ソフトウェア作成書

からくり工房 I.Sys 高村紀之

内容

0 章

余裕がない時はここを見る

・モノを追加せずポートを変えてない場合

BIT のプログラムが使える。今すぐコピペしろ

・モノを追加してポートを変えてない場合

BIT のプログラムに追加したポートを増やせ

・ポートを大幅に変えた場合

アキラメロン

1 章

C プログラミング

ここでは C 言語の構造化プログラムに必要な文法を書く

1-1 データ型 （配列　構造体）

（復習）データ型は次の種類がある

表 1

型名	表現	範囲	主な用途
Bool	1bit 符号なし整数	0,1	真理値
Char	8bit 符号付整数	-127~127	文字、データの節約
Short	16bit 符号付整数	-2^8~2^8-1	データの節約
Int	32bit 符号付整数	-2^{31}~$2^{31}-1$	普通の範囲での計算
Long	64bit 符号付整数	-2^{63}~$2^{63}-1$	大規模な計算
Float	32bit 浮動小数	~3.4×10^{38}	普通の範囲での計算
double	64bit 浮動小数	~3.4×10^{308}	大規模な計算

配列

行列やベクトルを扱いたいとき、同じデータをひとまとめにしたいとき、どのようにすればよいのでしょうか？

この章では、配列と呼ばれるデータ型を用いることで、多くの作業を反復に置き換えることや、データのやり取りをポインタと併用することで、高速化することができます。

１配列とは？

(2 次元)配列とはデータが n 行 m 列に並んだデータの集まりのことです。

例えば

```
int main_sensor[10][10];
```

のように変数名の後に[n][m]を付けることで、同じデータ型が数字の数だけ生成されます。

ベクトルのように扱いたい場合は、

```
int main_sensor[10];
```

のように[]を一つだけ設けるとよいです。

配列の使い方
変数に配列を代入したいとき

```
main = main_sensor[i];
```

のように代入したい配列の番号を配列の後に[i]と書きます
注意　C 言語の配列は 0 から始まり n-1 番目でおわります。C 言語は特に配列の要素を超えて代入ができるので、間違ったデータが入ることや、データの書き換えが起きる可能性があります。
使い方
配列に配列を代入したいとき
C 言語では for 文で繰り返し代入することで、配列の中身をほかの配列に代入することができます。

```
for (i = 0; i < 10; i++) {

main_sensor1[i] = main_sensor[i];

}
```

配列の威力
配列を用いることで、次のような処理が for 文で書き換えることができます。

```
main_sensor_0 = on_temp_sensor_0- off_temp_sensor_0;

. . .

main_sensor_9 = on_temp_sensor_9- off_temp_sensor_9;
```

が、

```
for (i = 0; i < 10; i++) {
```

```
main_sensor1[i] = on_temp_sensor[i]- off_temp_sensor[i];

}
```

のように 10 行が 3 行に省くことができ、見やすくすることができます。

構造体
この章では複数のデータ型をまとめ、一つのデータ型として扱うことのできる構造体の説明を行います。配列とは違い型として構造体はふるまうので、関数の引数に使うことができます。
定義方法

```
struct {

        float x;

        float y;

        float z;

}_vector;
```

のように先頭に struct をいれ、{}で入れたいデータ型を入れます。末尾に新しいデータ型の名前を入れれば完成です。
注意
　のちに説明するヘッダファイルに構造体の定義を行わないで下さい
使い方
宣言方法
構造体を定義した後に定義を行ってください。

```
_vector gyro;
```

もちろん構造体の配列も定義できます。

```
_vector gyro[10];
```

データ型から構造体への代入

```
gyro.x=vx;
```

配列と同じようにデータの要素を指定して代入してください。
データの要素を指定する方法は　変数名.メンバ　です。
構造体からデータ型への代入

```
vx= gyro.x;
```

もちろん逆もできます。

1-2 応用文法　(define　ポインタ)
Define 文　(マクロ)
この章では定数を変数として置き換える define 文の扱い方を見ていきましょう
宣言

```
#define LED1 PORTC.PODR.BIT.B0
```

他の文とは違い先頭に#がついていることや空白で単語を区切っています。
構造は#define <書き換える変数> <元の変数>　です。
ルール　define で定義した変数は（混乱を避けるため）すべて大文字で書きましょう
注意　define 関数はここでは説明をしないが、計算式を()でくくる等特に注意して扱うこと
define の威力
複数の場所に同一の定数を扱うときに効果を発揮します。

