的な作業を実現したいと思います.

そこで，この授業では

- 原則として**２人１組で作業**をしてもらいます.
- どちらかの任せるのではなく，**片方が作業すればその内容を他方が確認する**ようにします.
- どの作業でも，**どちらの人でも実行できるように交代して作業**してください.

2.3 準備

まず、以下の手順で Arduino を使えるようにします。

a) 授業用 PC を起動し、Student でログインします。

b) Arduino Uno を USB ケーブルで PC のキーボード右側の USB ポートに接続します。図 1 参照

c) Arduino/Suwano の COM ポートの確認，図 2 参照

- スタートボタンを押し，メニューの「コンピュータ」を選びます.
- 「コンピュータ」のウィンドウから「システムのプロパティ」を選びます.

図 1　Arduino Uno の接続

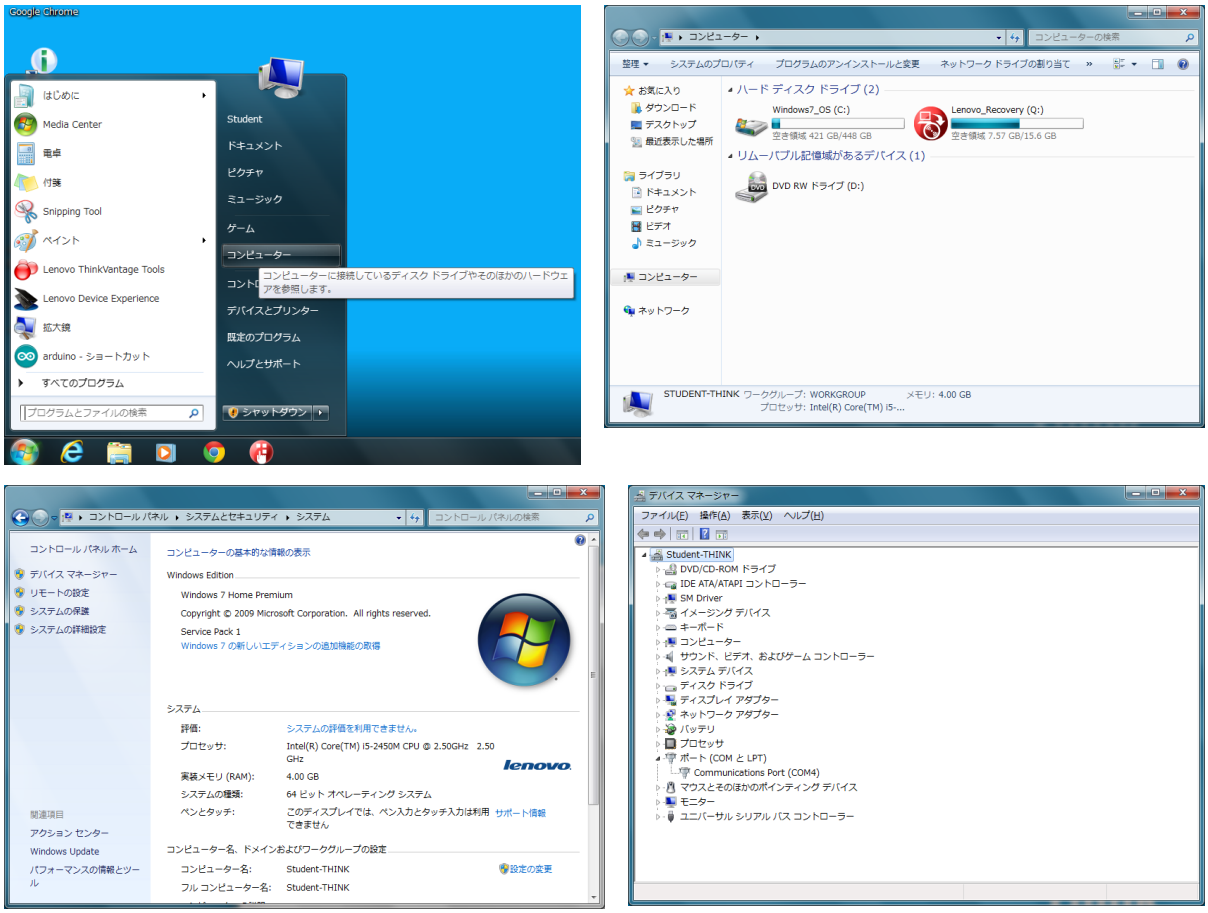


図 2

図 3

- 「システム」ウィンドウから「デバイスマネージャ」メニューを選びます．管理者権限を要求されますが閲覧するだけなので必要ありません．
- ポートを選択し Communications Port (COM4) という表示を確認します。COM の後の数字は異なっているかもしれませんが、数字を控えておきます。

d) Arduino IDE (統合開発環境)の起動

デスクトップの adruino　ショートカットをダブルクリックします．図 3 参照

e) Arduino IDE の初期設定

- メニューバーから「Tools」→「Board」→「ArduinoUno」あるいは「Suwano」を選びます．
- また「Tools」→「Serial Port」→「COM4」を選びます．COM のあとの数字は 4 とは限りません．c) (エ)で確認したものであればかまいません．

2.4　例題の実行

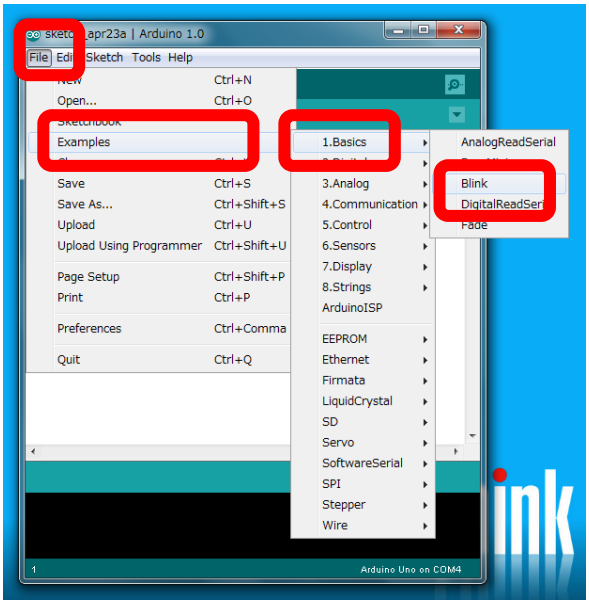


図 4

a) File メニューから「Examples」→「1.Basics」→ 「Blink」を選びます．図 4 参照

b) Blink というプログラムのウィンドウが現れます．

c) ベリファイ（コンパイルとも言います，一番左のボタン）を行います．終了すればウィンドウの下側に黒いパネルにメッセージが表れます． 図 5 参照

d) ベリファイ（コンパイル）が終了したら隣の矢印ボタンで実行可能プログラムを Arduino に送ります（アップロード）．図 6 参照

e) 転送が済めば Arduino でこのプログラムが自動的に動き始めます．このプログラムでは 1 秒おきにボード上の LED が点灯と消灯を繰り返します．

図 5

図 6

2.5 例題の変更と再実行

例題の下から 4 行目の delay(1000); と下から 2 行目の delay(1000); をそれぞれ delay(1500); と delay(500); に書き換えて，ベリファイとアップロードを行ってみてください．点滅する時間が変化すれば成功です．

注意：使う文字はすべて半角文字です．行末のセミコロン（;）を忘れないでください．

2.6 エラーを経験する

プログラムを作成しているときにキー入力を誤ったりするエラーにしばしば出会います．ここでは意図的にエラーを体験してみましょう．Blink の例題について以下のことを試みて，ベリファイのときにどのようなメッセージが出るのか確認してみてください．

- 行末のセミコロンをなくす．
- 開いた中括弧を閉じ忘れる．
- 大文字と小文字を間違う．

2.7 例題を最初から書いてみる

File メニューで新しいプログラムの作成を行います．内容は Blink の例題と同じものにしますが，コピー&ペースはせずにすべて自分で入力してみてください．

注意：以下の注意を守ってください．

- すべて半角文字で入力します．
- 行末のセミコロン（;）を忘れないでください．
- 中括弧({,})などの記号をまちがわないでください．
- setup, pinMode, OUTPUT, loop, digitalWrite, HIGH, LOW, delay などは**大文字，小文字をこの通り**入力してください．また綴り誤りがないか確認してください．
- setup(), loop() などの後の() は必要です．
- 文字の色は自動的にシステムがつけます．
- /* で始まり */ で終わる箇所，// から行末までは「注釈（コメント）」と言ってプログラムとしては意味をもちません．人がプログラムを理解するためにつけるものですので，省略することも可能です．ただし，プログラムに適切な注釈を入れることはプログラムの開発において重要なこととされています．

入力を終えたら必ず誤記がないか確認します．確認を終えたらベリファイ，アップロードを実行して動作するかどうか試してください．

ベリファイでエラーメッセージが出たらどこか誤っているはずです．エラーメッセージを参考に内容を確認してください．

2.8　プログラムの内容

行	ソースコード	解説
1	// Example 01：Blinking LED	// で始まる行は注釈です．プログラムとしては機能しません．
2		
3	void setup()	setup() という関数（命令群）の定義です．
4	{	{ から } までが関数の中で実行する内容です．
5	pinMode(13, OUTPUT);	13 番目のピンを出力用に使う設定にします．
6	}	
7		
8	void loop()	loop() という関数の定義です．
9	{	{ から} までが関数の中で実行する内容です．
10	digitalWrite(13, HIGH);	LED 番目のピンに高い電圧を出力します．
11	delay(1000);	1000 ミリ秒待ちます．
12	digitalWrite(13, LOW);	LED 番目のピンに低い電圧を出力します．
13	delay(1000);	1000 ミリ秒待ちます．
14	}	

ソースコード 1

Arduino のプログラムは一連の命令群を並べたもの「関数」を作成することで行います．関数は次のような記述形式をとります．

```
返り値の型 関数名(引数の記述)
{
  実行する内容
}
```

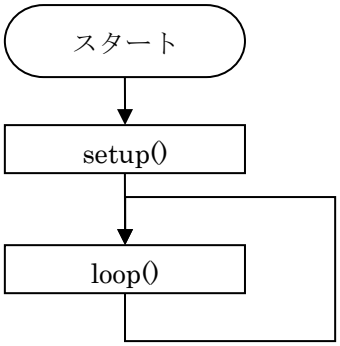


図 7　フローチャート

上の例の関数 setup は

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

返り値の型は void（なにも返さないことを意味します），関数名は setup，引数はなし，実行する内容は pinMode(13, OUTPUT); です．

Arduino では関数 setup() と loop() は特別な役割を持ちます．Arduino が起動すると，準備のためまず一回だけ setup() と呼ばれる関数が呼び出されます．次に loop() という関数が繰り

返し呼び出されます．フローチャートで表すと図 7 のようになります．

上のプログラムではまず関数 setup() の中で関数呼び出し pinMode(13, OUTPUT) により，13 番目のピンを出力用(OUTPUT)に設定します．LED と OUTPUT は pinMode() という関数を呼び出す際に与える引数です．Arduino では 13 番目のピンには予めボード上の LED が接続されています．

