

ド' ) ズーツ

バグベアード入門



Boost.勉強会 ( 2009-12-12 )

# バグベアード入門

# 概要

- バグベアードとは？
- Hello, Bugbeard!
- 既存のコードへの適用
- プロファイリング
- カバレッジ測定
- “悪魔の契約”
- ご利用に当たってのヒント

バグベアード入門

バグベアードとは？

# バグベアードとは？(1/5)

- Boost C++ Libraries とは関係ありません。
  - Boost 採用に向けて協力してくださる方がいれば検討します。
    - 私の英語力が頼りないので、主にコミュニケーションのサポートをして頂ければ...

# バグベアードとは？(2/5)

- printf デバッギングを簡単且つゴージャスに行う為のヘッダーファイル
  - サンプルおよびドキュメント関係やおまけのアイコンファイルを除けばバグベアードは bug.h だけで、他にはなにもありません
  - 若干のマクロによる指定とともにインクルードするだけ

# バグベアードとは？(3/5)

- バグベアードの機能
  - プログラム動作ログ出力
  - プロファイリング機能
  - カバレッジ測定機能

# バグベアードとは？(4/5)

- etc

- ライセンスフリー
- マルチプロセス/マルチスレッド対応
- 各種ログの出力先、出力形式を自在にカスタマイズ可能
- Windows と POSIX な環境に対応
  - その他の環境(カーネルモード含む)も要望があれば対応を検討します
- ANSI(MBCS)/Unicode に対応
- 日本語/英語を切り替え可能
- ソースコードレベルで組み込まれるのでユーザー環境でも特殊なツールをインストールしたりすることなくそのまま使用可能

# バグベアードとは？(5/5)

- URL

- <http://tricklib.com/cxx/ex/bugbeard/>

バグベアード入門

**HELLO, BUGBEARD!**

# Hello, Bugbeard! (1/19)

- まずは bug.h をダウンロード
  - <http://tricklib.com/cxx/ex/bugbeard/bug.h>

# Hello, Bugbeard! (2/19)

- ソースコードにおまじないを記述

```
#if !defined(NDEBUG)
#define BUG_EVIL_CONTRACT // “悪魔の契約”
#define BUG_DEFINE_GLOBAL_LOGGER ¥
    new bugbeard::bug_tree_logger(new bugbeard::bug_file_writer_base(stderr))
    // 標準エラーヘッリー形式の出力を行うロガーの定義
#else
#define BUG_DISABLE_BUGBEARD // バグベアードのOFF
#endif

#include “bug.h” // バグベアード本体の include
```

# Hello, Bugbeard! (3/19)

バグベアードを有効にする為のマクロ定義  
(※後半の“悪魔の契約”にて説明)

```
#if !defined(NDEBUG)
#define BUG_EVIL_CONTRACT // “悪魔の契約”
#define BUG_DEFINE_GLOBAL_LOGGER ¥
    new bugbeard::bug_tree_logger(new bugbeard::bug_file_writer_base(stderr))
    // 標準エラーヘッリー形式の出力を行うロガーの定義
#else
#define BUG_DISABLE_BUGBEARD // バグベアードのOFF
#endif

#include “bug.h” // バグベアード本体の include
```

デバッグ版

リリース版

# Hello, Bugbeard! (4/19)

プログラム動作ログの出力指定

```
#if !defined(NDEBUG)
#define BUG_EVIL_CONTRACT // “悪魔の契約”
#define BUG_DEFINE_GLOBAL_LOGGER ¥
    new bugbeard::bug_tree_logger(new bugbeard::bug_file_writer_base(stderr))
// 標準エラーヘッリー形式の出力を行うロガーの定義
#else
#define BUG_DISABLE_BUGBEARD // バグベアードのOFF
#endif

#include “bug.h” // バグベアード本体の include
```

デバッグ版

リリース版

# Hello, Bugbeard! (5/19)

ツリー形式のログ出力

```
#if !defined(NDEBUG)
#define BUG_EVIL_CONTRACT // “悪魔の契約”
#define BUG_DEFINE_GLOBAL_LOGGER ¥
new bugbeard::bug_tree_logger(new bugbeard::bug_file_writer_base(stderr))
// 標準エラーへツリー形式の出力を行うローガーの定義
#else
#define BUG_DISABLE_BUGBEARD // バグベア OFF
#endif

#include "bug.h" // バグベア の include
```