ポインタ
今までは関数に変数を引数としていましたが、変数の場所を引数とすることで関数に配列を扱うことができるようになります。
関数の引数と返り値にポインタ（配列）を使う

```
Float* sensor(float hoge* )
```

また、

```
Float* sensor(float hoge[] )
```

でも可能です(配列が 2 次元の場合は[][],or **)

1-3 違うファイルに入っている変数、関数の使い方
この章では、#include を用いて、ほかのファイルから、変数や関数を使う方法を見てみましょう。

ソースファイルとヘッダファイル
C 言語には命令を表すソースファイル(.c ファイル)と変数や関数を外部に渡すためのヘッダファイル(.h ファイル)があります。
ヘッダファイルの書き方は省略します。
ヘッダファイルは、ソースファイルに書かれた変数や関数を書くことができます。
また、include でヘッダファイルを読み込ませることで、ソースファイルにほかのソースファイルに書かれている変数や関数が使えるようになります。
関数の場合だと、

```
void get_linesen(void);
```

変数の場合だと

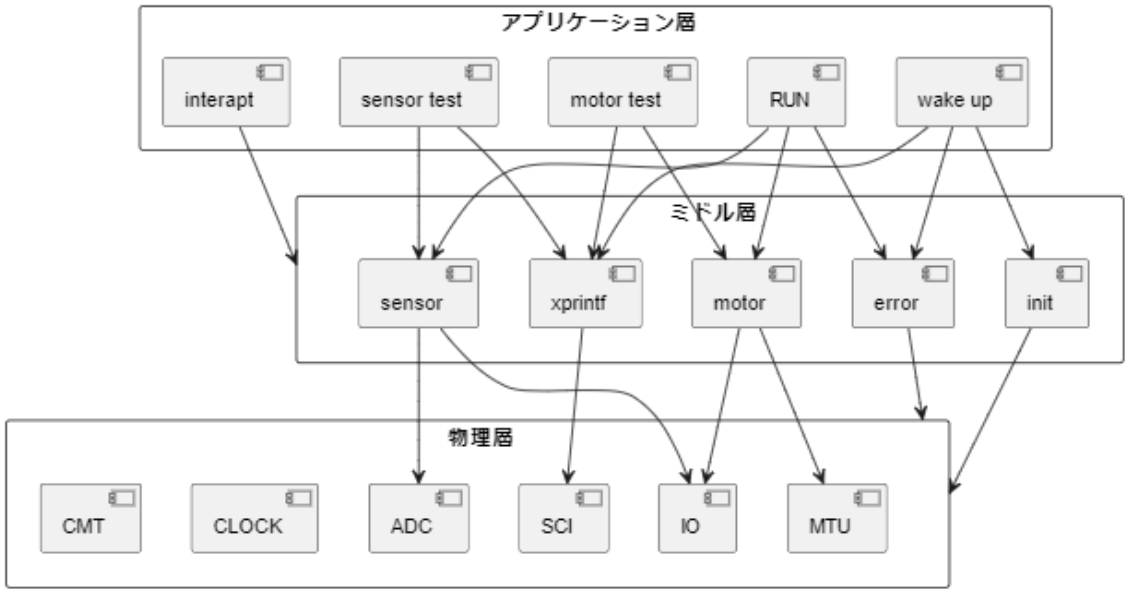
```
extern float p,i,d;
```

のように書きます。
#include <hoge.h>
ヘッダファイルを読み込む場合は、
ソースファイルの先頭に
#include <hoge.h>と入力します。

ライントレーサーの構造化プログラミング
2 部ではわかりやすいプログラムを書くために構造化プログラミングという手法を用いてライントレーサーのプログラムを書いていきます。詳しくは「組み込みソフトウェア開発のための構造化プログラミング」を参照してください。
2-1 構造化プログラミングって何？
構造化プログラミングとは主に次の特徴をもつプログラムを開発する手法である。
1 単一または少数の目的を持った関数群
2 関数またはファイル同士をグラフ構造で表すことができる
3 受け渡すデータ構造の明確化
構造化プログラミングのメリットとして
1 どこに何が書かれているかが明確（可読性の向上）
2 プログラム以外の形式で説明が容易
3 単体でのテストが可能(モジュール性)
4 プログラムの再利用が容易
があげられる。