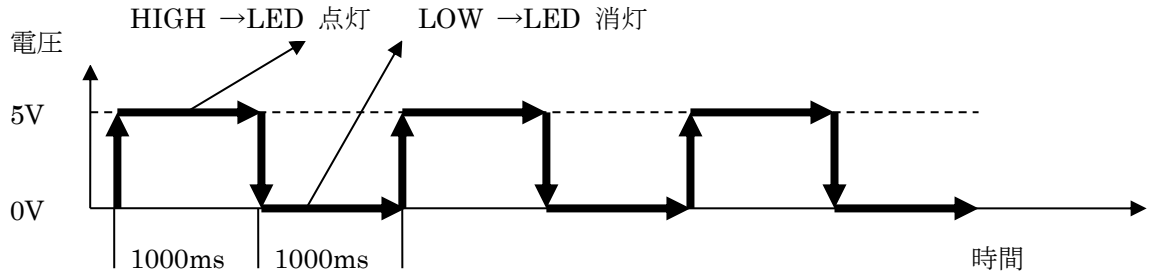


図 8 LED 点灯のタイムチャート

つぎに関数 loop() が繰り返し呼び出されるのですが，loop() の中では，まず digitalWrite(13,HIGH) の呼び出しで 13 番目のピンの電圧が高く（5V）設定されます．つぎに delay(1000) の呼び出しで 1 秒（1000 ミリ秒）待ち, digitalWrite(13, LOW) の呼び出しで 13 番目のピンの電圧が低く（0V）設定されます．そして delay(1000) の呼び出しで再び 1 秒（1000 ミリ秒）待ちます．これが繰り返されることで LED が 1 秒おきに点滅を繰り返します．

2.9 動作を組み合わせる

プログラムの基本は「上から下へ命令を実行してゆく」**逐次実行**です．次の図に示すようなタイムチャートで LED が点灯するスケッチ（プログラム）を作成してみてください．

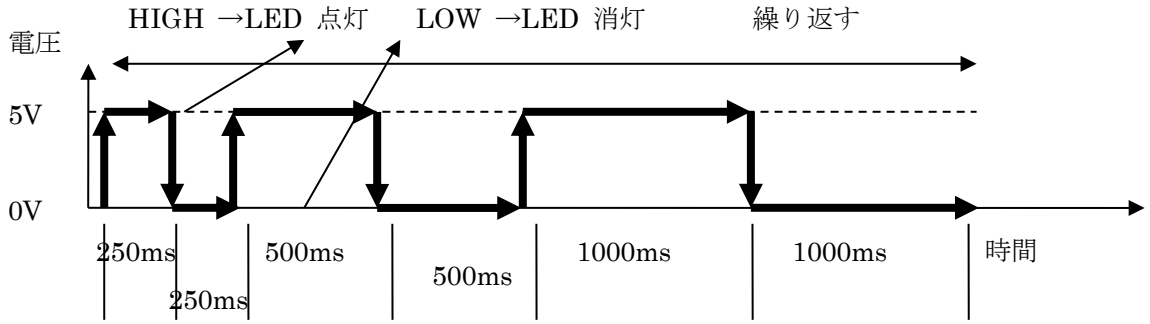


図 9

行	ソースコード	解説
1	// Example 01：Blinking LED	// で始まる行は注釈です. プログラムとしては機能しません.
2		setup() という関数（命令群）の定義です.
3	void setup()	{ から } までが関数の中で実行する内容です.
4	{	13 番目のピンを出力用に使う設定にします.
5	pinMode(13, OUTPUT);	
6	}	
7		
8	void loop()	loop() という関数の定義です.
9	{	{ から} までが関数の中で実行する内容です.
10	digitalWrite(13, HIGH);	LED 番目のピンに高い電圧を出力します.
11	delay(250);	250 ミリ秒待ちます.
12	digitalWrite(13, LOW);	LED 番目のピンに低い電圧を出力します.
13	delay(250);	250 ミリ秒待ちます.
14	digitalWrite(13, HIGH);	次の４行が間隔 500 ミリ秒での点滅です.
15	delay(500);	
16	digitalWrite(13, LOW);	
17	delay(500);	
18	digitalWrite(13, HIGH);	次の４行が間隔 1000 ミリ秒での点滅です.
19	delay(1000);	
20	digitalWrite(13, LOW);	
21	delay(1000);	
22	}	

ソースコード 2

3. ブレッドボードの使いかた

3.1 今回の学習目標

- ブレッドボードの使い方を覚える.
- ブレッドボード上で LED とコンタクトスイッチを使った回路作成し Arduino で制御する.(デジタルの入力と出力)
- 変数と条件分岐のプログラミングを覚える.

3.2 ブレッドボードを使う

ブレッドボードは電子部品を差し込んで回路を試作するための配線用のボードです。図 11 のブレッドボードでは，中央の部分は A 方向に相互が導通してします.

周辺部の２列は B 方向に相互に導通しています．ここは**電源線（赤色）**や **０ Ｖ の線（GND，青色）**などに使います.

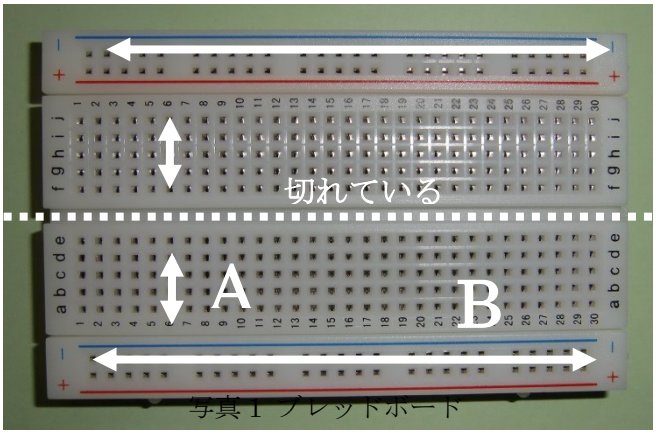


写真１ ブレッドボード

図 11 ブレッドボード

今回の実習ではコンタクトスイッチ（写真２）の押し下げによって LED の点灯，消灯を制御する回路を作ってみます．コンタクトスイッチには４本の足が出ていますが，B 方向の２本は内部で接続されています．スイッチを押し下げると A 方向の２本が導通します.

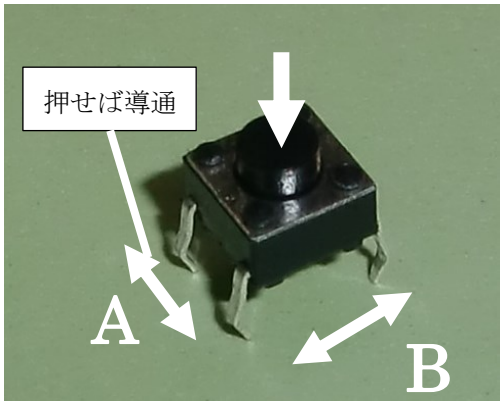


図 10

3.3 LED とコンタクトスイッチを使った回路の制御

3.3.1 ブレッドボード上での配線

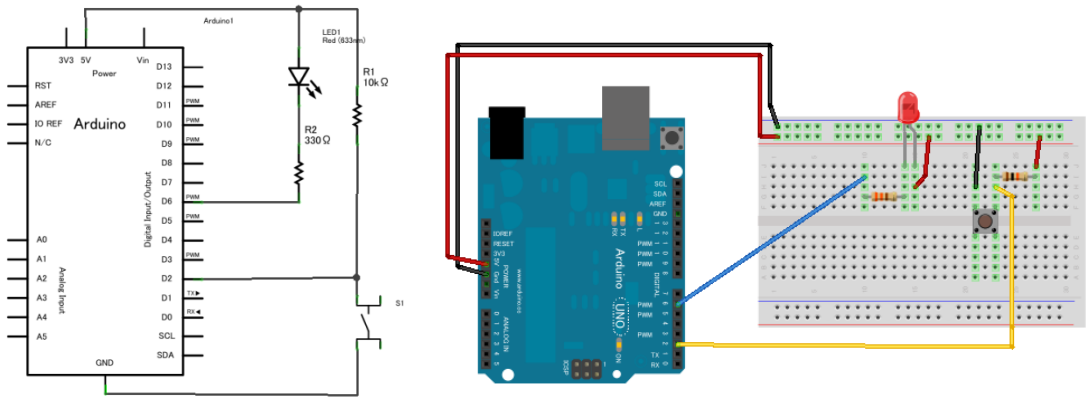


図 12　LED とコンタクトスイッチの配線

Arduino でコンタクトスイッチの押し下げ状態を検出するためにブレッドボードを使って図 12 のような回路を作成します．以下の手順に従ってください．

注意！　回路を構成している最中は Arduino を USB ケーブルから外しておいてください．これは電源などを短絡して Arduino や電子部品を壊さないようにするためです．

注意！　電子部品の端子は柔らかくて曲がったり折れたりしやすいものです．ブレッドボードに差し込む際には無理に押し込んだりせず，慎重に作業してください．

1. まず　Arduino の 5V と GND からの配線を行ないます．**5V には赤**の，**GND には黒**のジャンパ線を使います．
2. 次に LED と抵抗器からなる回路を作成してください．
   - 5V の電源から LED のアノード（足の長いほう）へ，LED のカソード（足の短いほう）から抵抗器（330Ω，橙，橙，茶，金のカラーコード）を介して Arduino の D6 に配線します．
3. 次にコンタクトスイッチの回路を作成します．
   - 5V の電源から抵抗器（10kΩ，茶，黒，橙，金のカラーコード）を介してコンタクトスイッ

チの片側に，コンタクトスイッチの反対側を GND に配線します．

- コンタクトスイッチと抵抗器が接続している部分から Arduino の D2 ピンに配線します．

この回路では D6 が 0V (LOW) のとき LED が点灯し，5V (HIGH) のとき LED は消灯します．またスイッチが押されているときは D2 には 0V (LOW) が，押されていないときには 5V (HIGH) の電圧がかかります．

回路図で表したものが図 12 の左の図です．**回路図では電源線(5V) を上側に，GND(0V) を下側に描く**ようにします．

注意！配線は電子回路における「プログラム」です．誤っていると動作しませんし，部品を壊す恐れもあります．動作させる前に必ず点検するようにしてください．

3.3.2 コンタクトスイッチからの入力の処理（その１）

スイッチを押している間, LED が点灯するするプログラムを作成します. 次のプログラム（スケッチ）を作成して実行してみてください.

行番号	ソースコード	説明
1	#define LED 6	LED と書いて 6 と,
2	#define BUTTON 2	BUTTON と書いて 2 を表す定義
3	int val = 0;	ボタンの状態を格納する整数型の
4		変数
5	void setup()	
6	{	
7	pinMode(LED, OUTPUT);	
8	pinMode(BUTTON, INPUT);	2 番ピンを入力用に使います
9	}	
10		
11	void loop() {	
12	val = digitalRead(BUTTON);	2 番ピンの状態を読み込みます
13	if (val == HIGH) {	状態が HIGH なら
14	digitalWrite(LED, HIGH);	6 番ピンを高い電圧にします
15	} else {	そうでなければ
16	digitalWrite(LED, LOW);	6 番ピンを低い電圧にします
17	}	
18	delay(10);	少し待ちます
19	}	

ソースコード 3

3.3.3 マクロ定義と変数の利用

マクロ定義: 1 行目の「#define LED 6」と 2 行目の「#define BUTTON 2」の #define（define は定義という意味）は次に現れる名前をその次に現れる定義（文字の列）の代わりに使えるようにする命令（マクロ定義）です.

```
#define 名前 定義
```

プログラムの中に直接 6 や 2 という数字を埋め込む代わりに LED とか BUTTON と書けば, ベリファイの際に 6 や 2 に置き換えてくれます. たとえば 7 行目の

```
pinMode(LED, OUTPUT)
```

は pinMode(6, OUTPUT) に置き換えられます. こうすることにより, 直接, 数字を書くより意味が分かりやすくなりプログラムが分かりやすくなります.