デバッグ版

リリース版

標準エラーへ出力

# Hello, Bugbeard! (6/19)

```
#if !defined(NDEBUG)
#define BUG_EVIL_CONTRACT // “悪魔の契約”
#define BUG_DEFINE_GLOBAL_LOGGER ¥
    new bugbeard::bug_tree_logger(new bugbeard::bug_file_writer_base(stderr))
    // 標準エラーヘッリー形式の出力を行うロガーの定義
#else
#define BUG_DISABLE_BUGBEARD // バグベアードのOFF
#endif
#include “bug.h” // バグベアード本体の include
```

デバッグ版

リリース版

バグベアードを無効にする為のマクロ定義

※ bug.h を include する前に BUG\_EVIL\_CONTRACT あるいは BUG\_DISABLE\_BUGBEARD のどちらかのマクロを定義

# Hello, Bugbeard! (7/19)

```
#if !defined(NDEBUG)
#define BUG_EVIL_CONTRACT // “悪魔の契約”
#define BUG_DEFINE_GLOBAL_LOGGER ¥
    new bugbeard::bug_tree_logger(new bugbeard::bug_file_writer_base(stderr))
    // 標準エラーヘッリー形式の出力を行うロガーの定義
#else
#define BUG_DISABLE_BUGBEARD // バグベアードのOFF
#endif

#include "bug.h" // バグベアード本体の include
```

バグベアード本体(ヘッダファイル)の include

# Hello, Bugbeard! (8/19)

- バグベアードを使った固定文字列出力

```
#include <stdio.h>
#include <stdlib.h>

~ここにおまじない~

int main(int argc, char * args[])
{
    for(int i = 0; i < argc; ++i)
    {
        printf("args[%d]: %s\n", i, args[i]);
    }

    BUG_puts("このロリコンどもめ!");

    return EXIT_SUCCESS;
}
```

# Hello, Bugbeard! (9/19)

```
#include <stdio.h>
#include <stdlib.h>
```

```
~ここに
```

```
int main
```

```
{
```

```
for
```

```
{
```

```
}
```

```
BUG_puts("このロリコンどもめ!");
```

```
return EXIT_SUCCESS;
```

```
}
```

日付&時刻 : 2009-12-11(金) 23:24:09.776

23:24:09.776 | - このロリコンどもめ! <..¥tutorial1.cpp>#21

日付&時刻 : 2009-12-11(金) 23:24:09.777

# Hello, Bugbeard! (10/19)

The image shows a debugger window with a pink background. The window displays the following text:

```
~ここに  
日付&時刻 : 2009-12-11(金) 23:24:09.776  
int mai 23:24:09.776 | このロリコンどもめ! <..¥tutorial1.cpp>#21  
{  
for {  
  {  
  }  
}  
BUG_puts("このロリコンどもめ!");  
}
```

Callouts from the debugger window:

- プログラムの開始日時 (Program start time) - points to the first timestamp.
- 該当行を実行した時刻 (Execution time of the current line) - points to the timestamp on the line containing the code.
- ソースファイル内の行番号 (Line number in source file) - points to the line number #21.
- プログラムの終了日時 (Program end time) - points to the second timestamp.
- ソースファイル名 (Source file name) - points to the file path <..¥tutorial1.cpp>.

# Hello, Bugbeard! (11/19)

- バグベアードを使った書式文字列出力

```
#include <stdio.h>
#include <stdlib.h>

~ここにおまじない~

int main(int argc, char * args[])
{
    for(int i = 0; i < argc; ++i)
    {
        printf("args[%d]: %s\n", i, args[i]);
    }

    BUG_puts(BUG_FORM("argc:%d", argc));

    return EXIT_SUCCESS;
}
```

# Hello, Bugbeard! (12/19)

```
#include <stdio.h>
#include <stdlib.h>
```

```
~ここに
```

```
int main
```

```
{
  for
```

```
{
```

```
    BUG_puts(BUG_FORM("argc:%d", argc));
```

```
    return EXIT_SUCCESS;
```

```
}
```

