講習会

Arduino

2回目

変数と演算

目的

- 変数を扱えるようにする
- 演算できるようにする
- 変数と演算の結果をモニターに映せるようにする

変数

- プログラミングにおいて、変数（へんすう、variable）とは、プログラムのソースコードにおいて、扱われるデータを一定期間記憶し必要なときに利用できるようにするために、データに固有の名前を与えたものである。 一人一人の人間が異なる名前によって区別されるように、一つ一つの変数も名前によって区別される。これにより、複数のデータを容易に識別することができる。変数を区別するための名前を特に識別子という。また一般に、変数が表しているデータをその変数の値（あたい）という。

変数の宣言

- プログラムの中でどのような名前の変数を用いるのかを、プログラミング言語の文法にのっとって明確に示すことを変数の宣言（せんげん）という。
- いくつかのプログラミング言語では、変数を宣言する際にその名前だけでなくそのデータ型も指定する必要がある。これにより、各変数が扱うことのできるデータの種類を制限できる。一般に、データに対して行える処理はデータ型によって異なるので、データ型を厳密に検査することで、誤ったプログラムを書いてしまうことを防止するのに役立つ。
- また一部のプログラミング言語では、型推論といって、プログラムで変数の型を明示しなくても、処理系が自動的に型を判断する機能がある。多くの場合、型推論は変数の宣言と代入が一度にまとめて行われることが前提となる。

変数の代入

- 宣言した変数に対して実際にデータを関連付けることを代入（だいにゅう）という。
- プログラミング言語によっては、変数の宣言と代入を一度にまとめて行うことができる。 変数を宣言せずにいきなり代入できる言語もあるが、これは宣言されていない変数に対して処理系が自動的に宣言を補ってくれていると考えることが出来る。 ある変数に対して初めて行う代入は、特に初期化（しょきか）という。
- 多くの手続き型言語では、変数は複数回代入をすることができる。すでに代入を行った変数に改めて代入をすると、その変数とそれまでのデータとの関連はなくなり、新しいデータと改めて関連付けされる。 関数型言語では、一つの変数には一度しか代入できないものも多い。このような言語では、宣言と初期化を一緒に行うのが一般的であり、また、一つの変数が常に同じ値を持つ（値が変化しない）ことが保証される。

変数の参照

- その変数に代入したデータを利用することを、変数の参照（さんしょう）という。
- 一度も代入を行っていない、つまり初期化していない変数を参照することは意味を成さず、不正である。 しかし一部のプログラミング言語では、変数を宣言した段階で、自動的に何らかのデータが変数に関連付けられる。このような言語では、初期化を明示的にプログラムしないまま変数を参照できる。

変数の定義と代入の例

- int ledPin = 13;　13をledPinに代入
  （ledPin + 1をすれば14になる）

  int var = val;

  var　int型の変数名
  val　その変数に代入する値

変数の参照の例

```
int ledPin = 13;
```

x >= ledPin
（xはledPin（13)以上）
x = ledPin + 1
（xにはledPin（13）に1を足したものを返す）

一般的な変数の定義

- 変数を定義するならデータ型を使えば何とかなる
- ここではデータ型の定義の書き方について教える

種類

- int
- long
- char
- byte

int

- 整数型は数値を格納するための主要なデータ型である。
- Arduino Uno(と他のATmegaベースのボード)では、intは16ビット(2バイト)の値を格納することができる。よって、-32768から32767(-2^15から(2^15)-1)の範囲の整数を表す。
- Arduino Dueでは、intは32ビット(4バイト)の値を格納することができる。よって、-2147483648から-2147483647(-2^31から(2^32)-1)の範囲の整数を表す。
- 負数は、2の補数と呼ばれる形式で表される。最上位ビットは符号ビットとも呼ばれ、1のときには負を表す。残りのビットを反転し1を足した値が絶対値である。
- Arduinoは期待したとおりに数値演算が行われるように負数を取扱う。ただし、ビット単位の右シフト演算子(>>)の動作は複雑なので注意すること。

long

- longは、数値を保持するための、大きさが拡張された型である。32ビット(4バイト)の大きさを持ち、-2147483684から2147483647までの整数を表すことができる。
- 整数演算を行う際には、最低一つの数値には、'L'をつけて、longであることを明示すること。

