Arduinoのsyntax

以下をコピペ Arduino(C/C++言語）でのデータ操作

変数

変数は値を格納するメモリ上の領域に名前を付けたものです。C/C++言語で変数を利用するためには、事前に変数を定義する必要があります。この際、変数に格納する情報に応じて適切に「型」を定義する必要があります。変数の型には、C/C++言語が規定している型とArduinoソフトウェアで独自に定義している型の2種類があります。

以下では、char型の変数c、int型の変数i、double型の変数d、word型の変数wを定義します。iについては0で初期化しています。

```
char c;
int i = 0;
double d;
word w;
```

同じ型の変数はまとめて定義することもできます。「,」で区切って変数を並べます。

```
int l, m, n;
```

言語仕様による定義

C/C++で利用できる基本的な型は、以下に大別することができます。

1. 真偽値を表す型
2. 1バイト(1文字)を表す型
3. 整数を表す型
4. 浮動小数点数を表す型
5. 複素数を表す型

整数と浮動小数点数、複素数を表す型は表すことができる数値の範囲により細分化されます。また、1バイトを表す型と整数型は正負の数値を表現できる型(符号付)と、非負数だけを表現できる型(符号なし)があります。具体的には以下の通りです。

型名	説明	Arduino Uno		Arduino Due	
		変数が占めるサイズ(sizeof())	取り得る値	変数が占めるサイズ(sizeof())	取り得る値
bool（C言語） bool（C言語）	bool（C言語） bool（C言語）	bool（C言語） bool（C言語）			
_Bool(C言語)	真偽値(trueとfalse)を格納する。	1	true/false(実際には他の値をとることも可能)	1	true/false(実際には他の値をとることも可能)

型名	説明	Arduino Uno		Arduino Due	
		変数が占めるサイズ(sizeof())	取り得る値	変数が占めるサイズ(sizeof())	取り得る値
char	1バイトの値を格納する。文字を格納する。	1	-128～127	1	-128～127
unsigned char	1バイトの値を格納する。文字を格納する。	1	0～255	1	0～255
short int	整数を格納する。	2	-32768～32767	2	-32768～32767
unsigned short int	非負整数を格納する。	2	0～65535	2	0～65535
int	整数を格納する。	2	-32768～32767	4	-2147483648～2147483647
unsigned int	非負整数を格納する。	2	0～65535	4	0～4294967295
long int	整数を格納する。	4	-2147483648～2147483647	4	-2147483648～2147483647
unsigned long int	非負整数を格納する。	4	0～4294967295	4	0～4294967295
long long int	整数を格納する。	8	-9223372036854775808～9223372036854775807	8	-9223372036854775808～9223372036854775807
unsigned long long int	非負整数を格納する。	8	0～18446744073709551615	8	0～18446744073709551615
float	浮動小数点数を格納する。	4	-3.4028235e+38～3.4028235e+38	4	-3.4028235e+38～3.4028235e+38
double	浮動小数点数を格納する。	4	-3.4028235e+38～3.4028235e+38	8	-1.7976931348623 2e308～1.79769313486232e308
long double	浮動小数点数を格納する。	4	-3.4028235e+38～3.4028235e+38	8	-1.79769313486232e308～1.79769313486232e308
float _Complex	複素数型を格納する。	8	-	8	-
double _Complex	複素数型を格納する。	8	-	16	-
long double _Complex	複素数型を格納する。	8	-	16	-

short intとlong int，long long intは、それぞれ，short，long，long longと表すことができ、通常はそのように記述します。

Arduino UnoとArduino Dueとで変数の大きさが異なることでもわかるように、同じ型名でも実際に表現できる値の範囲はプログラムを実行する環境により異なります。このため整数を表す型については、その型が占める大きさをもとにした型(幅指定整数型)が提供されています。プログラムの移植性を考慮する際などによく利用されます。