日付&時刻 : 2009-12-11(金) 23:53:50.666

23:53:50.671 | - argc:1 <..¥tutorial3.cpp>#21

日付&時刻 : 2009-12-11(金) 23:53:50.672

# Hello, Bugbeard! (13/19)

- バグベアードを使った変数出力

```
#include <stdio.h>
#include <stdlib.h>

~ここにおまじない~

int main(int argc, char * args[])
{
    for(int i = 0; i < argc; ++i)
    {
        printf("args[%d]: %s\n", i, args[i]);
    }

    BUG_puts(BUG_VAL(argc));
    BUG_puts(BUG_VAL(args));
    BUG_puts(BUG_VAL(args[0]));

    return EXIT_SUCCESS;
}
```

# Hello, Bugbeard! (14/19)

```
#include <stdio.h>
#include <stdlib.h>
```

```
~こ
```

```
int r
{
```

```
日付&時刻 : 2009-12-11 (金) 23:32:38.585
```

```
23:32:38.588 | • argc (0x0012FF5C) = 1 (0x00000001) <..¥tutorial2.cpp>#21
```

```
23:32:38.589 | • args (0x0012FF60) = HEX:D4215901 <..¥tutorial2.cpp>#22
```

```
23:32:38.590 | • args[0] (0x015921D4) = "...¥work¥tutorial2.exe" <..¥tutorial2.cpp>#23
```

```
日付&時刻 : 2009-12-11 (金) 23:32:38.592
```

```
BUG_puts (BUG_VAL (argc));
BUG_puts (BUG_VAL (args));
BUG_puts (BUG_VAL (args[0]));
```

```
return EXIT_SUCCESS;
```

```
}
```

# Hello, Bugbeard! (15/19)

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main(int argc, char* argv[])
{
    BUG_puts(BUG_VAL(argc));
    BUG_puts(BUG_VAL(argv[0]));
    return EXIT_SUCCESS;
}
```

変数名

変数の値(10進数)

変数の値(16進数)

変数の値(文字列)

変数の値(HEX)

変数の値が格納されているアドレス

```
日付&時刻 : 2009-12-11 (金) 23:32:38.585
23:32:38.588 | - argc (0x0012FF5C) = 1 (0x00000001) <..¥tutorial2. cpp>#21
23:32:38.589 | - argv (0x0012FF60) = HEX:D4215901 <..¥tutorial2. cpp>#22
23:32:38.590 | - argv[0] (0x015921D4) = "...¥work¥tutorial2. exe" <..¥tutorial2. cpp>#23
日付&時刻 : 2009-12-11 (金) 23:32:38.592
```

# Hello, Bugbeard! (16/19)

- ステートメントハック

```
#if !defined(NDEBUG)
#define BUG_EVIL_CONTRACT // “悪魔の契約”
#define BUG_DEFINE_GLOBAL_LOGGER ¥
    new bugbeard::bug_tree_logger(new bugbeard::bug_file_writer_base(stderr))
    // 標準エラーヘッリー形式の出力を行うロガーの定義
#define BUG_STATEMENT_HACK // ステートメントハックの設定
#else
#define BUG_DISABLE_BUGBEARD // バグベアードのOFF
#endif

#include “bug.h” // バグベアード本体の include
```

# Hello, Bugbeard! (17/19)

```
#if !defined(NDEBUG)
#define BUG_EVIL_CONTRACT // “悪魔の契約”
#define BUG_DEFINE_GLOBAL_LOGGER ¥
    new bugbeard::bug_tree_logger(new bugbeard::bug_file_writer_base(stderr))
    // 標準エラーヘッダー形式の出力を行うローガーの定義
```