2-2 全体の構造

プログラムはアプリケーション層、ミドル層、物理層の 3 つに分けることができる。
以下にコンポーネント同士のつながりを書く

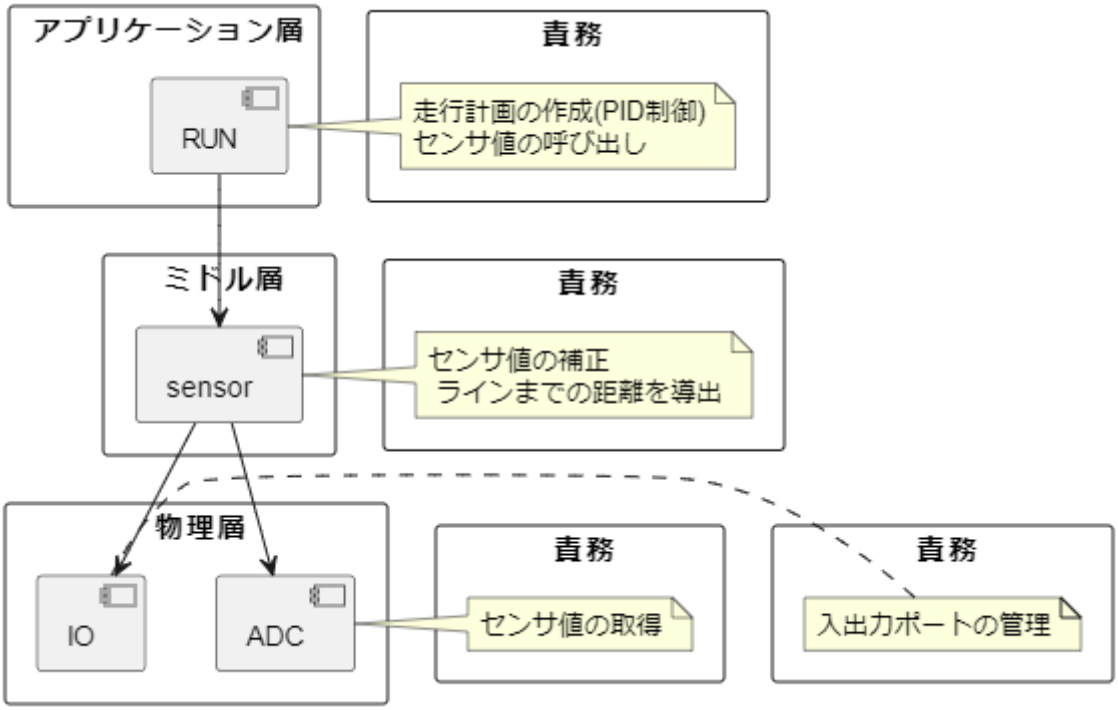


Interrupt, init は それぞれミドル層と物理層全体にかかっている

以降の章では 2-2 だけだとわからない、想像できないのでより RUN を例に具体的に説明していく

2-3 入力の構造 （センサ入力→ラインまでの距離→PID 制御）

この章では RUN のセンサ値の呼び出しから PID 制御入力までの導出を見ていく
アプリケーション層の RUN は 4 部で書く通り、センサからの入力に従い走行計画を作成し実行に移す。
その際の各コンポーネントの構造は次の図で示すことができる