型名	説明
int8_t	8ビットの整数を格納する。
uint8_t	8ビットの非負整数を格納する。
int16_t	16ビットの整数を格納する。

型名	説明
uint16_t	16ビットの非負整数を格納する。
int32_t	32ビットの整数を格納する。
uint32_t	32ビットの非負整数を格納する。
int64_t	64ビットの整数を格納する。
uint64_t	64ビットの非負整数を格納する。

変数を利用する際は、必要な大きさをもつ変数を選ぶ必要があります。適切な大きさの変数を利用しないと、数値が入りきらなかったり(オーバーフロー)、不必要なメモリ領域を確保したりしてしまいます。

なお long long型は現状Printクラスが対応していないため、例えば Serial.print()を使って値をコンソールに表示することはできません。

Arduinoソフトウェアによる独自定義

Arduinoソフトウェアでは C/C++言語の機能を利用して以下の型を独自に定義しています。

型名	説明	Arduino Uno		Arduino Due	
		変数が占めるサイズ(sizeof())	取り得る値	変数が占めるサイズ(sizeof())	取り得る値
boolean	bool型の別名。真偽値格納用。	1	true/false	1	true/false
byte	uint8_t型の別名。	1	0～255	1	0～255
word	unsigned int型の別名。	2	0～65535	4	0～4294967295
String	文字列。	-	-	-	-

