第2特集 一歩踏み込む基本姿勢と基礎知識

「組み込みソフト開発」で身を立てる！

第2章 「組み込み」ならではの基礎知識

スタートアップ・ルーチンからハードウェアまで

セサミアン3人組

組み込みソフトウェアには，一般のアプリケーション・ソフトウェアにはない考えかたが存在する．ここでは，「組み込み」ならではのいくつかの概念について解説する．例えば，プログラムのランタイム構造やスタートアップ・ルーチン，割り込み，配列の実体，同期/非同期，volatile，ハードウェアなどを説明する．　（編集部）

ここでは，組み込みソフトウェア開発者をめざす新人の皆さんに理解していただきたい考えかたについて解説します．ここで解説する内容は，一般的なC言語やソフトウェア開発の解説書にはあまり説明されていないものです（もしかすると，皆さんの周りの先輩もちゃんと説明できないかもしれない）．しかし，組み込みソフトウェア開発ではきわめて重要な考えかたです．

また，ハードウェアに関する説明もありますが，「わたしはソフトウェア技術者だから関係ないや」などとは決して考えないでください．この世界で長く技術者としてしごとをしていくためには，ソフトウェア技術者だからとか，ハードウェア技術者だからという枠を持つことは厳禁です．

本稿で説明する知識は，大きく次の二つに分けられます．

1）プログラムを起動させるためのしくみと，起動したプログラムが動くための基本メカニズム

ランタイム構造，スタートアップ・ルーチン，配列の実体，volatile，非同期呼び出し

2）プログラムがマイコン（システム）と外部との間でデータをやり取りするための基礎知識

ポーリングと割り込み，エッジ・トリガとレベル・センス，メモリとポート

C言語に関する本の内容は，C言語の文法や使いかたについての解説がほとんどです．WindowsやLinux/UNIX上でプログラミングするだけならそれで十分でしょう．例えば，LinuxやUNIX上ならば，おまじないのように，

```
% cc example.c
```

とコマンド入力すれば，ファイルexample.cに書かれたC言語のプログラムがコンパイルされ，a.outという名まえの実行可能なファイルが生成されます．そして，

```
% ./a.out
```

とコマンド入力すれば，ファイルexample.cに書いておいたプログラムが（あわよくば意図したとおりに）実行されます．この手順は，使用するOS環境やコンパイラなどによって若干変わりますが，WindowsやLinux/UNIXの環境ならば，基本的にはmain関数以降のプログラムを書き，そのファイルをコンパイルして実行するだけです．プログラムを動かすために，プログラムの起動や動作時のしくみなどを気にする必要はありません．

ところが，組み込みシステムの場合はそうはいきません．つまり，魔法のようなccコマンドはないのです．すでに実機でプログラムを動かした経験があって，「普通にコンパイルしてプログラムが動いたよ」という方もおられるかもしれません．でもよく思い出してください．先輩から「コンパイル環境」をもらって，コンパイル対象のファイル名を書き換えただけではないですか？ そのコンパイル環境に，ccコマンドのような秘密が隠されているのです．その秘密を理解しないことには，いつまでたっても一人前の技術者にはなれません．

また，さらに経験を積むうちに，「プログラムの動作が

おかしいのだが，実行するたびに動作が異なっているため，どうしてよいかわからない」とか，「ソース・レベルのデバッグができない」といった場面に出くわすことでしょう．こうしたときのために，プログラムのランタイム構造を理解しておくことはきわめて重要です．

また，WindowsやLinux/UNIX上でプログラミングする場合は，標準入出力を使ってデータを取り込んだり，デバッグ・ライトと呼ぶデバッグのための情報を出力したりできます．ところが，組み込みシステムの実機環境では，この「簡単なこと」がたいへん難しいのです．また，実機ではファイル・システムを搭載していないことがほとんどです．標準入出力やファイルといったデータの入出力を使わずに，いったいどうしているのでしょうか？

組み込み機器は，例えば「ユーザがボタンを押した」といったON/OFFの情報や，光や音などのアナログ的な情報を取り込みながら動作しています．それらのデータは通常マイコンの周辺機能を使って取り込まれ，プログラムはシリアル・インターフェースやポート，あるいは割り込み信号を介してデータを受け取ります．このようなデータの取り込みは，ある程度ハードウェアの知識を持っていると，ぐっと理解しやすくなります．

1 プログラムはどのように動くのか

プログラムがどういったしくみで動いているかということ（プログラムのランタイム構造）は，一般のC言語の解説書には書かれていません．それは，WindowsやLinux/UNIXといったOS上で動くプログラムを開発する場合，ランタイム構造はOS側で決まっており，C言語を使ってプログラミングする人はそれを気にしなくてもいいからです．

でも，組み込みソフトウェアの開発ではそうはいきません．それは，使用できるメモリ量に制限がある（つまりメモリ量が少ない）のと，デバッグ環境が必ずしもWindowsやLinux/UNIXのときのように整っているとは限らないからです．

実際には，こんなことを面と向かって説明してもらえる機会は少なく，ちょっと勉強すればもう少しサクサクとしごとができて楽だろうにという人が多いのが現実のようです（それに，プログラムがどう動いているかわからないなんて，くやしいではありませんか）．

ここでは，プログラムのランタイム構造の基本的なメカニズムについて説明します．

●例：再帰呼び出しのプログラム

プログラムのランタイム構造を説明するために，まずは自然数nの階乗計算を考えます．自然数nの階乗$n!$は，

$$n! = n \times (n-1) \times \cdots \times 2 \times 1 \qquad (1\text{-}1)$$

と計算します．

また式(1-1)において，$(n-1) \times \cdots \times 2 \times 1$とは$(n-1)$の階乗のことですから，

$$n! = n \times (n-1)! \qquad (1\text{-}2)$$

とも書けます．

さて，nの階乗を計算する関数factをコーディングしてみます．ここでは式(1-2)を使ってコーディングします．

```
int fact(int n)
{
    if (n == 1)
        return 1;
    return(fact(n-1) * n);
}
```

関数factの中で自分自身を呼び出していますね．これを再帰呼び出しと言います[注1-1]．

例えば，fact(3)の計算(3!)は次のように行われます．

fact(3)： 3==1 ではないのでfact(2)*3を返す
そのためにfact(2)を計算する
fact(2)： 2==1 ではないのでfact(1)*2を返す
そのためにfact(1)を計算する
fact(1)： 1==1 なので1を返す

注1-1：再帰呼び出しを使うとプログラムをエレガントに書けるが，組み込みソフトウェアでは通常は使わない（関数factの例なら，式(1-1)をwhile文を使って書く）．ただし，本記事ではあえて再帰呼び出しを例にしている．