変数の宣言: 2行目の「int val = 0;」は変数の宣言です.

- 変数には値を入れたり（書き換えたり）, 入っている値を参照したりすることができます. プログラムの中で情報を保持するために基本的な仕組みです.
- この行では整数(int, integer の略)型の変数で名前が val というものを宣言し, その内容を 0 に初期化しています.
  - int 型の整数は Arduino では -2^{15} (-32768) から $2^{15}-1$ (32767) までの整数を扱うことができます. 桁数が限られていることに注意してください.
  - より桁数の多い数値の利用には long 型 (-2,147,483,648 から 2,147,483,647 まで扱えます) を,
  - また小数点以下の値を持つ場合には float 型 (floating point number は浮動小数点数という意味) を用います.
- 12 行目の

  ```
  val = digitalRead(BUTTON);
  ```

  の = は変数に値を代入（設定）する命令（演算子）で,
  - まず右辺の値（ここでは関数 digitalRead(BUTTON) で得られるスイッチの値）を計算し,
  - これを左辺の変数 val に設定します.
- 13 行目の

  ```
  if (val == HIGH) {
  ```

  の if 文（後述します）の中で val == HIGH とありますが,「==」（等号２つ, 等号１つは代入に使われるので注意が必要）は左辺と右辺の値が等しいかどうかを検査する命令（演算子）です. 左辺に val とありますが, これにより val という変数に設定している値を読み出し（評価し）, 右辺の定数 HIGH と等しいかどうか比較しています.
- setup() や loop() といった**関数の定義の外側で定義される変数**は**「外部変数」**と呼びます.
  - このプログラムでは loop() の中で val に値を代入したり, 値を評価したりしていますが,
  - 外部変数はどの関数の中でも代入や評価ができます.
  - また外部変数の値はプログラムが起動すれば, 常に値を保持しています.

3.3.4 コンタクトスイッチからの入力の処理（その２）

今度はスイッチを押す度に点灯と消灯を切り替えるプログラムを作ります．そのためには

- スイッチが押されていない状態から押された状態への変化を検出し，
- その度に消灯すべきか，点灯すべきかを切り替えなければなりません．

プログラムは少し複雑になります．

行番号	ソースコード	説明
1	#define LED 6	
2	#define BUTTON 2	BUTTON と書いて 2 を表す
3	int val = 1;	ボタンの状態を格納する変数
4	int oldVal = 1;	直前のボタンの状態用の変数
5	int lampOn = 0;	LED の状態用の変数，点灯時に 1,
6	void setup()	消灯時に 0 とします．
7	{	
8	pinMode(LED, OUTPUT);	
9	pinMode(BUTTON, INPUT);	2 番ピンを入力用に使います
10	digitalWrite(LED, HIGH);	最初，消灯しておきます．
11	}	
12	void loop() {	
13	val = digitalRead(BUTTON);	2 番ピンの状態を読み込みます
14	if ((val == LOW)&&(oldVal == HIGH)) {	今の状態が LOW で前の状態が
15	if (lampOn == 0) {	HIGH ならスイッチが押されたと
16	lampOn = 1;	きなので lampOn の値を反転しま
17	} else {	す．
18	lampOn = 0;	
19	}	
20	}	
21	if (lampOn == 1) {	lampOn の値に応じて LED を点
22	digitalWrite(LED, LOW);	灯，消灯します．
23	} else {	
24	digitalWrite(LED, HIGH);	
25	}	
26	delay(10);	少し待ちます
27	oldVal = val;	今の状態を前の状態として保存し
28	}	ます．

ソースコード 4

先のプログラムは 15-19,21-25 行目を変更して以下のように書いても同様に動作します．これは lampOn の値が 0 と 1 を取ること，HIGH と LOW の定義が 1 と 0 であることを使っています．プログラムはコンパクトになりますが，分かりにくくなります．

行番号	ソースコード	説明
1	#define LED 6	
2	#define BUTTON 2	BUTTON と書いて 2 を表す
3	int val = 1;	ボタンの状態を格納する変数
4	int oldVal = 1;	直前のボタンの状態用の変数
5	int lampOn = 0;	LED の状態用の変数
6	void setup()	
7	{	
8	pinMode(LED, OUTPUT);	
9	pinMode(BUTTON, INPUT);	2 番ピンを入力用に使います
10	digitalWrite(LED, HIGH);	
11	}	
12		
13	void loop() {	
14	val = digitalRead(BUTTON);	2 番ピンの状態を読み込みます
15	if ((val == LOW)&&(oldVal == HIGH)) {	今の状態が HIGH で前の状態が
16	lampOn = 1 - lampOn;	LOW なら lampOn の値を反転しま
17	}	す.
18	digitalWrite(LED, 1-lampOn);	lampOn の値で LED を点灯，消灯
19	delay(10);	し,少し待ちます
20	oldVal = val;	今の状態を前の状態として保存し
21	}	ます.

ソースコード 5

3.3.5 コンタクトスイッチからの入力の処理（その３）

今度はスイッチが押されたら暫くの間，LED を点灯するプログラムを作りましょう．

行番号	ソースコード	説明
1	#define LED 6	
2	#define BUTTON 2	BUTTON と書いて 2 を表す
3	#define DURATION 1000	継続時間
4	#define TICK 10	刻み幅
5	int val = 1;	ボタンの状態を格納する変数
6	int oldVal = 1;	直前のボタンの状態用の変数
7	int count = 0;	LED の点灯期間用の変数
8	void setup()	
9	{	
10	pinMode(LED, OUTPUT);	
11	pinMode(BUTTON, INPUT);	2 番ピンを入力用に使います
12	digitalWrite(LED,HIGH);	最初，消灯します．
13	}	
14		
15	void loop() {	
16	val = digitalRead(BUTTON);	2 番ピンの状態を読み込みます
17	if ((val == LOW)&&(oldVal == HIGH)) {	今の状態が HIGH で前の状態が
18	count = DURATION;	LOW なら count を 1000 にしま
19	}	す．
20	if (count > 0) {	count が正なら
21	digitalWrite(LED, LOW);	LED を点灯し
22	count = count - TICK;	count を TICK だけ減らします
23	} else {	
24	digitalWrite(LED, HIGH);	そうでなければ消灯
25	}	
26	delay(TICK);	TICK だけ少し待ちます
27	oldVal = val;	今の状態を前の状態として保存し
28	}	ます．

ソースコード 6

3.4　変数，代入文と条件分岐のプログラミング

3.3 節のプログラムで新しく出てきたプログラムの概念に変数のほかに条件分岐があります.

3.4.1 代入文での変数の評価と代入

変数は代入文の右辺に現れれば，その値を評価し，左辺に現れれば右辺の値を代入することになります．例えば 2.4 節のプログラムで 22 行目に

```
count = count - TICK;
```

とありますが，例えば count に値 1000 が，TICK の定義が 10 に設定さているとすれば，この行は以下のように解釈，実行されます.

1. まず右辺を評価（します）
   (ア) 右辺に現れた count では値を評価して 1000 を得ます.
   (イ) 右辺を計算すると 1000 - 10 を評価して 990 という値（評価値）を得ます.
2. これを左辺に現れた変数 count に代入します. count の値は右辺の評価値である 990 に更新されます.

すなわち，この行は変数 count に代入されている値を１０だけ減らすプログラムです.

代入文は　「変数 = 式；」の形式をとる.
等号の右辺を計算して結果を左辺の変数に設定する.
右辺に現れる変数は代入されている値が使われる.

3.4.2 条件分岐

変数の値などによってプログラムの動作を変えるには if 文を用います.

2.2 節のプログラムでは 13 ～ 17 行目のプログラムで

```
if (val == HIGH) {
  digitalWrite(LED, HIGH);
} else {
  digitalWrite(LED, LOW);
}
```

のように記述されています．これは変数 val の値が HIGH と等しいかどうか(val == HIGH) を検査し，それが成り立っていたら LED を消灯（digitalWrite(LED, HIGH)）し，そうでなければ(else) LED を点灯する（digitalWrite(LED, LOW)）というプログラムです.

注意！等しいかどうかの検査には演算子 ==(等号２つ）を用います．等号が１つ (=) だと代入文になってしまうので注意が必要です.

これをフローチャートに描いたものが図 13 です．条件判断には菱形の記号を使います．

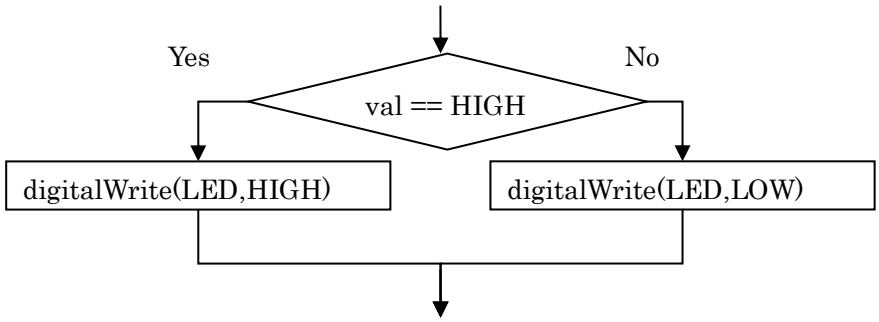


図 13　条件分岐のフローチャート

複合的な条件を設定することもできます．2.3 節のプログラムでは２つの等号がともに成立していることを判定するために

```
if ((val == LOW)&&(oldVal == HIGH)) {
…
}
```

という表記を用いています．ここで，「&&」は論理積(AND，両方が成立しているときに成立)を表します．論理和（OR，どちらかが成立しているときに成立）には「||」を用います．

if 文の形式は以下のようになります．

```
if（条件）{
   成立している時の動作
} else {
   成立していない時の動作
}
```

成立していない時に何もしないなら，else { … } を省略できます．

```
if（条件）{
   成立している時の動作
}
```

if 文を入れ子にして使うこともできます．先の複合的な条件設定は if 文の入れ子で次のようにプログラムすることもできます．

```
  if (val == LOW) {
    if (oldVal == HIGH) {
      …
    }
  }
```

3.5 コンタクトスイッチのプログラミング

コンタクトスイッチの状況は関数 digitalRead(BUTTON) を呼び出すことによって知ることができます.「スイッチが押されている」ことを検出するには，digitalRead(BUTTON) の値が HIGH かどうかを検査すればいいのですが,「スイッチが（新たに）押された」ことを検出するには

- 直前には押されていない
- 今の時点では押されている

という２つの条件がともに成り立っていることを確認する必要があります.

