

第10章

期待される挙動のアサーション

デバッグの理論と実践 (P.233~263)

Andreas Zeller 著

発表者 吉富 楓雅

目次

- アサーションとは何か？
- 不変条件のアサーション
- システムにおけるアサーション
- 製品コードの確認

アサーションとは何か？



© Can Stock Photo

アサーションとは何か？

- プログラミング言語などの仕様・機能の 1 つ
 - * プログラムの前提として満たされるべき条件を記述
 - * 実行時にそれが満たされない場合はエラーや例外を発生させる

アサーションを行う上で必要なこと

- 基本的にはプログラムにチェックを行うコードを挿入
 - * しかしデバッグに関するコードはプログラムコードから分離すべき
 - ▶ 特定の条件を明示的にチェックする特別な関数を用意する
- 実際はアサーションを無効にできるようにしたい
- 診断情報の報告もしてほしい
 - * “failed”だけでは情報として不十分

C++プログラム用のassert()の典型的な定義の例

- #ifndef NDEBUG
- #define assert(ex)
- ((ex) ? 1 : (std::cerr << __FILE__ << ":" << __LINE__
- << ": assertion “ #ex “ failed¥n”,
- abort(), 0))
- #else
- #define assert(x) ((void) 0)
- #endif

不変条件の アサーション



photo: 423752

不変条件のアサーション

- アサーションの重要な役目は,データの不変条件を保証すること
 - * 不変条件：実行全体を通じて成立しなければならないもの
- 実用的には,メソッドの前後に不変条件を盛り込むと良い
- アサーションは不整合な状態にしないことを保証する
 - * アサーションに違反した場合はすぐに感染がわかる
- しかし冗長で読みにくいものはあまり良くない

C++のTimeクラス

- Class Time{
- public:
- int hour(); //0..23
- int minutes(); //0..59
- int seconds(); //0..60(閏年の60秒を含む)
- void set _ hour(int h);
-
- }

Timeクラスのアサーション

- void Time:: set _ hour(int h)
- {
- // 事前条件
- assert(0 <= hour() && hour() <= 23) &&
- (0 <= minutes() && minutes() <= 59) &&
- (0 <= seconds() && seconds() <= 60) ;
-
- // 事後条件
- assert(0 <= hour() && hour() <= 23) &&
- (0 <= minutes() && minutes() <= 59) &&
- (0 <= seconds() && seconds() <= 60) ;

Timeクラスのアサーション

(改)

- `bool Time:: sane()`
- `{`
- `return(0 <= hour() && hour() <= 23) &&`
- `(0 <= minutes() && minutes() <= 59) &&`
- `(0 <= seconds() && seconds() <= 60) ;`
- `}`
- `void Time:: set _ hour(int h)`
- `{`
- `assert(sane()); // 事前条件`
- `//実際の処理コードがここに入る`
- `assert(sane()); // 事後条件`
- `}`

アサーションが失敗した場合

- 欠陥についての仮説を絞り込むことができる
 - * 事前条件に違反していたら, `set _ hour()`の前で感染が発生したに違いない
 - * 事後条件に違反している場合は, `set _ hour()`内で感染が発生したに違いない
 - * 事後条件が満たされていれば, `set _ hour()`の中で感染が発生した可能性はない

システムにおける アサーション



保証されるべきもの

- プログラムの性質の中で,実行の間ずっと持続でされるべきもの
 - * プログラムが他のプロセスのデータを変更したり,アクセスしないこと
 - * 算術的な例外が見過ごされないこと
 - * 設定された権限をプログラムが逸脱しないこと

CやC++の例

- CやC++ではヒープメモリの利用上の誤りが一般的エラーの例
 - * オブジェクトが不要になったらヒープメモリを明示的に解放しなければならない
 - ▶ 解放されたメモリは他のオブジェクトに割り当て可能な状態になる
- プログラマは解放後のメモリ領域を使用しないようにしなければならない
 - * ミスによる障害は特定するのが難しい

製品コードの確認



チェックを無効にすべきではない例

- クリティカルな結果

- * プログラムの計算が,人の命,人の健康,お金に関わる場合

- ▶ 無効にできる点で重大な結果の確認にアサーションは適切でない

- 外部条件

- * 制御できない条件はどんなものでも整合性を確認すべき

- ▶ 全ての構文的,意味的な制約を満たすことを検証する必要があり,エラーがあった場合はユーザに知らせなければならない

その他検討すべき点

- 有効になっているアサーションが多いほど感染を捕捉できる可能性が高くなる
 - * アサーションは有効にすべき
- プログラムの失敗が早いほど欠陥の追跡が簡単
 - * 有効になっているアサーションが多いほど感染が早い段階でわかる

その他検討すべき点

- リリースされてしまった欠陥は追跡が最も困難
- デフォルトの設定では,アサーションが失敗することはユーザにとってわかりやすいものではない
- * プログラムを単にアボートするだけではユーザの助けにはならない

所感

[illegible]

所感

- アサーションについての研究はどのようなものがあるのか
- メモリの誤使用を検出するツールがあるらしい
 - * このツールについて掘り下げてみても面白いかも
- リリースされてしまった欠陥にアサーションはできないのか