第1章 処理手順を章立てして見出しを付けることで順番に読めるプログラムに修正する

論理的文章でリファクタリング② …UARTの割り込みプログラム

村井 和夫

リスト1　UARTの受信割り込みのプログラム（リファクタリング前）

```
/**** 変数定義 ****/
unsigned char State, Flag;
unsigned char RecBuf[64];
unsigned char SndBuf[64];
int Index, i, j;

/*
 * UART送受信サブ関数
 */
void interrupt isr(void) {
  unsinged char data;
  if (PIRbits.RCIF) { //
    PIRbits.RCIF 0;    // 割り込み受信フラグ・クリア
    // エラーが発生した場合
    if ((RCSTAbit.OERR) || (RCSTAbits.FERR)) {
      RCSTA = 0;        // UART無効化
      RCSTA = 0x90;     // UART再有効化
    } else {            // 正常受信の場合
      if ((State > 1) && (Index < 64)) {
        RcvBuf[Index++] = RCREG; // データ取り出し
        if (Index >= 64)       // 64バイト受信完了か？
          Flag = 1;            // 受信完了フラグ・オン
      } else            // コマンド実行中でない場合
        data = RCREG; // 読み飛ばして無視
      }
    }
  }
```

リファクタリング対象は割り込み受信プログラム

● リファクタリング前のプログラム

IoTの通信処理プログラムの実例でリファクタリングの仕方を説明します．基本は文章と同じように章，段落立てして見出しを付けることで順番に読んでいくことが可能なプログラムに直していきます．

リスト1は一般的なUARTの受信割り込みプログラムです．UARTのレジスタ（**表1**）は，個別に#defineで定義しているとします．コンパイラ独自の機能として，この関数が割り込み処理に使えるように，interruptを関数名の前に付けています．

このプログラムにはさまざまなリファクタリング要素が詰まっています．まず割り込み処理手順を記述した技術文書となっていないので，そこから修正していきます．

受信割り込みを確認して，受信エラーを確認，受信状態の確認をしながら，割り込み受信処理を進めていることは分かります．このためインデントが非常に深く複雑になり，このプログラムを一見して処理の流れがすぐ分かる人は，まずいないでしょう．

技1 構造を示すブロック・コメントの追加

処理手順が分かるように，Doxygenの構造化した

表1　UARTのレジスタ

レジスタ名	内　容
PIRbits	UARTの割り込み状態を表すレジスタのアドレスへのポインタの内容
RCSTAbit	UARTのステータス・レジスタのアドレスへのポインタの内容
RCREG	UARTの受信データ・レジスタのアドレスへのポインタの内容

文章の章立てとして，インデント・レベルを処理手順の見出し・段落に対応させて，処理単位となるブロックの先頭に見出しとなるブロック・コメントを埋め込みます．元の行コメント（//）と区別できるように，新しいコメントは，旧スタイルのコメント（/**　**/）で入れています（**リスト2**）．

このようなコメントを付けても，この処理手順を理解するのは困難で，元のプログラムがいかに不適切かよく分かります．

● 手順1．プログラムの構造上の問題点を把握する

新しいブロック・コメントだけを抜き出します（**リスト3**）．これによってプログラムの構造上の問題点がはっきりします．文章として見れば，1章だけの文章で，章，節，段落が，4段にもなってどんどん深くなっています．

本来，同一レベルで行われるべき処理を，条件を付