そこで

```
oldVal = digitalRead(BUTTON);
delay(10);
val = digitalRead(BUTTON);
if ((val == LOW)&&(oldVal == HIGH)) {
  成立したときの処理
}
```

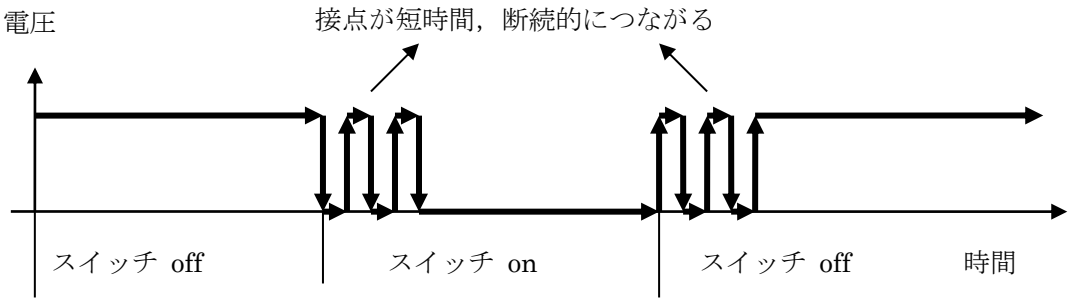


図 14 スイッチのバウンシング

などとして検査しなければなりません. ここで delay(10)を挿入しているのは機械的なスイッチは金属片を相互に接触させて回路を導通させるのですが，接触は単純に生じるのではなく，スイッチの投入時や切断時には短時間に何度が接触，非接触を繰り返すことがあります．これはバウンスと呼びます．このようなバウンスを無視するために 10 ミリ秒だけ（値は経験的に決める）待つようにプログラムするのです.

2.3 節や 2.4 節のプログラムでは loop() 関数が繰り返し呼ばれることを利用して以下の動

作をしています。

- oldVal は初期値として 1 (HIGH, 押されていない)としている。
- loop() 関数の最初に val = digitalRead(BUTTON); としてスイッチの値を読み取っている。
- if ((val == LOW)&&(oldVal == HIGH)) { … } によりスイッチが「(新たに) 押された」ことを検出して処理している。
- delay(10) などで適当な時間待っている。
  次回の loop() の呼び出しに備えて今回読み込んだスイッチの値 (val に代入されている)を oldVal に代入 (oldVal = val;) して保存する。

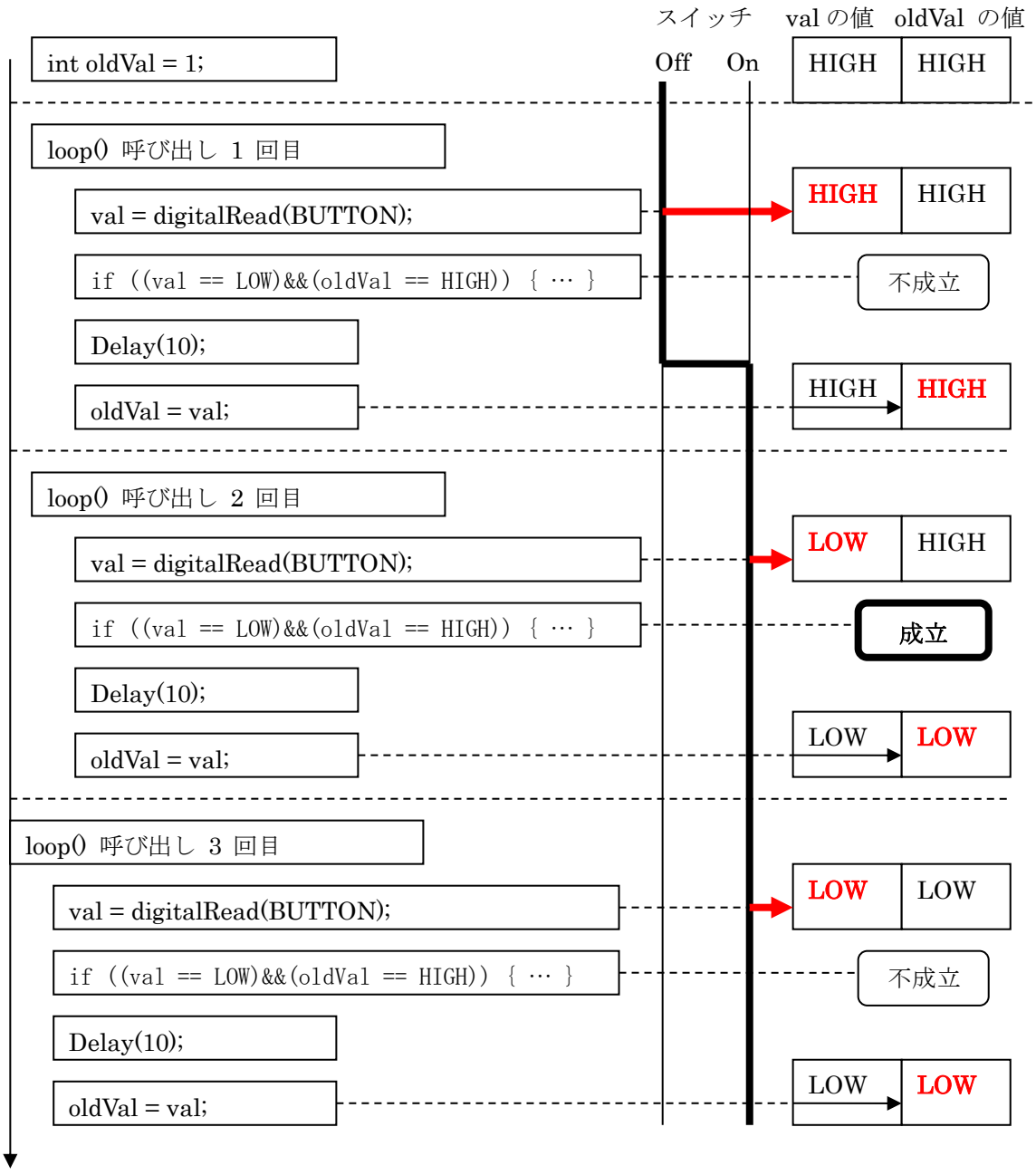


図 15

3.6　プログラムを少し変えてみる

先の回路に LED をもう一つ加え, 10 番ピンからの出力で制御するようにします. 片方は 3.3.2 節のプログラムと同じように, 他方は 3.3.4 節のプログラムと同じように動作するプログラムを作成してみなさい. 回路図を図 16 に示します.

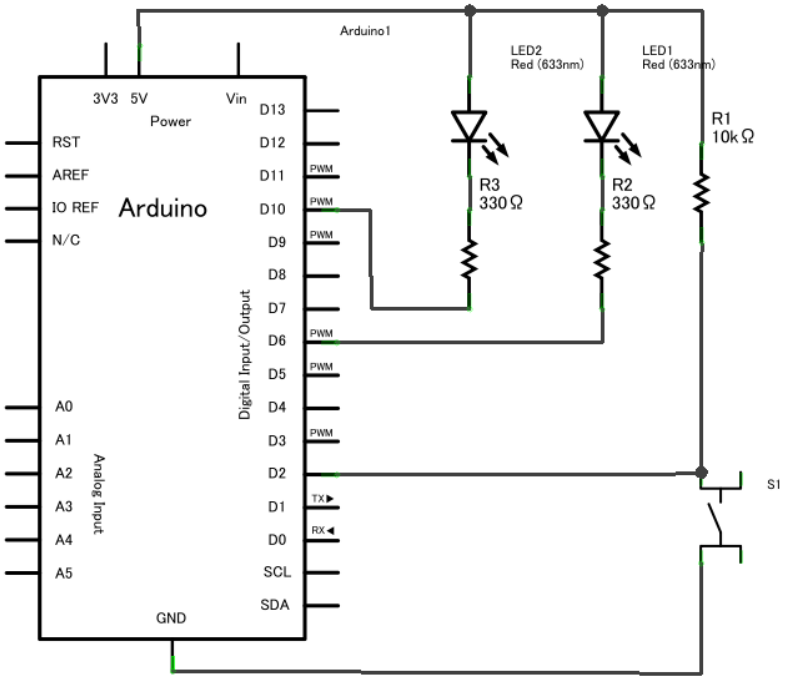


図 16 LED を 2 つ使う回路

4. Arduino で音出す

4.1 今回の学習目標

- 音の出し方を知る.
- 配列の使い方を知る.
- 関数を定義して使う.
- プログラムの改変に挑戦する.

4.2 回路の準備.

ブレッドボードを用いて図 17 のような回路を作成してください. 圧電ブザーの電極に極性はありません.