```
#define BUG_S
#else
#define BUG_D
#endif
```

```
#include "bug
```

日付&時刻 : 2008-02-24 (日) 23:26:11.874

```
23:26:11.874 | ▽if (1 < current) == true; <tree.cpp>#104
23:26:11.874 | ▸·while (1 < current) == true; <tree.cpp>#109
23:26:11.874 | ▽if (current %p) == true; <tree.cpp>#111
23:26:11.874 | ▽if (pn) == false; <tree.cpp>#113
23:26:11.874 | ▽if (pn) == false; <tree.cpp>#113
23:26:11.874 | ▸·p(0x00D8FC5C) = 3(0x00000003) <tree.cpp>#126
23:26:11.874 | ▽if (current %p) == true; <tree.cpp>#111
23:26:11.874 | ▸·while (1 < current) == true; <tree.cpp>#109
23:26:11.874 | ▽if (current %p) == true; <tree.cpp>#111
23:26:11.874 | ▽if (pn) == false; <tree.cpp>#113
23:26:11.874 | ▽if (pn) == false; <tree.cpp>#113
23:26:11.874 | ▸·p(0x00D8FC5C) = 5(0x00000005) <tree.cpp>#126
23:26:11.874 | ▽if (current %p) == true; <tree.cpp>#111
23:26:11.874 | ▸·while (1 < current) == true; <tree.cpp>#109
23:26:11.874 | ▽if (current %p) == true; <tree.cpp>#111
23:26:11.874 | ▽if (pn) == false; <tree.cpp>#113
23:26:11.874 | ▽if (pn) == false; <tree.cpp>#113
```

# Hello, Bugbeard! (18/19)

If/while/switch 条件式の評価結果も出力

各種ステートメントのログを自動的に出力

```
#include <string>
#include <string.h>
#define BUG_DEFINE_GLOBAL_LOGGER
    new bugbeard::bug_tree_logger(new bugbeard::bug_file_logger_base(stderr))
    // ... ヘッダー形式の出力を行うログの定義
#endif
#define BUG_STATEMENT
#else
#define BUG_DEFINE_GLOBAL_LOGGER
#endif
#include <string>
```

日付&時刻 : 2008-02-24 (日) 23:26:11.874

自動的に階層化して出力

```
23:26:11.874 | ▽if (1 < current) == true; <tree.cpp>#104
23:26:11.874 |   ▸• while (1 < current) == true; <tree.cpp>#109
23:26:11.874 |     ▽if (current %p) == true; <tree.cpp>#111
23:26:11.874 |       ▽if (pn) == false; <tree.cpp>#113
23:26:11.874 |         ▸△if (pn) == false; <tree.cpp>#113
23:26:11.874 |         ▸• p(0x00D8FC5C) = 3(0x00000003) <tree.cpp>#126
23:26:11.874 |         ▸△if (current %p) == true; <tree.cpp>#111
23:26:11.874 |       ▸• while (1 < current) == true; <tree.cpp>#109
23:26:11.874 |         ▽if (current %p) == true; <tree.cpp>#111
23:26:11.874 |           ▽if (pn) == false; <tree.cpp>#113
23:26:11.874 |             ▸△if (pn) == false; <tree.cpp>#113
23:26:11.874 |             ▸• p(0x00D8FC5C) = 5(0x00000005) <tree.cpp>#126
23:26:11.874 |             ▸△if (current %p) == true; <tree.cpp>#111
23:26:11.874 |           ▸• while (1 < current) == true; <tree.cpp>#109
```

詳細についてはこちらを参照のこと <http://tricklib.com/cxx/ex/bugbeard/#statements>

```
23:26:11.874 |             ▸△if (pn) == false; <tree.cpp>#113
```

# Hello, Bugbeard! (19/19)

- Windows用情報収集サンプル
  - <http://tricklib.com/cxx/ex/bugbeard/#step2>
  - Windowsで次のような情報を収集する為の関数群が用意されています
    - Windowsバージョン情報
    - プロセスのEXE+全DLLのバージョン情報+ハッシュ値
    - メモリ情報
    - 全ドライブの各種情報
    - etc
  - Windows以外の環境でも次のような情報を収集する為の関数群が用意されています
    - コンパイラ情報
    - コマンドライン引数

バグベアード入門

既存のコードへの適用

# 既存のコードへの適用(1/4)

- ステートメントハックの為のソースコードの修正

- ステート内での変数定義の禁止

```
if (int i = ...) ...  
switch(int i = ...) ...  
while(int i = ...) ...
```

この形での変数定義は禁止

※ただし for 文についてはこの制限はありません

- 例外仕様構文の禁止

```
void func(...) throw(...);
```

例外仕様構文は使えません

# 既存のコードへの適用(2/4)

- 特定ブロックでのステートメントハックのオフ
  - ステートメントハックを使用する為の制限を受け入れがたい場所やどうしてもコンパイラ内部エラーが発生する箇所だけステートメントハックをオフにすることができます

# 既存のコードへの適用(3/4)

BUG\_STATEMENT\_HACK を #undef  
して bug.h を再 #include

```
#define BUG_STATEMENT_HACK // ステートメントハックの設定  
#include "bug.h" // バグベアード本体の include
```

ステートメントハックが有効な箇所