変数の初期化

変数の宣言と同時に変数に値を代入することができます。変数名の後に、「= 初期値」を記述します。

```
int i = 0;
```

変数の有効範囲と記憶域期間

定義した変数には有効範囲と記憶域期間(変数の存続する期間)があります。このセクションは前方参照が多いので後からもう一度読んでもらうとよりわかりやすいと思います。

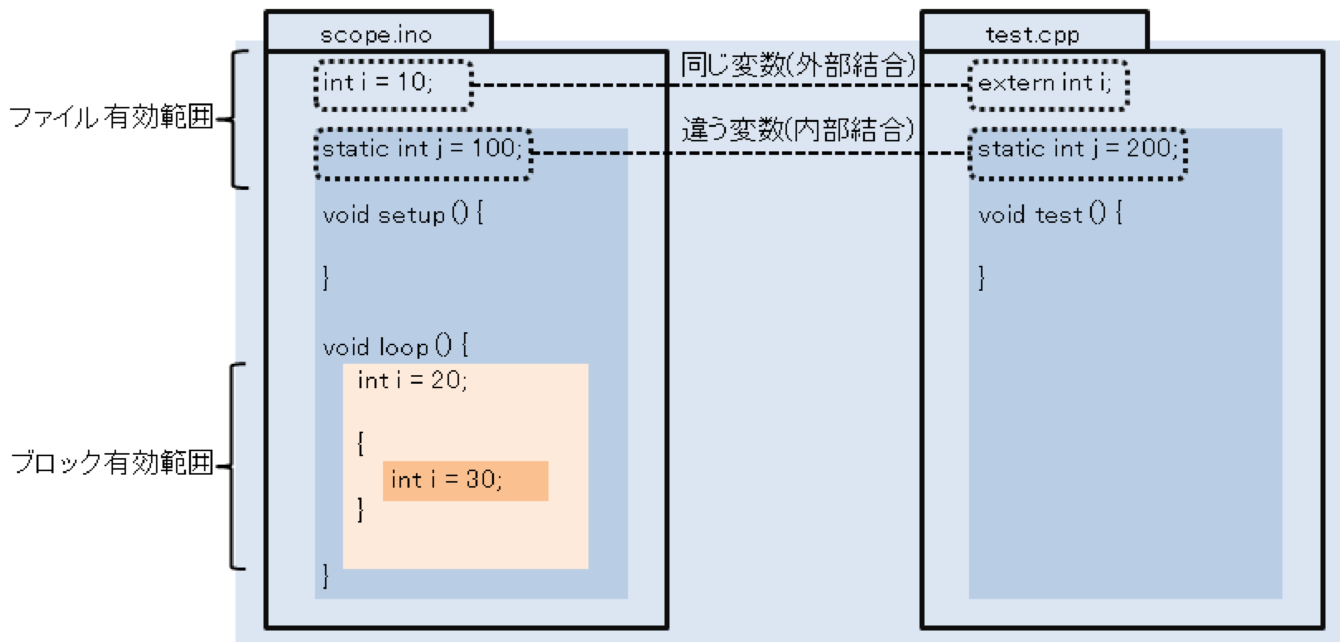
有効範囲

変数の有効範囲には以下の2種類が存在します。

有効範囲	意味	指定方法
ファイル有効範囲	同一のソースファイル内で有効	全てのブロックの外側で定義する。
ブロック有効範囲	同一のブロック内で有効	ブロックの内側で定義する。

ブロックの中にブロックがある場合、同じ名前で変数を定義してしまうと、外側のブロックの変数はそのブロックの中では見えなくなってしまいます。ファイル有効範囲を持つ変数については、変数名の前に「::」をつけることで、参照することができます。ただし、このようなプログラムは見通しが悪くなるのでお勧めはしません。

ファイル有効範囲を持つ変数を、他のソースファイルから参照する仕組みも用意されています。ファイル有効範囲を持つ変数を定義する際にstaticというキーワードをつけると、そのソースファイルの中でだけ有効な変数となります(内部結合)。externというキーワードをつけるか何もつけない場合は、他のソースファイルとの間で同じ変数を共有することができます(外部結合)。以下に概念図を示します。



以下では、scope.inoとtest.cppを同一のスケッチとして定義しています。実際に試してみる場合は、scope.inoとtest.cppを同一のディレクトリに置いてください。もしくは、Arduinoソフトウェアのファイル名が表示されている行の右側にある「▼」をクリックして、「新規タブ」を選択して新しいタブを追加してください。

scope.ino

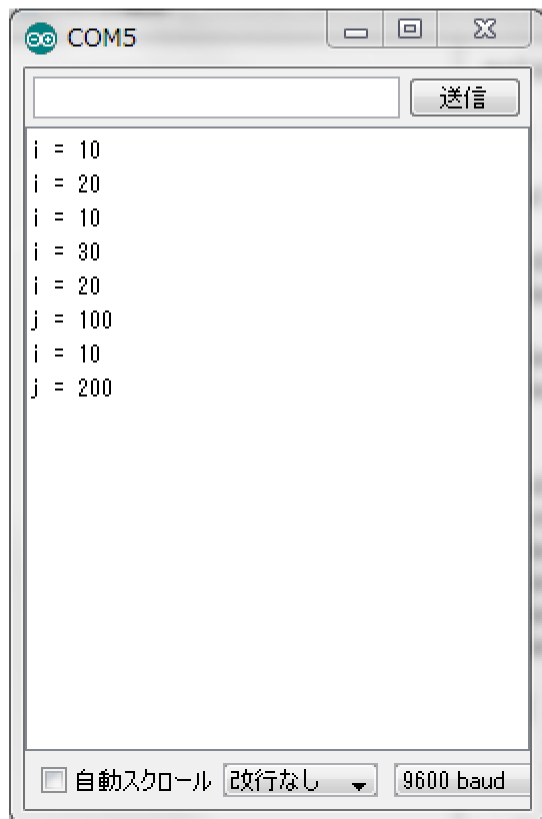
```
extern void test();

int i = 10;                    // (1)  ファイル有効範囲・外部結合

static int j = 100;            // (2)  ファイル有効範囲・内部結合

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

  Serial.print("i = ");
  Serial.println(i);
}

void loop () {
  int i = 20;                  // (3)  ブロック有効範囲
  Serial.print("i = ");
  Serial.println(i);          // (3)のi
```

```
  Serial.print("i = ");
  Serial.println(::i);      // (1)のi

  {
    int i = 30;               // (4)  ブロック有効範囲
    Serial.print("i = ");   // (4)のi
    Serial.println(i);
  }

  Serial.print("i = ");
  Serial.println(i);        // (3)のi

  Serial.print("j = ");
  Serial.println(j);

  test();
  while(1);
}
```

test.cpp

```
#include <Arduino.h>

extern int i;        // (1)のi

static int j = 20    // (2)のjとは異なる

void test () {
  Serial.print("i = ");
  Serial.println(i);

  Serial.print("j = ");
  Serial.println(j);
}
```