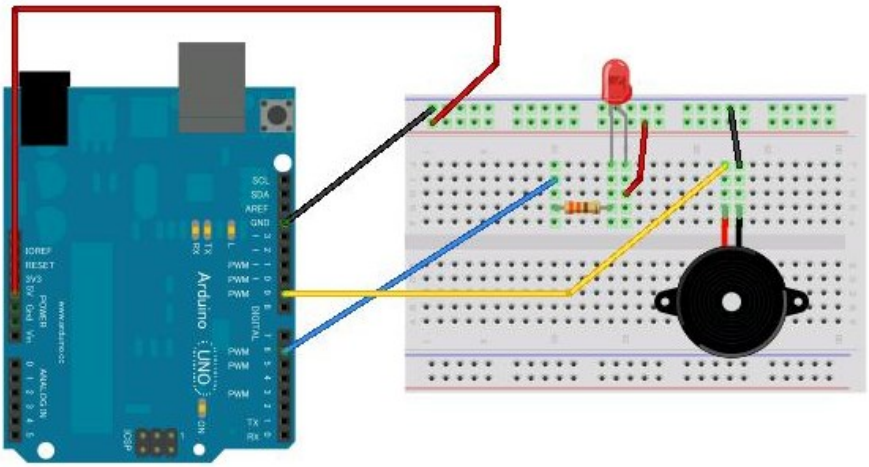


図 17 作成する回路（実体図）

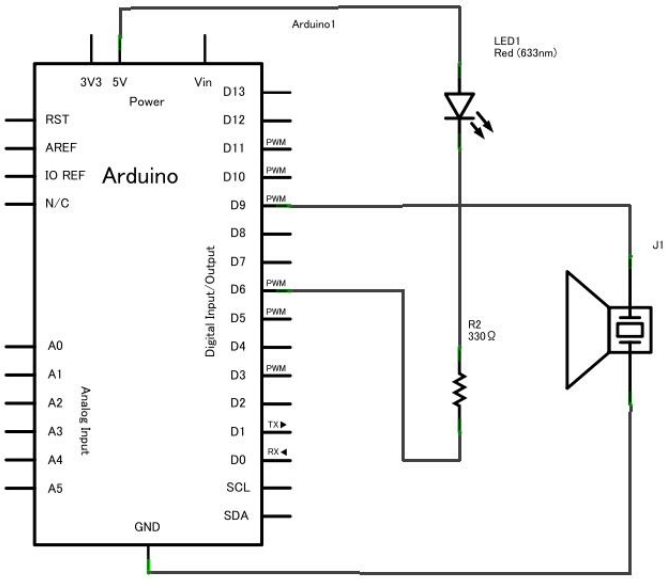


図 18　作成する回路（回路図）

4.3 音の出し方を知る

次のプログラムを作成し，実行してみてください．

行番号	ソースコード	説明
1	#define SPEAKER 9	9 番ピンを SPEAKER と，
2	#define LED 6	6 番ピンを LED とします.
3	int freq[] =	初期値を持った配列の宣言
4	{440,466,494,523,554,587,622,659,698,740,	音階に相当する周波数
5	784,831,880,932,988,1047,1109,1175,1245,	
6	1319,1397,1480,1568,1661,1760};	
7	int TONE_MIN = 0;	音階の添え字の最小値
8	int TONE_MAX = 24;	音階の添え字の最大値
9	int melody[] = {3,5,7,8,10,12,14,15,-1};	曲の音程.
10	int M_LENGTH = 9;	曲の長さ
11	int mIndex = 0;	曲の場所を指す変数
12	int DURATION = 750;	１音の長さ(750ms)
13	void rewind(void);	関数 rewind() と
14	void makeNextTone(void);	makeNextTone() のプロトタ
15		イプ宣言
16	void makeNextTone() {	次の音を出す関数
17	int fIndex = melody[mIndex];	音程を取り出す
18	if ((fIndex>=TONE_MIN)&&(fIndex<TONE_MAX)){	音階の範囲にあれば
19	tone(SPEAKER,freq[fIndex],DURATION);	tone 関数で音を出す
20	delay(DURATION);	音の長さだけ待つ
21	} else {	
22	digitalWrite(LED,LOW);	そうでなきえれば LED を点灯
23	delay(DURATION);	750msec 待って
24	digitalWrite(LED,HIGH);	LED を消灯
25	}	
26	mIndex = mIndex + 1;	曲の場所を進める
27	if (mIndex >= M_LENGTH) {	範囲に達したら
28	mIndex = 0;	最初にもどす
29	}	
30	}	
31		
32	void rewind() {	曲の先頭にもどる関数

```
33      mIndex = 0;
34    }
35
36    void setup() {
37      pinMode(SPEAKER, OUTPUT);                ピンの設定
38      pinMode(LED, OUTPUT);
39      digitalWrite(LED,HIGH);                  LED を消灯
40      rewind();                                曲の先頭に
41    }
42
43    void loop() {
44      makeNextTone();                          次の音を出す
45    }
```

ソースコード 7

音をスピーカー(圧電ブザーと呼びます. あまり電流を消費しないので Arduino のポートに直接つなぐことができます)から出すための関数が

tone(int ポート, int 周波数, int 長さ)

です. 周波数は Hz, 長さは ms (ミリ秒)です. この関数は音を鳴らしながら, 平行してプログラムの続きを実行するので同じ長さだけ, delay() 関数で待つようにしています.

4.4 配列と関数

4.4.1 配列

2 節のプログラムでは音階を表す周波数と曲の音程をそれぞれ

```
int freq[] =
  {440, 466, 494, 523, 554, 587, 622, 659, 698, 740,
   784, 831, 880, 932, 988, 1047, 1109, 1175, 1245,
   1319, 1397, 1480, 1568, 1661, 1760};

int melody[] = {3, 5, 7, 8, 10, 12, 14, 15, -1};
```

と表現しています. このように多くのデータを一括してプログラムで扱う手段として配列があります.
これらの例では int 型の配列 freq や melody を初期データとともに外部変数として宣言しています.
一般には

配列要素の型 配列名[] = {初期値データ, …, 初期値データ};

という形式になります. このほかに, 配列の大きさを指定して宣言する方法として

配列要素の型 配列名[要素数];

という方法もあります. 例えば

int melody[9];

とすると, int 型の要素を 9 つ持つ配列 melody を用意します.

配列の添え字は C 言語では 0 から始まります. 先の例の melody の配列要素と初期値は以下のようになります.

melody[0] → 3, melody[1] → 5, melody[2] → 7, melody[3] → 8, melody[4] → 10, melody[5] → 12, melody[6] → 14, melody[7] → 15, melody[8] → -1

配列の要素は[]内に要素の添え字を指定すれば後は普通の変数のように内容を得たり, 値を代入したりすることができます.
ここでは曲の先頭にもどる関数 rewind() と次の音を出す関数 makeNextTone() を定義して呼び出しています. いずれも引数も戻り値もありません. ただし, どちらの関数も外部変数 mIndex の値を変更します. 先のプログラムでは

```
int fIndex = melody[mIndex];
if ((fIndex>=TONE_MIN)&&(fIndex<TONE_MAX)){
  tone(SPEAKER,freq[fIndex],DURATION);
  delay(DURATION);
} else {
```

とありますが, fIndex = melody[mIndex]; で曲を表す配列 melody の添え字 mIndex が指す要素の内容 (音程) を読み出して音程の添え字 fIndex を得ています.
次に tone(SPEAKER,freq[fIndex],DURATION); の中で音程を表す周波数の配列 freq から添え字 fIndex の示す要素の内容 (周波数) を得て, 音を出す関数 tone() の引数として与えています.

変数は「値」を書いた札を入れる箱, 配列はこのような箱が連なって, 配列名と添え字で参照できるものに例えることができます. 変数 mIndex, fIndex を使って配列 melody[], freq[] の値を参照している様子を図解したものが下の図です.

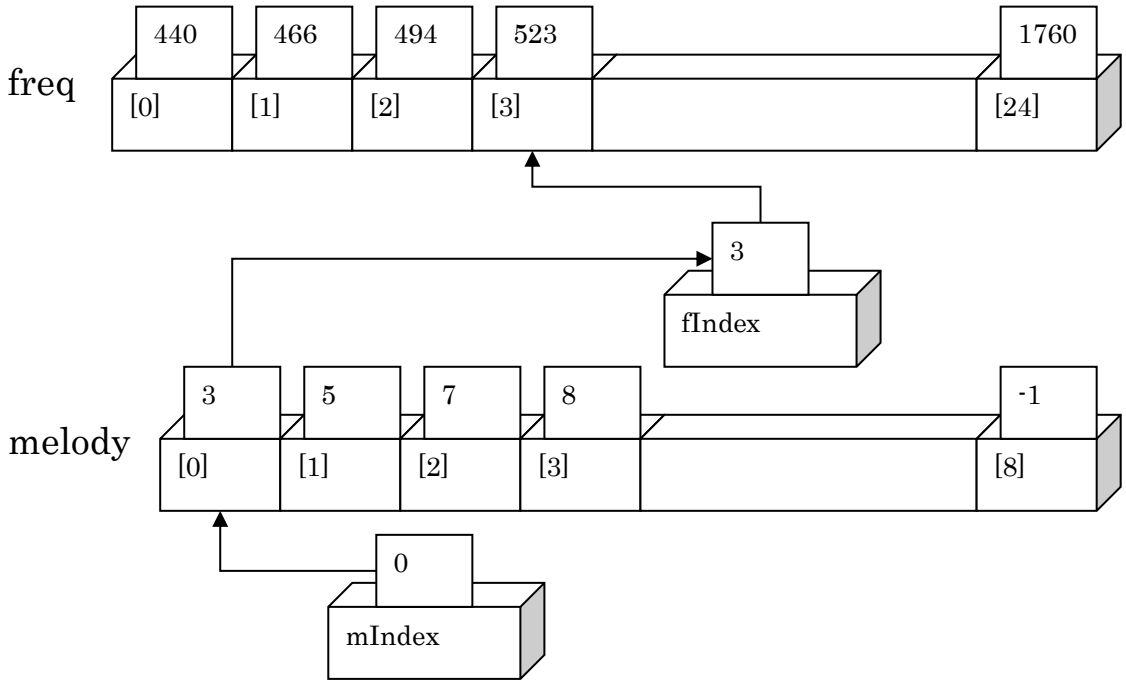


図 19

4.4.2 関数

3.2 節のプログラムでは rewind() と makeNextTone() という２つの関数を定義し、呼び出しています。

これまで関数として定義したものは

```
void setup()
void loop()
```

でした。これらはどちらも Arduino でのプログラムに必須のものとして想定されているものを利用者が記述したものです。

一方、利用者が呼び出したライブラリ関数としては

```
void pinMode(pin, mode)
void digitalWrite(pin, value)
void delay(ms)
int digitalRead(pin)
```

void tone(pin, frequency, duration)

があります.

3.2 節のプログラムではプログラム内に新たに関数 makeNextTone() と rewind() を定義し, 関数 setup() や loop() の中で呼び出しています.

- **関数プロトタイプ宣言**:13, 14 行目は「関数プロトタイプの宣言」と言い, 定義する関数について予め「返り値の型」,「関数名」,「引数の型」を実際の関数の定義やこの関数を呼び出す関数の定義の前に書きます. どちらの関数も引数も返り値もないのでともに void と書かれています. 最後にセミコロン(;) が付くことに注意してください.
- **関数の定義**:16 ～ 30 行目と 32～34 行目がそれぞれ関数 makeNextTone() と rewind() の定義です. 関数の返り値の型, 関数名, (引数の型と引数名) のあと { } で囲まれた部分が関数の定義です.
- **関数の呼び出し**: 40 行目と 44 行目でそれぞれ rewind() と makeNextTone() を呼出してます.

4.4.3 引数や返り値のある関数

例題:先の例では1音の長さを 750 msec に固定し, マクロ定義でこれを DURATION という文字列で表せるようにしていました. 音楽ではテンポは 1 分間の拍数で定めることが多いので, 拍数から待ち時間を計算する関数 duration() を作ってみましょう.

関数プロトタイプ宣言は

```
unsigned long duration(unsigned long tempo);
```

となります。

関数定義本体は

```
unsigned long duration(unsigned long tempo){
  if (tempo <= 0) {
      return 0;
  }
  return (unsigned long)60*1000/tempo;
}
```

とします. 引数 tempo は 0 だったり, 負の数だったりしてはいけないのですが, そのことを検査して, tempo の値が適切でない場合は値 0 を戻すことにします. そうでない場合は拍数に対して1拍の長さをミリ秒単位で計算して return 文で返します. 計算は int 型では桁数が足りないので unsigned long 型の演算を行なったうえで、呼び出す時は例えば 1 分間に 80 拍の場合は duration(80) と呼び出せばよいので, tone 関数の中では以下のように使います.

tone(SPEAKER, freq[fIndex], duration(80));

4.5 プログラムを改造してみる

4.2 節の回路, 4.3 節のプログラムを改造して次の機能を付加してください.

1. コンタクトスイッチを回路に加えて D2 ピンでオンオフを読み取るようにしてください.
2. コンタクトスイッチを押すことで音楽の再生, 停止を行なえるようにしてください.
3. 現在は曲については音程しか与えていませんが, 音符の長さを同時に与えられるようにしてください. 8 分音符から全音符程度を表現できるようにしてください. 音程は melody[] に 12 音階で表現したものを格納し, 音程に相当する周波数へは freq[] という配列を参照して変換しています. 同様の考え方に基づいて音符での表現とそれから時間の長さへの変換を行なうようにしてください.

5. 回路とプログラムの腕試し

5.1 今回の学習目標

- 力試し：今まで学んだものを組み合わせて課題を実行する.
  - 回路の組み立て.
  - プログラムの作成.