```
#undef BUG_STATEMENT_HACK // ステートメントハックの設定を解除  
#include "bug.h" // バグベアード本体の再include
```

ステートメントハックが無効な箇所

```
#define BUG_STATEMENT_HACK // ステートメントハックの再設定  
#include "bug.h" // バグベアード本体の再include
```

ステートメントハックが有効な箇所

BUG\_STATEMENT\_HACK を再度  
#define して bug.h を再 #include

# 既存のコードへの適用(4/4)

- マルチスレッドなプログラムへの適用
  - <http://tricklib.com/cxx/ex/bugbeard/#step3>
  - #define BUG\_MULTI\_THREAD の指定
  - 共用ライターの定義と各スレッド別にロガーを定義
  - 後述のプロファイラについても各スレッド別に定義

バグベアード入門

# プロファイリング

# プロファイリング(1/4)

- プロファイリング情報の出力指定

```
#if !defined(NDEBUG)
#define BUG_EVIL_CONTRACT // “悪魔の契約”
#define BUG_DEFINE_GLOBAL_LOGGER ¥
    new bugbeard::bug_tree_logger(new bugbeard::bug_file_writer_base(stderr))
    // 標準エラーヘッリー形式の出力を行うロガーの定義
#define BUG_DEFINE_GLOBAL_PROFILER ¥
    new bugbeard::bug_tsv_profiler( ¥
    new bugbeard::bug_file_writer(“profile.tsv”), ¥
    new bugbeard::bug_file_writer(“coverage.tsv”)
    // “profile.tsv” へプロファイル結果の出力を、
    // “coverage.tsv” へカバレッジ測定結果の出力を
    // 行うプロファイルロガーの定義
#else
#define BUG_DISABLE_BEARD // バグベアードのOFF
#endif
// プロファイラの設定
#include “bug.h” // バグベアード本体の include
```

- ※カバレッジ測定が不要な場合はその出力先ライターを省略可能

# プロファイリング(2/4)

- プロファイリング情報の出力指定

```
#if !defined(NDEBUG)
#define BUG_EVIL_CONTRACT // "悪魔の契約"
new bugbeard::bugbeard(new bugbeard::bug_file_writer_base(stderr))
// 標準エラーヘッダー形式の出力を行うログの定義
#define BUG_DEFINE_GLOBAL_PROFILER ¥
new bugbeard::bug_tsv_profiler( ¥
new bugbeard::bug_file_writer("profile.tsv"), ¥
new bugbeard::bug_file_writer("coverage.tsv"))
// "profile.tsv" へプロファイル結果の出力を、
// "coverage.tsv" へカバレッジ測定結果の出力を
// 定義
#else
#define BUG_DISABLE_BUGBEARD // バグベアードのOFF
#endif

#include "bug.h" // バグベアード本体の include
```

TSV形式での出力を行うプロファイラ

"profile.tsv" ファイルへ出力

デバッグ版

リリース版

# プロファイリング(3/4)

- プロファイリング情報の出力例

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	__FILE__	__LINE__	スコープ	合計総稼働時間	合計総バグ時	合計総実値	合計自稼働	合計自バグ	合計自実値	合計子稼働	合計子バグ	合計子実値	コールカ...
2	@OVERALL_SCOPE	0	@OVERALL_SCOPE	0.00357	0.002757	0.003212	0.00357	0.000085	0	0.001543	0.002672	0.003212	1
3	profile.cpp	48	if (1 < current) == true;	0.001865	0.002458	0.001882	0.001865	0.000382	0.001482	0.000969	0.002075	0.000399	1
4	profile.cpp	55	if (current %p) == false;	0.000539	0.000507	0.000031	0.000539	0.000507	0.000031	0	0	0	2
5	profile.cpp	55	if (current %p) == true;	0.000304	0.001444	0.000365	0.000304	0.000142	0.000162	0.001351	0.001302	0.000202	1
6	profile.cpp	57	if (pn) == true;	0.001351	0.001302	0.000202	0.001351	0.00115	0.0002	0.000153	0.000151	0.000002	1
7	profile.cpp	80	if (1 < pn) == false;	0.000153	0.000151	0.000002	0.000153	0.000151	0.000002	0	0	0	1
8	profile.cpp	88	if (1 < pn) == false;	0.000125	0.000123	0.000002	0.000125	0.000123	0.000002	0	0	0	1
9	profile.cpp	107	for	0.001543	0.002672	0.003212	0.001543	0.000213	0.00133	0.001865	0.002458	0.001882	1

- 各項目の説明はこちらを参照のこと
- <http://tricklib.com/cxx/ex/bugbeard/#profile-columns>

# プロファイリング(4/4)

- プロファイリングのポイント
  - [合計自実働時間](TotalSelfWorkTime)がより大きなスコープがパフォーマンス上のネック
  - [最小スタンプ](MinStamp)と[最大スタンプ](MaxStamp)で示される位置の動作ログを見比べることでそのスコープがもっとも速く動作した時ともっとも遅く動作した時の挙動を比較

バグベアード入門

# カバレッジ測定

# カバレッジ測定(1/3)

## ■ カバレッジ測定結果の出力指定