上記プログラムの実行結果を以下に示します。



記憶域期間

記憶域期間には、静的記憶域期間、自動記憶域期間、割付け記憶域期間の3種類があります。それぞれの変数の存続する期間と確保(指定)方法、デフォルトの初期値は以下の通りです。

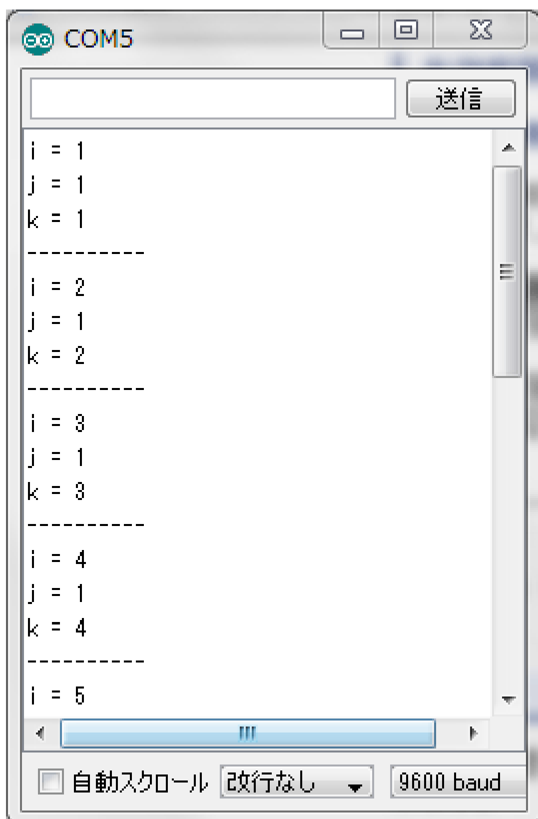
記憶域期間	存続期間	確保(指定)方法	デフォルトの初期値
静的記憶域期間	プログラムの開始から終了まで	関数の外側で定義する。	静的記憶域期間
staticをつけて定義する。	0	staticをつけて定義する。	0
自動記憶域期間	定義をした時点から有効範囲を離れるまで。	ブロック内でstaticをつけないで定義する。	不定
割付け記憶域期間	割付けてから解放するまで	malloc()等で確保する。	割付け方法に依存する。
malloc()□不定	malloc()□不定	malloc()□不定	malloc()□不定
calloc()□0	calloc()□0	calloc()□0	calloc()□0

以下のプログラムでは、1行目と9行目のiとkは静的記憶域期間を持ちます。8行目のjは自動記憶域期間です□Arduinoでは割付け記憶域期間を利用することはあまり多くはないと思います。

```
int i = 0;                    //静的記憶域期間：プログラムの開始時に1回だけ初期化

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

void loop () {
```

```
  int j = 0;              //自動記憶域期間：関数が呼ばれるたびに初期化
  static int k = 0;       //静的記憶域期間：プログラムの開始時に1回だけ初期化

  i++, j++, k++;

  Serial.print("i = ");
  Serial.println(i);
  Serial.print("j = ");
  Serial.println(j);
  Serial.print("k = ");
  Serial.println(k);
  Serial.println("----------");

  delay(1000);
}
```