5.2 課題　プレゼンテーションの練習用のタイマを作成

力試しにプレゼンテーションの練習用のタイマを作成します．構成と動作は以下のとおり.

ハードウェアの構成は

- コンタクトスイッチ１つ
- LED 3 つ（LED 1, LED 2, LED3 とします）
- 圧電ブザー

動作は以下の通りです

- 初期状態ではコンタクトスイッチが押されるまで何もしません.
- コンタクトスイッチが押されると一定時間（例えば２分間，プレゼンテーション用時間），LED 1 が点灯します.
- プレゼンテーション用時間が経過するとブザーが短く１度鳴ります.
- 次の一定時間（例えば１分間，質疑用時間）は LED2 が点灯します.
- 質疑用時間が経過するとブザーが短く２度鳴ります.
- その後，LED３が点灯します.
- この間，スイッチがもう一度押されると初期状態に戻ります.

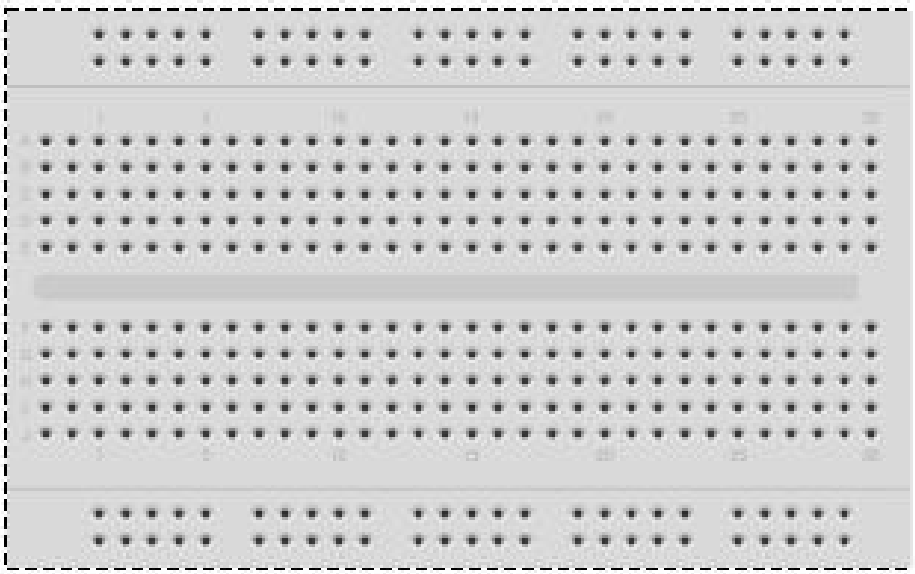
5.3 回路の作成

- ピン割り当て、LED、ブザー、スイッチの入出力をどのピンに割り当てるかを決めてください。

LED1	LED2	LED3	ブザー	スイッチ

- まず必要な部品を用いた配線を回路図として書いてください.

- 次に回路図を実現するブレッドボード上の割付を考えてください.



- 実際に回路を組み，回路図どおりかどうか確認してください．かならず配線を追いかけ、回路図通りか確認すること。

5.4　動作検証プログラムによる回路の点検

課題のプログラムを作成する前に回路の各部品の動作を確認するためのプログラムを作成，実行し Arduino から部品に適切に働きかけられるか確認してください．

- LED1～3 の動作検証プログラム
- コンタクトスイッチの動作検証プログラム
- ブザーの動作検証プログラム

Arduino では 13 番はボード上の LED に配線されている。スイッチの動作検証にはこれの点灯、消灯などを使うと良い。

LED1の動作検証プログラム	
LED2の動作検証プログラム	
LED3の動作検証プログラム	
圧電ブザーの動作検証プログラム	
コンタクトスイッチの動作検証プログラム	

5.5 プログラムの設計

まずプログラムが行うべきことを理解し，日本語で話せるようにしてください.

ここでは装置の動きを「状態」という概念で考えます.
状態としては「どの LED を点灯させなければならないか」を考えるとよいでしょう.
問：状態はいくつありますか?
問：各状態に名前と番号をつけなさい.
問：状態と点灯させる LED の一覧表を作りなさい.
問：状態を覚えておくための変数と状態を表す変数の値をきめなさい

そして,

- 各状態の時にすべきこと
- 状態を切り替えるきっかけを考えること
- 状態が切り替わったときにすべきこと

で行うべきことを整理します.

状態を切り替えるきっかけは何でしょうか?

- コンタクトスイッチが押された
- 状態が切り替わってから所定の時間が経過した

を考えればよいと思います.

状態が切り替わったときにすべきことは

- 状態を表現している変数の値を変える
- 経過時間を計測するカウントダウン用の変数を設定する
- 音をならす

を考えればよいと思います．下図のような状態の遷移を考えるといいでしょう.

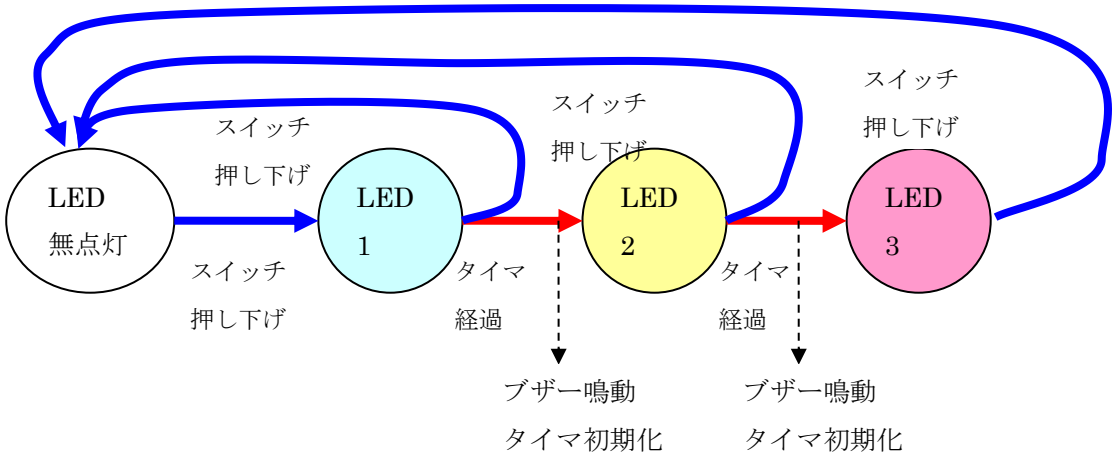


図 20

次にプログラムとしての詳細化を行ないます。

1. 入出力のポートを確認して下さい。
2. 状態やタイマの値を覚えておくために変数を決めてください。
3. 関数 setup() で行なうべき仕事をきめてください。主に以下の仕事でしょう。
   - 入出力ポートの設定
   - 変数への値の設定
4. 関数 loop() 関数で行なうべき仕事をきめてください。

   関数 loop() ではをある程度短い間隔で回しながら、
   - スイッチの押し下げ検出、
   - タイマのカウントダウン、
   - タイマ終了を検査して，
   - 状態の変化とそれに伴う必要な処理を行なえばいいでしょう．

5.6 プログラムの実装

ここまで準備ができたらプログラムとして実装します。

6. アナログ入出力

6.1 今回の学習目標

- アナログ出力(PWM)の方法を知る.
- 繰り返し処理の方法を知る.
- 変数とスコープ
- アナログ入力の方法を知る.
- PC へのデータの送出を行う.
- Arduino Uno のピン配置を知る.

6.2 回路の準備.

ブレッドボードを用いて次図のような回路を作成してください. 光センサの CdS （硫化カドミウムのセル）は中に模様のある丸い部品です. 電極に極性はありません. 片側を 5V に、他方には直列に 10kΩの抵抗をつないで GND に接続します。また可変抵抗器は両側を 5V と GND に、中央を Analog1 につなぎます。

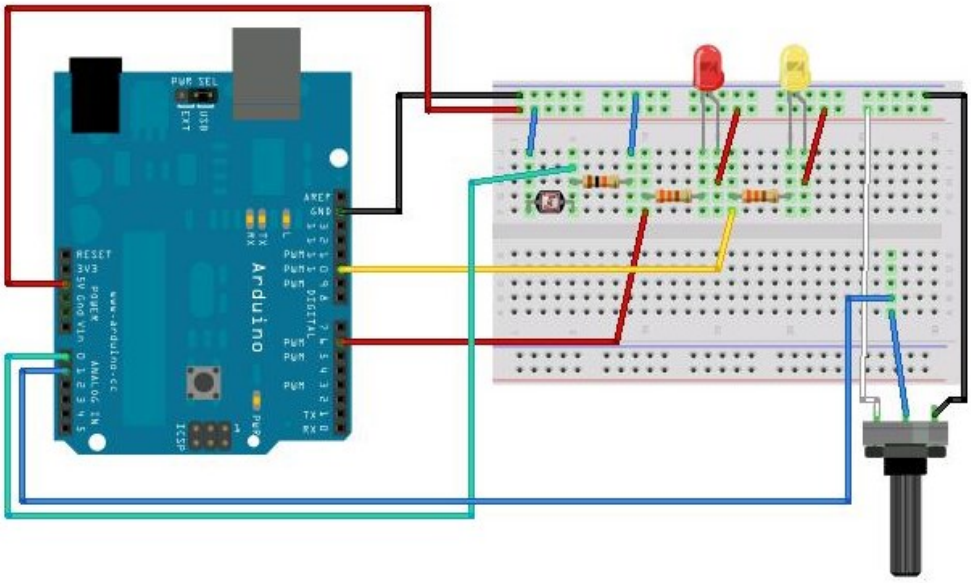


図 21

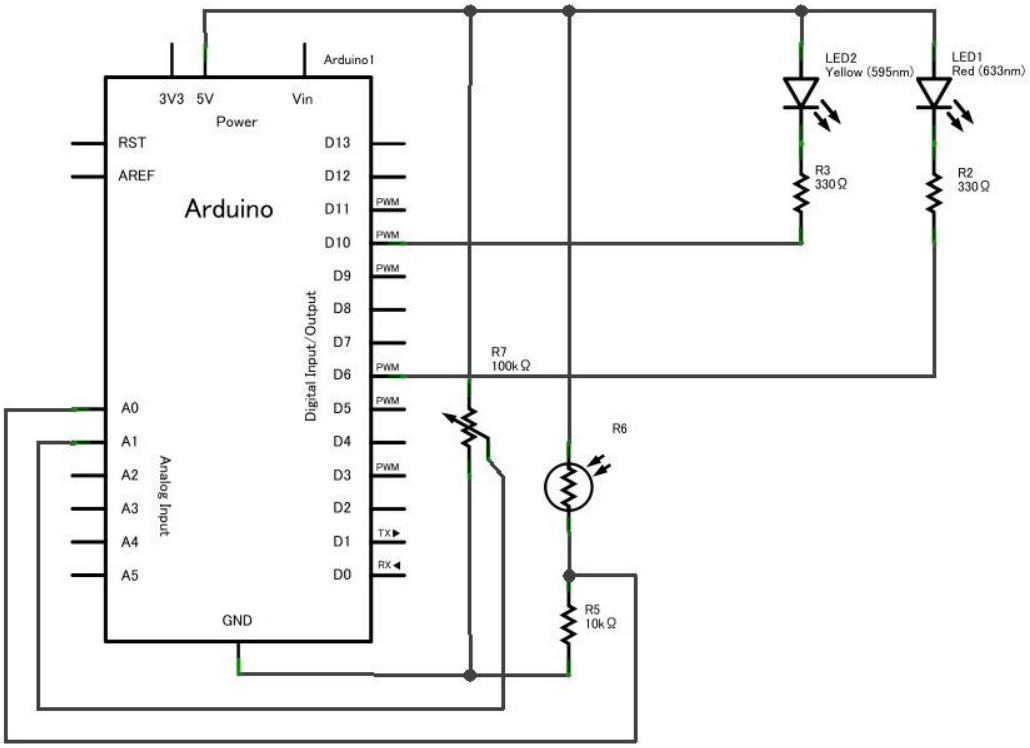

Arduino1
3V3
5V
Vin
Power
RST
AREF
Arduino
D13
D12
D11
D10
D9
D8
D7
D6
D5
D4
D3
D2
D1
D0
PWM
TX
RX
Digital Input/Output
A0
A1
A2
A3
A4
A5
Analog Input
GND
LED2
Yellow (595nm)
LED1
Red (633nm)
R3
330Ω
R2
330Ω
R7
100kΩ
R6
R5
10kΩ

