第1部

第5章 5分でできる簡単な事例から，C/C++固有の手法，構造に手を入れる大がかりな技まで

レッスン⑤…リファクタリングのパターンを学ぶ

中林 智之

本章ではリファクタリングを実践するための具体的なパターンを紹介します．

リファクタリングのパターンを知ることで，やみくもにコードを変えるのではなく適切な手法を選べるようになります．

各パターンでは次の内容を解説します．

1，リファクタリングを実施する理由
2，リファクタリング前後のサンプル・コード
3，リファクタリングの実施手順

各サンプル・コードは非常に単純化したものです．可能なものにはユニット・テストも書くようにしました．

サンプル・コードの1つ1つは取るに足らない改善に見えるかもしれません．しかし，このような小さなリファクタリングの積み重ねが大きな変更を安全に進める基盤になります．

5-1 今日から使える5分でできるパターン

気軽に始められるパターンから説明を始めます．これらのリファクタリング・パターンは5分もあれば実施できるものばかりです．まずはこれらのリファクタリングを躊躇せずに実施できるようになることを目指しましょう．これらのリファクタリング・パターンを適用することでコードが整理され，さらに高度なリファクタリングが可能となります．

技1 デッド・コードの削除

● 理由

デッド・コード（Dead Code）とはプログラムの実行に影響を与えない不要なコードのことです．

デッド・コードは次のような原因で発生します．

- 機能の削除：ある機能を削除した際に一部のコードが残ってしまう
- リファクタリングの過程：コードの整理中に不要になった関数や変数がそのまま放置される
- 条件分岐の変化：ある条件分岐が常に真または偽になり一部のコードが実行されなくなる

デッド・コードは，プログラムの可読性を下げます．コードを読む人は，このコードはなぜここに存在するのだろうか，消しても問題ないのだろうかという疑問が頭の中で渦巻くことになります．

コードが将来必要になるかもしれない，と心配する必要はありません．Gitなどのバージョン管理システムからいつでも復元できます．

● サンプル・コード

リファクタリング前のコード［**リスト1（a）**］には次の問題点があります．

リスト1 デッド・コードの削除

```
void unused_function(void)          ← 呼び出さない関数の定義
{
    printf("this function is not used.\n");
}

void used_function(void)
{
    printf("this function is used.\n");
    // unused_function();           ← 関数の呼び出しがコメント・アウトされている
}
```

（a）ビフォー

```
void used_function(void)
{
    printf("this function is used.\n");
}
```

（b）アフター