書式

long var = val;

var　long型の変数名
val　その変数に代入する値

char

- charは、1文字を格納するために1バイト分のメモリを占めるデータ型である。文字定数は、'A'のように一重引用符によって囲む。複数の文字定数を表す文字列は、"ABC"のように、二重引用符で囲む。
- 文字は数値としても格納されている。ASCII符号集合をみると文字と数値との対応がわかる。このことにより、文字に対して数値演算を行うことが可能となる。このとき、ASCII符号集合で割り当てられた数値が利用される。例えば、'A'は65なので、'A'+1は66となる。詳細は、Serial.printlnのリファレンスを参照のこと。
- charは符号付の型であり、-128から127の間の値をとる。符号なしの1バイト(8ビット)のデータ型には、byteを使うこと。

書式

```
char myChar = 'A‘;
char myChar = 65;      // どちらも同じ
```

byte

- byteは、1バイト分のメモリを占めるデータ型であり、0から255までの整数を格納する。

書式

byte b = B10010;

"B" は2進数を示す接頭語 (B10010 は 10進数では18)

整数定数

- 整数定数はスケッチ内で直接記述される、123のような整数値を表す。デフォルトでは、これらの数値はint型の数値として扱われる。UやLという接尾語をつけることでこの振る舞いを変えることもできる。
- 通常、整数定数は10進数として扱われる。他の数字を底とする定数には特別な記法を用いる。

整数定数の例

底	例	接頭語	注釈
10	123	なし	0から9の数字が有効。0以外の数値で始める。
2	B1111011	B	8ビット以内の値(0から255)を表現可能。0と1だけが有効。
8	0173	0	0から7の数字が有効。
16	0x7B	0x	0から9、AからF、aからfが有効。

10進数

- 接頭語のない数値は10進数とみなされる。

101　　// 10進数の101　((1 * 10^2) + (0 * 10^1) + 1)

2進数

- 2進数では0と1だけが利用可能である。先頭にBをつける。

B101　// 10進数の5　((1 * 2^2) + (0 * 2^1) + 1)

- 2進表記では1バイト(8ビット)だけを表すことができる。よって実際に表される数は0(B0)と255(B11111111)の間である。int型の数値が必要なときは、以下のように2段階で生成することができる。

myInt = (B11001100 * 256) + B10101010;
// B11001100 は上位バイト

8進数

- 0から7までの数値が有効である。8進数は先頭に0をつける。

0101　// 10進数の65　((1 * 8^2) + (0 * 8^1) + 1)

- (意図せず)定数を0で始めると、コンパイラは8進数と解釈するので、見つけるのが困難なバグを埋め込む可能性がある。

16進数

- 0から9とAからFまでが有効である。Aは(10進数で)10、Bは11、Fは15である。16進数は0xという接頭語をつける。AからFは、小文字のaからfでもよい。

0x101　// 10進数の257　((1 * 16^2) + (0 * 16^1) + 1)

接尾語のUとL

- デフォルトでは整数定数はint型として取り扱われ、必然的にとりうる値に制限がある。整数定数を他のデータ型として取り扱うには以下のようにする。
- 接尾語のuもしくはUをつけると、unsigned型の数値になる。例：33u。
- 接尾語のlもしくはLをつけると、long型の数値になる。

  例： 100000L
- 接尾語tのulもしくはULをつけると、unsingned long型の数値になる。例： 32767ul

unsigned

- 正の数のみを取り扱うことができるようになる
- Int、long、char等の前につける

unsigned int

- Arduino Unoと他のATmegaベースのボードでは、unsigned intはintと同様2バイトの値を格納することができる。負数は格納せず非負数だけを格納する。よって、0から65535((2^16)-1)の範囲の整数を表す。
- Dueは4バイト(32ビット)の値を格納することができる。よって、0から4294967295((2^32)-1)の範囲の整数を表す。
- unsigned intとintとの違いは、符号ビットとも呼ばれる最上位ビットの違いである。Arduinoのintでは、最上位ビットが1のとき、その数値は負数となり、残りの15ビットが2の補数として解釈される。