ここで、責務とは各コンポーネントの内部処理やほかのコンポーネントに提供、要求するインターフェース(構造体)を表す。

見方

RUN の責務を見ると、「走行計画の作成、センサ値の呼び出し」が書かれている。

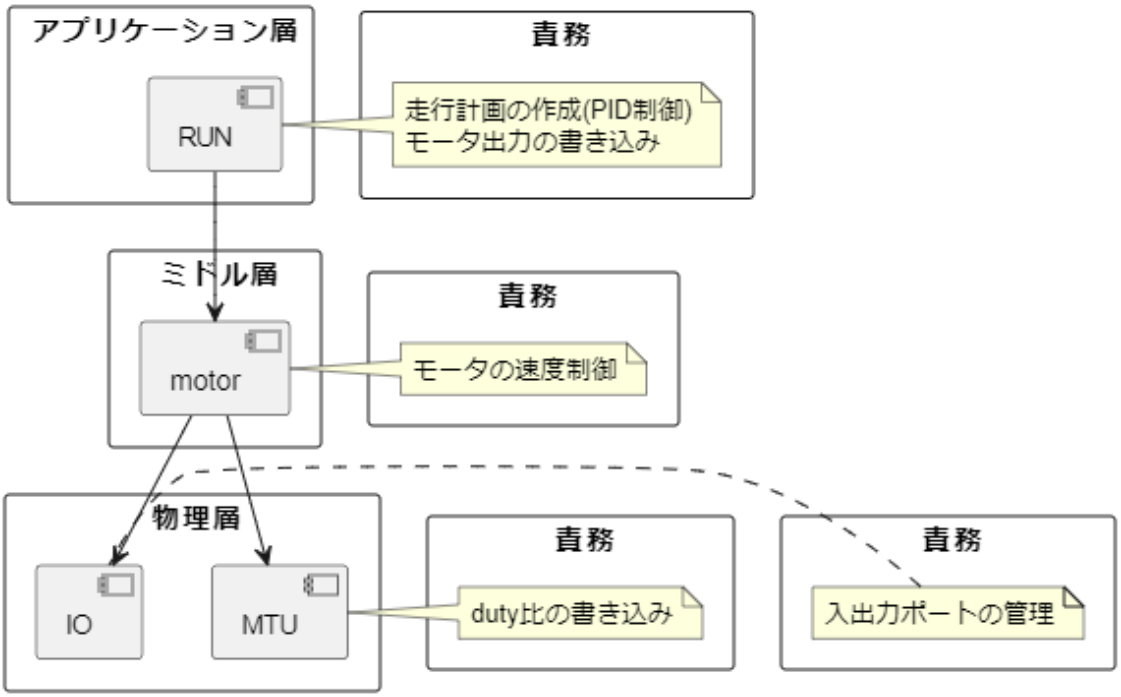
これは、RUN コンポーネントは走行計画の作成、センサ値の呼び出しを内部処理として行うことができるということである。

RUN から sensor への矢印が書かれている。

これは、RUN は sensor コンポーネントを呼び出すことができることを示している。

センサ層、物理層の責務や矢印も同様である。

2-4 演習：出力の構造　(PID 制御→回転速度→モータ出力)



2-3 と同様に RUN の責務、motor の責務をしらべ、それぞれの関係を調べる

コンポーネントを C 言語で書く

以降はコンポーネントを C 言語で書くために必要な知識を紹介します。

1 部の知識を用いますので振り返りつつ読んでいってください

3-1 コンポーネントの構成

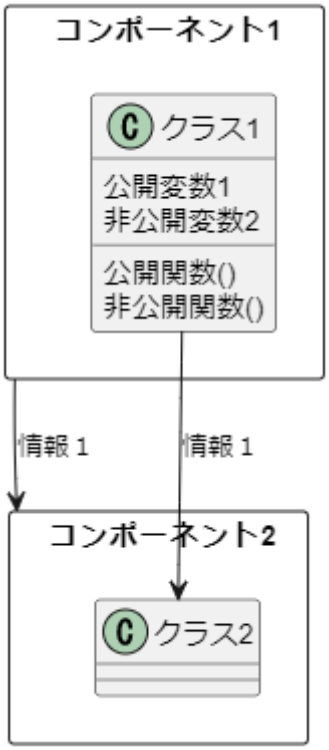
コンポーネントを具体化するにあたって重要な、機能やそれを実現する関数、変数をしまうための箱であるクラスを解説する