```
#if !defined(NDEBUG)
#define BUG_EVIL_CONTRACT // “悪魔の契約”
#define BUG_DEFINE_GLOBAL_LOGGER ¥
    new bugbeard::bug_tree_logger(new bugbeard::bug_file_writer_base(stderr))
    // 標準エラーヘッリー形式の出力を行うロガーの定義
#define BUG_DEFINE_GLOBAL_PROFILER ¥
    new bugbeard::bug_tsv_profiler(¥
    new bugbeard::bug_file_writer(“profile.tsv”), ¥
    new bugbeard::bug_file_writer(“coverage.tsv”)
    // “profile.tsv” へプロファイル結果の出力を、
    // “coverage.tsv” へカバレッジ測定結果の出力を
    // 行うプロファイルロガーの定義
#else
#define B “coverage.tsv” ファイルへ出力
#endif
```

デバッグ版

リリース版

```
#include “bug.h” // バグベアード本体の include
```

- ※プロファイリングのデータが不要な場合はその出力先ライターとして NULL を指定

# カバレッジ測定(2/3)

- カバレッジ測定結果の出力例

	A	B	C	D
1	FILE	LINE	スコープ	コールカウント
2	profile.cpp	48	if (1 < current) == false;	0
3	profile.cpp	48	if (1 < current) == true;	1
4	profile.cpp	55	if (current %p) == false;	2
5	profile.cpp	55	if (current %p) == true;	1
6	profile.cpp	57	if (pn) == false;	0
7	profile.cpp	57	if (pn) == true;	1
8	profile.cpp	60	if (1 < pn) == false;	1
9	profile.cpp	60	if (1 < pn) == true;	0
10	profile.cpp	88	if (1 < pn) == false;	1
11	profile.cpp	88	if (1 < pn) == true;	0

# カバレッジ測定(3/3)

- if文のカバレッジのみ
  - switch文やその他のカバレッジについては出力されない
  - 全く実行されなかったif文も出力されない

バグベアード入門

“悪魔の契約”

# “悪魔の契約”(1/3)

- バグベアードは外道なテクニックを使って作成されています
- 注意事項を把握した上でそれを了承する意思表示として `BUG_EVIL_CONTRACT` マクロを定義してください

# “悪魔の契約”(2/3)

- 注意事項
  - 規格違反
  - エラー情報の侵食
  - 構文の制限
  - 動作不安定
  - 出力ログの変形
  - 順序不安定

# “悪魔の契約”(3/3)

- 詳細は...
  - bug.h 内のコメント 「★バグベアードをご利用に当たっての注意事項」
  - <http://tricklib.com/cxx/ex/bugbeard/#evil-contract>

バグベアード入門

ご利用に当たってのヒント

# ご利用に当たってのヒント(1/2)

- ご利用に当たってのヒント
  - バグがあった場所のログ出力の為の埋め込みは消さない
  - 三項演算子のかわりに if を使用する
  - 明記が不要な場合でも return を記述する
  - フィルタ機能を利用する
  - デバッグ以外での利用
  - Excel を \*.tsv ログのビューアとして使用する場合は Excel2007 以降で

# ご利用に当たってのヒント(2/2)

- 詳細は...
  - bug.h 内のコメント 「☆バグベアードをご利用に当たってのヒント」
  - <http://tricklib.com/cxx/ex/bugbeard/#hints>

バグベアード入門

質疑応答



ご静聴ありがとうございました。