図 22

6.2.1 アナログ出力の方法

次のプログラムを作成し，実行してみてください．

行番号	ソースコード	説明
1	#define LED1 6	6 番ピンを LED1 と，
2	#define LED2 10	10 番ピンを LED2 とします.
3		
4	int brightness[] =	初期値を持った配列の宣言
5	{0,10,20,30,40,50,60,70,80,90,100};	
6		
7	void setup()	
8	{	
9	pinMode(LED1, OUTPUT);	6, 10 番ピンを出力用にします.
10	pinMode(LED2, OUTPUT);	
11	}	
12		
13	void loop() {	
14	for (int i = 0; i<11; i++) {	For 文で変数 i を 0 から 10 まで 1 つづつ増やしながら実行します.
15	int br=brightness[i]*255/100;	
16	analogWrite(LED1, br);	
17	analogWrite(LED2,255-br);	analogWrite は 0 から 255 までの段階で PWM 出力を行います. LED は 5V 側につながっている（プルアップされている）ので大きな値になるほど暗くなります。
18	delay(50);	
19	}	
20	for (int i = 10; i>-1; i--) {	
21	int br=(brightness[i])*255/100;	
22	analogWrite(LED1, br);	
23	analogWrite(LED2,255-br);	今度は i を 10 から降順に 0 まで繰り返します.
24	delay(200);	
25	}	
26	}	

ソースコード 8

LED1, 2 の明るさは直線的に変化しているはずですが、人間には分かりにくいかもしれません。明るさの変化を指数関数的に変える(例えば 1,2,4,8,16…)とどうなるか試してみてください。

analogWrite() 関数は短い時間のオンオフの繰り返しにより平均値を連続的に変化させる方法(PWM)で出力する関数です. 使えるポートは 3, 5, 6, 9, 10, 11 (～が付いている)です.

6.2.2 PWM とは

Arduino のアナログ出力は短時間の HIGH (5V)と LOW(0V) を繰り返すパルス幅変調(Pulse Width Modulation, PWM) として出力されます。HIGH となる時間の比率を変えることで平均的な電圧の値を変えています。LED などを点灯する場合はこのままでも人間の視覚として平均値を感じることができますが, 実際に 0V と 5V の間の電圧を得るためには平均の値を出力する回路を付加する必要があります。

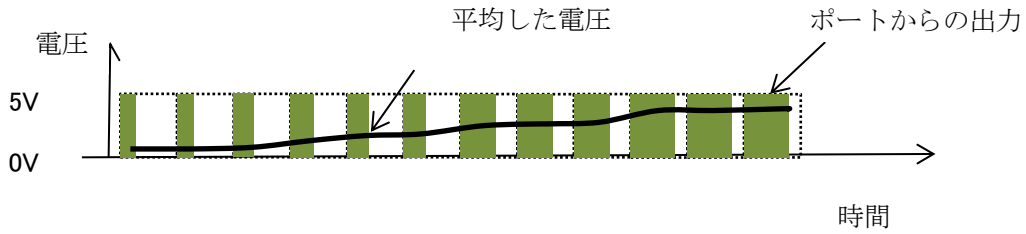


図 23　パルス幅変調（ＰWM）

２．節のプログラムで新しく出てきたプログラムの概念に繰り返し処理があります.

6.2.3 for 文

動作を繰り返すためによく使う構文が for 文です．For は「～の間」という意味にとればいいでしょう．以下のような表記をします.

```
for（初期化；継続条件；繰り返し時の操作）｛
  繰り返す内容
｝
```

for 文は以下のように訳読するとよいでしょう.

「初期化」から初め,「継続条件」が成り立っている**間**,「繰り返し時の操作」を**しながら**「繰り返す内容」を繰り返す.

例えば

```
for (int i = 0; i<11; i++) {
 …
}
```

は

```
for ( int i=0;          i<11;          i++) { … }
  ↓                     ↓              ↓
変数 i を 0 から，i が 11 未満の間，i を 1 づつ増やしながら…を繰り返す
```

と読めばいいでしょう. ここで**「 ++ 」は変数の値を 1 だけ増やす演算**です．不思議な記号ですがプログラムでは変数を1だけ増やすことが多いので、こういう表記が導入されています．1減らす演算は「--」です.

6.2.4 変数のスコープ

２節のプログラムでは配列 brightness は関数 setup() や loop() の外側で宣言されており．**「外部変数」**とか**「グローバル変数」**と呼ばれます．外部変数は

- プログラムの実行中は永続的に存在し，値を保持するとともに,
- どの関数からでも読み書きできます.

一方，loop() 関数の中の br や for 文の中の i などは**自動変数**とか**ローカル変数**と呼ばれ，

- 当該の関数が呼ばれている間あるいは for 文が繰り返し動作をしている間だけ存在します.
- また，他の関数からは読み書きできません.

6.3　可変抵抗器、CdS セルからの入力

CdS セルは明るさに応答して電気抵抗が変化する（明るいと下がる）素子です．図 22 の回路では明るいと analog 0 ピンの電圧が高く，暗いと低くなります．CdS セルで明るさを検出してスイッチ代わりに使いましょう．次のプログラムを実行してみてください．

ここでは PC 側とプログラムの実行時にもデータのやりとり（シリアル通信）をします．プログラムのアップロードが終わったら tools → serial monitor を選びます．Arduino 側で検出した A0 番ピンの信号レベル（0 ～ 1023）の値が表示されるはずです．

行番号	ソースコード	説明
1	#define LED1 6	
2	#define LED2 10	
3		
4	void setup()	
5	{	
6	Serial.begin(9600);	PC と通信するための初期設定
7	pinMode(LED1, OUTPUT);	
8	pinMode(LED2, OUTPUT);	
9	digitalWrite(LED1, HIGH);	最初は消灯しておきます
10	digitalWrite(LED2, HIGH);	
11	}	
12		
13	void loop() {	
14	int val1 = analogRead(0);	A0, A1 ピンの電圧を読み込みます.
15	int val2 = analogRead(1);	
16	Serial.print(val1,DEC);	読み込んだ値を PC に 10 進数表記で送ります.
17	Serial.print(″ ″);	
18	Serial.println(val2,DEC);	読み込んだ値 val1 の大小で LED を点灯，消灯します．閾値の 600 は適宜，調整してください.
19	if (val1>600) {	
20	digitalWrite(LED1, LOW);	
21	} else {	
22	digitalWrite(LED1, HIGH);	読み込んだ値 val2 の大小で LED を点灯，消灯します．閾値の 600 は適宜，調整してください.
23	}	
24	if (val2>600) {	
25	digitalWrite(LED2, LOW);	
26	} else {	
27	digitalWrite(LED2, HIGH);	

```
28      }
29      delay(50);
    }
```

ソースコード 9

6.3.1 アナログ入力

A0 ～ A5 に入力された電圧（0 から 5 V の範囲であること）を読み取って 0 ～ 1023 の値で返す関数が analogRead(pin) です．ピン番号は 0 から 5 です．

6.3.2 PC との通信

Arduino から USB ケーブルで接続された PC にデータを送ることができます．Arduino 内部での計算が正しく行われているかどうかはそのままでは確認すべきすべがありません．PC 側に変数の値などを書き出して確認する方法を知っておくことは重要です．

- Serial.begin(9600); シリアル通信を開始するための設定です．()内の数字は通信速度で 9600 は1秒間に 9600 bit 送る(大体 1000 文字程度)というものです．
- Serial.print(val1,DEC); val1 という変数の値を 10 進数(decimal number)表記で書き出します．
- Serial.print(″ ″); ″（２重引用符）で囲まれた文字列を書き出します．
- Serial.println(val2,DEC); val2 という変数の値を 10 進数(decimal number)表記で書き出し、行末で改行します．
- PC 側では Arduino ソフトウェアの Tools → Serial Monitor というメニューを選べば送られてくる内容を確認できます。

6.3.3 map() 関数を使う

アナログの変数の変域を調整するのに便利な関数が

```
map(value, fromLow, fromHigh, toLow, toHigh)
```

です．例えば 0 から 1023 までの値をとる変数 x (analogRead() で得られるものはこの範囲です)の値を 0 から 255 に変換する（analogWrite()で許される範囲）に変換するには

```
y = map(x, 0, 1023, 0, 255);
```

とします．数値の桁あふれをなるべく起さないように long 型の演算が行われます．fromLow と fromHigh，toLow と toHigh それぞれの組み合わせで必ずしも Low が High より小さくなければならないという制限はなく範囲の上下を逆転させる使い方もできますが，fromLow と fromHigh の値が同じだと 0 による割り算でエラーを生じます．