上記のプログラムを実行すると，jだけはloop()が呼ばれるたびに0に初期化されますが，iとkは初期化されずにインクリメントされ続けることがわかります。



変数の格納場所

Arduino Unoでは通常変数はSRAMに配置されます。ただし，SRAMは2キロバイトしかありません。一方，Flashメモリは32キロバイトあります。ただし，Flashメモリはプログラムをロードしたときに値を設定できますが、その後値を変更することはできません。大量の文字列や固定の配列などを使うときはSRAMを節約するためにFlashメモリに変数を配置することができます。このための拡張をavr-gccは行っています。

配列

C/C□□言語での配列は同じ型の変数を複数個連続して確保するための手段です。変数名の後ろに“ [要素数] “ をつけて変数を宣言します。配列を構成する元の型を要素型と呼びます。

以下にshort型の変数を6個定義する例を示します。

```
short analog[6];
```

C/C□□言語では配列の番号は0から始まります。このため、上記の例では□analog[0]□analog[1]□analog[2]□analog[3]□analog[4]□analog[5]の6個の変数としてアクセスすることができます□analog[0]からanalog[5]までのそれぞれはshort型です。

C/C□□言語では配列の範囲を超えたアクセスに対するチェックは行われません。例えば、上記の例において□analog[6]にアクセスするようなプログラムを書いてもコンパイルは正常に終了します。配列の範囲のチェックはプログラマの責任で行う必要があります□Arduinoではプログラム実行時の予期しないエラーを伝える手段がないため充分注意する必要があります。

```
short analog[6];

a[6] = 0;   /* コンパイルは正常終了 */
```

配列を定義したときに同時に初期化することもできます。初期化する値を配列名の後に並べて書きます。下記の例を見てわかるとおり、初期値を指定する場合は、配列の要素数を省略することができます。

```
short analog[6] = {0, 1, 2, 3, 4, 5};
short digital[] = {0, 1, 2, 3, 4, 5};
```

定数

C/C□□言語で利用できる定数の説明をします。定数は変数に代入することができます。

整数定数

プログラムの中で整数を表す整数定数は以下のようなものが利用できます。

種類		意味、表記法	例
整数定数	10進定数	10進数を表します。	0□1□-10
	16進定数	16進数を表します。数値の先頭に「0x□もしくは□0X□をつけます。10進数の10から15については、アルファベットのaから□もしくはAからFを使います。	0xab□0XFFFF
	8進定数	8進数を表します。数値の先頭に「0」をつけます。	0123□07654
	2進定数		

```
enum direction {left, right};
```

```
enum direction d = 10; //エラー
```

データに対する操作

データに対する操作を以下に示します。

乗除演算、加減演算

いわゆる四則演算と剰余が可能です。

演算子	意味
*	掛け算
/	割り算
%	剰余(整数に対してだけ適用可能)
+	足し算
-	引き算

整数同士の割り算では小数点以下は切り捨てらます。以下の例でわかるように、例えば20.0と20とは扱いが異なります。

```
20 * 30;  // 600
20.0 / 3; // 6.6666…
20 / 3;   // 6
20 % 3;   // 2
20 + 3;   // 23
20 - 3;   // 17
```

ビット単位の演算

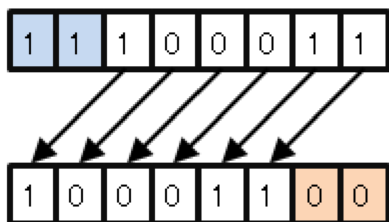
char型を含む整数型に対してビット単位で演算を行うことができます。

演算子	意味	
«	左シフト演算	
»	右シフト演算	
&	ビット単位のAND演算/ビット積算	
		ビット単位の排他OR演算/ビット差演算
		ビット単位のOR演算/ビット和演算
~	ビット単位の補数演算(この演算子は単項演算子)	

例えば unsigned char型の変数は8ビットの大きさを持つ正数(0～255)を表します。これを一つの数値としてではなく、各ビットごとに演算を行うことも可能です。例えば、デジタルピンの値はHIGH(1)もしくはLOW(0)の2種類の値をもつだけなので、一つのピンごとに変数を用意するのではなく、ひとつのピンの値を1ビットに対応させて保持することも可能です。