- 変数が最大値を超えると、桁あふれにより最小値となる。これはどちらの方向(最大値から最小値へ、最小値から最大値へ)にも起こりうる。

```
unsigned int x;
x = 0;
x = x - 1;      // xは65535になる。最小値から最大値への桁あふれ。
x = x + 1;      // xは0になる。最大値から最小値への桁あふれ。
```

unsigned long

- unsigned longは、数値を保持するための、大きさが拡張された型である。32ビット(4バイト)の大きさを持つ。標準のlongとは異なり、unsigned longは負数を保持しない。よって、0から4294967295までの整数を表すことができる。

```
unsigned long time;

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

void loop(){
  Serial.print("Time: ");
  time = millis();
  //プログラムが開始してからの時間を表示する
  Serial.println(time);
  // 大量のデータを送らないよう、1秒待つ
  delay(1000);
```

unsigned char

- unsigned charは、1バイト分のメモリを占めるデータ型であり、byteと同じである。
- unsigned charは、0から255の値をとる。
- Arduinoのプログラミングスタイルの一貫性を保つために、byte型の利用が好まれる。

変数の有効範囲

- Arduinoが使っているC言語の変数は有効範囲と呼ばれる属性を持っている。全ての変数が大域変数であるBASICなどの言語とは異なっている。
- 大域変数はプログラムの全ての関数から可視の変数である。局所変数は、その変数が宣言された関数内で可視の変数である。Arduino環境では、関数の外側で定義された変数が大域変数である。
- プログラムが大きく複雑になってくると、局所変数はその関数の中からしかアクセスできないことを保証するための有効な手段となる。このことは、ある関数が他の関数で利用している変数を誤って変更することを防ぐ。
- for文の中での変数宣言も使いやすい。これは、for文のブロック内からだけアクセス可能な変数を作る。

```
int gPWMval;  // すべての関数からこの変数は見える

void setup(){
  // ...
}
void loop(){
  int i;    // i は loop() の中でだけ有効
  float f;  // f は loop() の中でだけ有効
  // ...
  for (int j = 0; j <100; j++){
  // j は forループの{}の中でだけ有効
  }
}
```

static

- 関数の中で宣言される変数を、staticを使って宣言すると、自動変数と同様に、その関数の中からだけ参照可能な変数を作成する。staticを使って宣言された変数は、自動変数とは異なり、静的記憶域期間を持つ。すなわち、関数が呼ばれるたびに生成され、関数が終了するときに削除される自動変数とは異なり、static変数は、関数が終了した後も、その値は保持され続ける。
- staticと宣言された変数は、プログラム開始処理前に一度だけ初期化される。
- 例に関しては別紙参照

volatile

- volatileは型修飾子の一つであり、変数のデータ型の前に書き、コンパイラやコンパイラが生成するプログラムによるその変数の取り扱い方法を変更する。
- 変数をvolatileとして修飾することは、コンパイラに対して指令することである。コンパイラとは、C/C++言語のコードを機械語に翻訳し、ArduinoのAtmegaチップ上の実際の命令セットを生成するものである。
- 具体的には、プログラムの変数が一時的に配置されて操作されるレジスタではなく、必ずRAMから読み込むように指示する。特定の状況では、レジスタに格納された変数の値は不正確になることがある。
- 並行処理するスレッドなど、変数が、そのコード以外によって変更されるような場合に、volatile修飾を行う必要がある。Arduinoでは、このようなことが起こりうるのは、割り込みサービスルーチンと呼ばれる、割り込みに関するコードだけである。
- 例は別紙参照

const

- constキーワードは定数を意味する。constは、変数を読み取り専用にするための修飾型である。これは、その変数はその型を持つ他の変数と同じように利用できるが、値は変更不可となることを意味する。const修飾された変数に値を代入しようとすると、コンパイラがエラーを出力する。
- const修飾された変数によって定義された定数も、他の変数と同様の有効範囲規則に従う。このことが、#defineを使うことの欠点であり、constを使って定数を定義する方法がすぐれている理由でもある。constは#defineよりも好まれる。

```
const float pi = 3.14;
float x;



x = pi * 2;    // 数学の定数にconstを使うのはいい。

pi = 7;        // 不正。定数を変更することはできない。
```

#define ? const ?