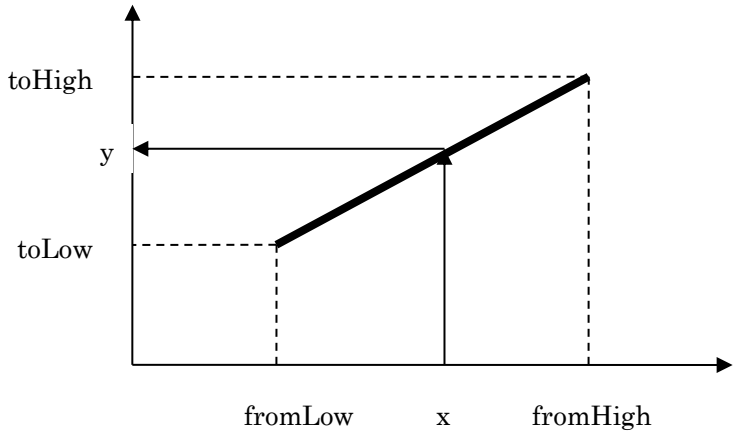


図 24 map() 関数による値の範囲の変換

7. LED の使い方

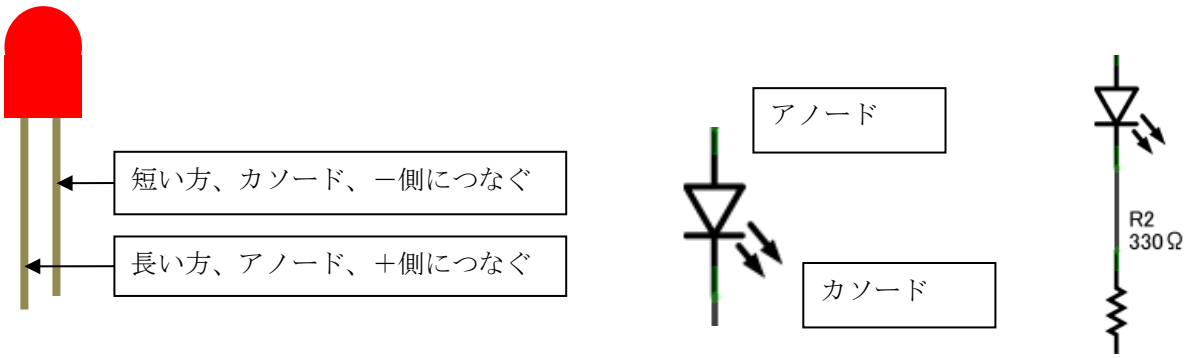


図 25 LED のピンと回路記号

LED は適正な電圧，電流で使わなければなりません．

- **LED の足の長いほうがアノード，短いほうの足がカソードと呼ばれ，点灯する際にはアノード側を高い**電圧にします．
- 赤色の LED が点灯する電圧は 1.8V，青色や白の LED が点灯する電圧は 3.5V 程度です．
- また砲弾型の LED に流すべき電流は 10mA 程度です．

このため，LED を直接，5V の電源線と GND につなぐと電流が流れすぎて LED を破損します．

適切な電流，電圧で使用するためには LED に直列に 150 Ω（青），330 Ω（赤～緑）といった値の抵抗器を直列につなぎます．

例えば 330 Ωの抵抗器に 10mA の電流を流すと電圧降下は 3.3 V になります．したがって 5V の電源からの差が 1.7 V になり，想定した電流値で概ね適正な電圧が LED に加わることになります．

8. 抵抗のカラーコード

抵抗器には４本の色が塗られていますが、これは抵抗器の抵抗値と精度を表しています。

8.1 カラーコード表

色	１，２桁目の数	乗数	精度
黒	0	10^0 = 1	
茶	1	10^1 = 10	
赤	2	10^2 = 100	
橙	3	10^3 = 1, 000	
黄	4	10^4 = 10, 000	
緑	5	10^5 = 100, 000	
青	6	10^6	
紫	7	10^7	
灰	8	10^8	
白	9	10^9	
金		10^{-1}	±５%
銀		10^{-2}	±１０%
無着色			±２０%

8.2 カラーコードの読み方

抵抗器の抵抗値は

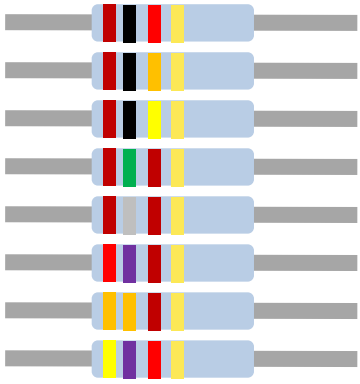
（（１番目の色の数字）×１０＋（２番目の色の数値））×（３番目の色の乗数）Ω

になります。４番目の色が精度を表します。

図 26 カラーコードの読み方

8.3 抵抗値の例

電子工作によく使うものの例をいくつか示しておく。



茶、黒、赤 → $10\times10^2 = 1\text{k}\Omega$

茶、黒、橙 → $10\times10^3 = 10\text{k}\Omega$

茶、黒、黄 → $10\times10^4 = 100\text{k}\Omega$

茶、緑、茶 → $15\times10 = 150\Omega$

茶、灰、茶 → $18\times10 = 180\Omega$

赤、紫、茶 → $27\times10 = 270\Omega$

橙、橙、茶 → $33\times10 = 330\Omega$

黄、紫、赤 → $47\times10^2 = 4.7\text{k}\Omega$

8.4 Ｅ１２系列

市販されている抵抗器の抵抗値はＥ１２系列という値で設計されている。これはカラーコードの２桁の値が 10, 12, 15, 18, 22, 27, 33, 39, 47, 56, 68, 82 を取るもので、不規則に見えるが、10 倍になる範囲を概ね対数で１２等分するように定められている。こうすることで値の絶対値に寄らず、一定の精度の部品が入手できるようにしたものである。電子回路を設計する場合は、利用可能な部品で抵抗器の値を定める必要がある。

9. Arduino の電源

9.1.1 Arduino への電源供給

Arduino への電源の供給は2種類あります。

1. PC に USB 接続されているときには USB ケーブル経由で PC から供給される電圧 5V の電源が使われます。最大 500mA 程度までしか使えません。
2. 外部電源端子経由で供給される電池などの電源. ここに電源が接続されている場合、Arduino のボード上でコンピュータ用の電圧 5V に変換して使われます. どちらの電源が使われるかは接続されている電源に応じて Arduino が自動的に切り替えます。
   つなぐ電源は 7V ～ 12V の範囲です。7Vより低いと動作が不安定になり、12Vより高いと過熱してボードを壊す可能性があります。

サーボモータなど大きな電流が必要となる場合や PC から取り外して使う場合には外部電源を接続しなければなりません。

9.1.2 Arduino の電源の外部への取り出し

ブレッドボード上で電子回路などを作る際には回路への電源供給法として以下のような方法があります。

- Arduino 上の 5V の電源を使う。「5V」と書かれたピンから取り出します。
- Arduino 上の 3.3V の電源を使う。電子部品によっては 3.3V の電源が必要なものがあります。「3.3V」と書かれたピンから Arduino 上で 3.3V に変換された電源を取り出せます。ただし電流は最大 50mA しか使えません。
- Arduino に供給されている外部電源をそのまま使う。Vin というピンは Arduino 内部で外部電源端子と直結されています。外部電源端子につないだ電源をこの端子から使えます。
- 別途、電源を供給する。大きな電流や異なる電圧の電源を使いたい場合は、別途、電源を用意します。

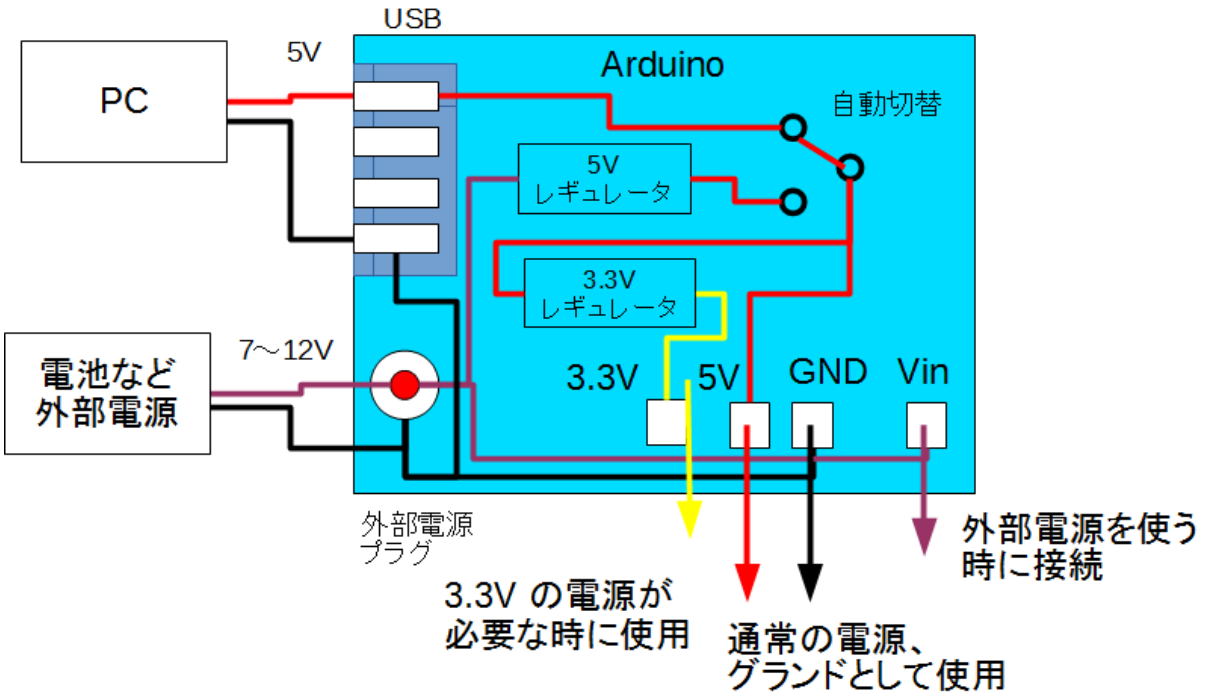


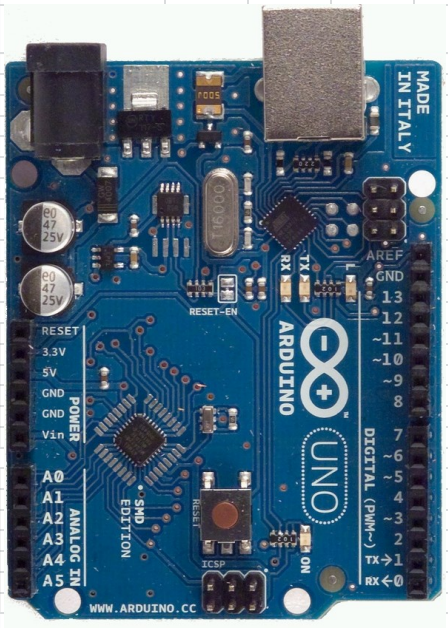
図 27 Arduino の電源

10. Arduino のピン配置

Arduino Uno のピン配置は図 28 のようになっています. デジタル入出力は 0 ～ 13 の 14 本ありますが, このうち 0 と 1 は USB での通信用に使われており, 13 は予めボード上で LED が配線されています. これらのピンを使うときは要注意です. またこれらのうち 3, 5, 6, 9, 10, 11 は analogWrite() 関数での PWM 出力が可能です.

Arduino Uno のピン配置

外部電源入力　　USB



	リセット
電源出力	3.3V
	5V
グランド	
グランド	
外部電源出力	

	A0
	A1
	A2
	A3
	A4
	A5
	アナログ入力

グランド	電気回路の基準となる電圧(0V)の配線です.
電源出力 3.3V	Arduino の中でグランドに対して 3.3V に制御されている電源出力です. 最大 50mA
電源出力 5V	Arduino の中でグランドに対して 5V に制御されている電源出力です.
外部電源出力	外部電源入力から直接配線されています.
外部電源入力	電池などをつなぐコネクタです. 中央がプラスの
参照電圧	アナログ入力時の参照電圧の入力です.

参照電圧	
グランド	
13	内臓LEDに接続
12	
11	PWM出力可
10	PWM出力可
9	PWM出力可
8	
7	
6	PWM出力可
5	PWM出力可
4	
3	PWM出力可
2	
1	USB送信に利用
0	USB受信に利用
デジタル入出力	

図 28 Arduino のピン配置