左シフト演算・右シフト演算

左シフト演算と右シフト演算は、もとの値を指定したビット数だけ左もしくは右にずらした値を返します。空いたビットは0となります。例えば unsigned char型(8ビット幅)の0b11100011 « 2 は、0b10001100となります。



ずらすビット数が元の値のもつビット数より大きい場合や負の値を指定した場合は、C/C□□言語の動作としては未定義です□Arduinoでは、それぞれ、0、逆のシフト操作となるようですがそのような演算は行わないようにしてください。

ビット演算

ビット単位のAND演算(&)・排他OR演算(^)□OR演算(|)は、両辺の値のそれぞれの対応するビット単位で演算を行います。演算の規則は以下の通りです。

<table>
<tr><th>演算子</th><th>規則</th></tr>
<tr><td>AND演算(&)</td><td>双方が1であれば1、そうでなければ0。<table><tr><td>&</td><td>0</td><td>1</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>1</td><td>0</td><td>1</td></tr></table></td></tr>
<tr><td>排他OR演算(^)</td><td>双方が異なれば1、そうでなければ0。<table><tr><td>^</td><td>0</td><td>1</td></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table></td></tr>
<tr><td>OR演算(|)</td><td>どちらかが1であれば1、そうでなければ0。<table><tr><td>|</td><td>0</td><td>1</td></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table></td></tr>
</table>

補数演算

補数演算は、各ビットを反転させます。

代入

変数には値を代入することができます。代入にはいくつかの手段があります。

まずは、すでに何回か出ている「=」を使う方法です。「=」は左辺と右辺が等しいという意味ではなく、右辺の値(計算結果)を左辺に代入するという意味です。

以下ではint型の変数iに10をfloat型の変数fに12.3を代入しています。

```
int i;
float f;

i = 10;
f = 12.3
```

また、関数が戻り値を返す場合、関数の実行結果を代入することもできます。例えばanalogRead()という関数はint型を返す関数です。以下ではanalogRead()の結果を変数valueに代入しています。

```
int value;
value = analogRead(A0);
```

プログラムを書く際にはよく、変数の値を操作してその結果をもとの変数に代入することがありますC/C++言語では、「*=、/=、%=、+=、-=、<<=、>>=、&=、^=、¦=」という代入演算子が用意されています。例えばiに5を足した結果をiに代入するというプログラムは、以下のように書くことができます。

```
int i = 0; // iを0で初期化する。

i = i + 5;  // iに5を足した結果をiに代入する。
i += 5;     // iに5を足した結果をiに代入する。

i = i * 5;  // iに5を掛けた結果をiに代入する。
i *= 5;     // iに5を掛けた結果をiに代入する。
```

代入を実行した結果そのものが代入演算の結果となります。例えばi = 10という代入が行なわれるとiに10が代入されるとともに、この式自体の評価結果も10となります。

インクリメント・デクリメント

指定した変数の値を1増やす(インクリメント)「++」、1減らす(デクリメント)「–」という演算子が利用できます。++と–には、それぞれ、 変数の前に記述する前置と変数の後に記述する後置の2種類があります。変数の値が1増える、1減るという結果は変わりませんが、式を評価したときの結果が異なります。

```
int i, j;
i = 0;
j = 0;
k = 0;

i++;      // i = 1
```

```
++i;        // i = 2
j = i++;  // i = 3, j = 2
k = ++i;  // i = 4, k = 4
```

i++の評価結果はiそのもので、その後iが1増やされます。++1の場合は、iを1増やした結果が評価結果です。

From:
http://www.deepsky.jp/wiki/ - うごくといいな

Permanent link:
http://www.deepsky.jp/wiki/doku.php?id=elechobby:arduino:syntax&rev=1760827746

Last update: **2025/10/19 07:49**