- 数値や文字列の定数は、constを使っても#defineを使っても作成可能である。配列にはconstを使う必要がある。一般的にはconstのほうが#defineよりも好まれる。

- 配列の大きさを宣言する際にも#defineを使うことができる。

#define

- #defineは、プログラムがコンパイルされる前に定数に名前を付ける通常のC言語の有用な構成要素である。定義された名前はArduinoチップのメモリ領域を利用することはない。コンパイル時にコンパイラがこれらの名前を定義された値に置き換える。
- #defineには、望ましくない副作用も存在する。例えば、定義された名前が他の定義名や変数の名前に含まれる場合がある。このときは、その文字列が定義された数字や文字に置き換えられる。
- 一般に、定数を定義する際には、#defineを使うのではなく、const修飾型を使うのが好まれている。
- ArduinoのdefineはC言語のdefineと同じ文法が適用される。

#define ledPin 3

// コンパイラがコンパイル時に、ledPin という文字列を、3に置き換える。

（注意）セミコロン（;）もイコール(=)もつけない

演算

- 変数と演算を組み合わせるとセンサーのアナログ値を入力して計算することができる
- 制御には変数と演算がよく使われる

演算の例（比較）

- １ == ２　（1と2が同じ）
- １ != ２　（1と2は違う）
- １ <= ２　（1は2以下）
- １ >= ２　（1は2以上）
- １ < ２　（1は2未満）
- １ > ２　（1は2より大きい）

演算の例（処理内容）

- 1 + 1　（加）　+=
- 1 - 1　（減）　-=
- 1 * 1　（乗）　*=
- 1 / 1　（商）　/=
- 1 % 1　(余り)　%=
- 1 = 1　(代入)

例

int LED = 0;（LEDは0）
LED++;　（LEDを1増加）
LED--;　（LEDを1減少）
++やー ーはその行を通過時に反映される。

例　（計算）

- int a = 10;　（10をaに代入する）
- int b = 20;　（20をbに代入する）
- int c = 30;　（30をcに代入する）
- int answer;　（answerという変数を作る）

- answer = a + b + c;
- （answerはaとbとcを足したもの）
- （answer = 60）

例　（数字を増加）

```
int sum = 0;

void setup(){
}
void loop(){
sum++;
}
```

演算の例（論理）

- && (論理積)
- || (論理和)
- ! (否定)

&& (論理積)

- 2つの値がどちらもtrueのときtrueとなる。
- 2つの入力ピンがどちらもHIGHのとき実行されます。

例

```
if (digitalRead(2) == HIGH  && digitalRead(3) == HIGH) {
  // 実行されるコード
}
```

|| (論理和)

2つの値のどちらか一方でもtrueならばtrueとなる。

xかyのどちらか一方でも0より大きければ実行されます。
例
```
if (x > 0 || y > 0) {
  // 実行されるコード
}
```

! (否定)

• 値がfalseならばtrueを、trueならばfalseを返す

例

xがfalseのとき実行されます。

```
if (!x) {
  // 実行されるコード
}
```

x＋＋と＋＋xの違い

- x++; // xを返し、1を加えます
- ++x; // 1を加えたxを返します

- x--; // xを返し、1を引きます
- --x; // 1を引いたxを返します

例

```
x ＝ 2;
y ＝ ++x;      // 結果：xは3、yも3
y ＝ x--;      // 結果：xは2に戻り、yは3のまま
```

以上です

- このほかにも演算子や変数の定義に使われるデータ型は存在する。色々調べて実際に使ってみよう。
- 基本的に数値は変数に格納して利用される。

- 次回はディジタル入力と出力を扱います

課題

- 複数変数を用意して四則演算を行いシリアルモニタに表示する。（画面表示を行うためのプログラムは課題用フォーマットに記載しておくので確認すること）

青文字の演算を行うこと

演算の例（処理内容）

- 1 + 1　（加）　+=
- 1 - 1　（減）　-=
- 1 * 1　（乗）　*=
- 1 / 1　（商）　/=
- 1 % 1　（余り）　%=
- 1 = 1　(代入)