■assert(アサート)禁止
●論理エラー検出手法
C言語およびC++言語のプログラミングテクニックの一つとしてassert()マクロの利用があります。
例えば、ポインタを引数とする関数を利用する場合、引数となるポインタがNULLではないということを「期待」しているような場合、以下のようになります。

```
void funcp ( void * p )
{
    assert( NULL != p );
    //ポインタ正常時処理
}
```

このように作られた関数に対して引数にNULLを与えて関数コールすると以下のようなエラーメッセージが出力されます。

```
atest: asserttest.c:22: funcp: Assertion `((void *)0) != p' failed.
```

assert()マクロはこのように論理的に期待している状態と異なる状態を検出してプログラムの実行を停止させることが出来るのです。
非常に便利な機能であることは確かで、デバッグ後には「-DNDEBUG」をCコンパイラの引数に指定するなどの方法で簡単にassert()が生成するコードを削除することができます。
ソースコード上にassert()を挿入して単体テストの効率を上げるというような発想の下に積極的な活用が薦められることも少なくありません。

●性善説のassert
先の例のようにポインタがNULLになるような条件というものは一過性のデバッグ時期にのみ発生する不具合と言い切ることが出来るでしょうか。
呼び出し側で値を保証すべきであり、呼び出された側ではそのようなチェックを行う必要は本当にないのでしょうか。
これは不具合のないプログラム実装が行われるべき、という考えが前提の「性善説」が基本になっているという言い方もできます。

仮想記憶システムではない環境では複数のタスクから共通のRAM領域へのアクセスが可能であることから、関係のない処理の想定外の不具合によって自関数のみが使用しているはずのポインタ変数がクリアされるような現象が発生しないとは言い切れません。
「いつの間にか」、「突然に」、「想定外に」NULLに変わってしまうというような現象が発生することになります。
assert()でチェックしている「デバッグ時期」には現象が発生せず、assert()を無効にした「リリース後」に発生する不具合かもしれません。

●assertとfail-safe
一方で、組込み機器は実行時に不正な状況に陥ってもシステムを停止させずに安定稼動させる必要があるという命題を持っています。
「デバッグ時にのみ存在する不正な状態」とは限定すべきではなく、「常に不正な状態を回避する実装」が要求されます。
いわゆるfail-safe(フェイルセーフ)の概念が必要で、不正な状況に陥っても単純にプログラムを停止させて「プログラマに知らせる」のではなく、安全な状態でシステムを落ち着かせる必要があります。

この意味においてはassertの概念と組込みシステム開発におけるfail-safeの概念は根本から食い違っていると言えます。

したがって、デバッグ時にのみ論理的に不正な状態を検出するassert()は組込みシステム開発では使用すべきではなく、常に論理的な不正を検出して対応を行うための機能をリリース版として実装する必要があるわけです。

最後に先ほどの関数のassert()を使わない場合のコードは以下のようになります。

```
void funcp ( void * p )
{
```

```
    if( isvalid(p) ) {
        //ポインタ正常時処理
    }
    else {
        //ポインタ不正時処理
    }
}
```