クラス

クラスとは、コンポーネントをプログラムで実装できるように具体化したものである。

クラスは変数と関数からなり、以下の図のようになる

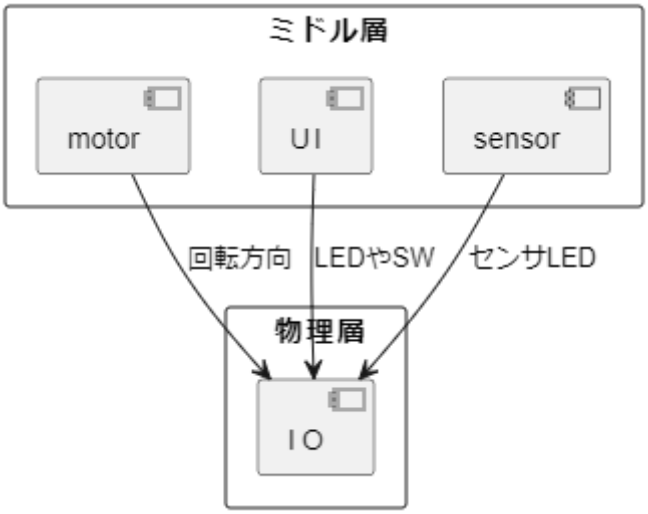
コンポーネントとクラスの関係は次の通り



・コンポーネントの中に一つだけクラスが存在する場合、コンポーネント間をやり取りするインターフェースはクラス間でやりとりする構造体に等しい。
・コンポーネントの中にクラスが複数存在する場合、コンポーネント間をやり取りするインターフェースは同じコンポーネント同士でやり取りするすべてのクラスの構造体と等しい。

C 言語ではヘッダファイルとソースファイルで構成されており、ヘッダファイルを読み込むことで、ヘッダファイルで宣言されている関数や変数が使えるようになる
そこで、クラスの構成は次になる。
ヘッダファイル hoge.h
・使用する　#define 文
・公開変数 extern
・公開関数 extern
ソースファイル hoge.c
・利用するファイルのヘッダ
・自ファイルのヘッダ
・公開変数
・非公開変数
・公開関数
・非公開関数

コンポーネントの書き方は次の通り
1 実現したい機能を書く
2 どのコンポーネントとインターフェースをやり取りするかを決める
3 機能を実現するための関数群(クラス)を決める
4 クラスを作成する
クラスの書き方は次の通り
1 インターフェースを決める
2 公開変数、公開関数を決める
3 実現したい機能を構成するための関数(非公開関数)を決める
3-2 インターフェース(構造体)の作成
インターフェースは、必要な情報だけをやり取りできるようにする
例えば IO に注目するとコンポーネント間のやり取りは次の通り



IO とセンサにはセンサ LED の on off が
IO とモータには回転方向を決めるポートの on off が
IO と UI には LED やスイッチの on off が
それぞれやり取りすることがわかる。
したがって、IO のインターフェースは
・LED の on,off
・センサ用 LED の on,off
・モータの正転、逆転
・スイッチの on,off
で構成される必要がある。
したがって IO のインター―フェースである構造体は

```
typedef struct {
```

```
bool led[5]; /* led ポート*/

bool sen_led; /* メインセンサー用 IO ポート*/

bool goal_led; /* ゴールセンサ−用 IO ポート*/

bool sw[2]; /* スイッチ用 IO ポート 0:BLUE 1:GREEN*/

bool motor_io[2];/* モーター用 IO ポート 0:A 1:B*/

}io;
```

と決定した。

3-3 関数の作成

インターフェースが定まったところで、クラスに含まれる関数を考える

コンポーネントに実装したい機能は、IO の操作と初期設定なので、

クラスの構成は次の通りになる

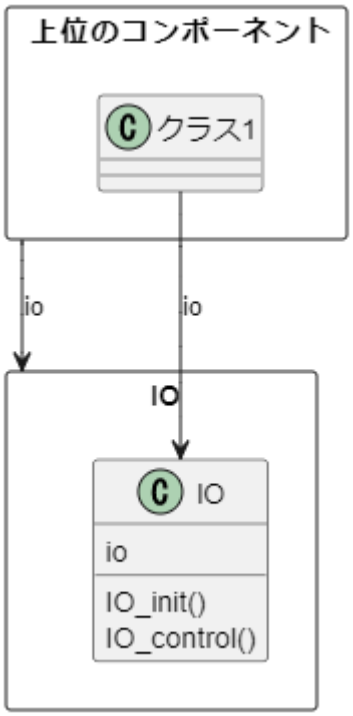
公開変数:io

非公開変数:なし

公開関数: 初期設定,IO の操作

非公開関数:なし

となる。



初期設定の関数や、IO の操作は次の部を参照してください。

各関数の実装とテスト

Layer1 物理層

埋め込み文書にテンプレートを添付する

図 1 .C ファイルのテンプレート

図 2 .h ファイルのテンプレート

4-1 IO

4-2 CLOCK

4-3 SCI

4-4 ADC

4-5 MTU

Layer2 ミドル層

4-6 SENSOR

4-7 MOTOR

4-8 UI

4-8 INIT

4-9 ERROR

Layer3 アプリケーション層

4-10 WAKE_UP

4-11 RUN

4-12 LOG