



Q&A about SuperTest for compiler users *working in safety critical markets*



SuperTestは何のためのツール？

SuperTestは、CおよびC++用のコンパイラのテストと検証スイートです。SuperTestには、テストを実行するツール（テストドライバ）、レポートジェネレータ、および大量のテストが含まれています。SuperTestは、コンパイラがCまたはC++プログラミング言語を正しく実装していることを確認するために使用されます。

C/C++言語標準規格？

C言語およびC++言語は、ISO規格でISO/IEC 9899:2011およびISO/IEC 14882:2014という名前で定義されています。これらの標準は定期的に更新されるため、例えば一般的な言語バージョン'C99'は、C言語の古いバージョンであるISO/IEC 9899:1999で指定されています。SuperTestのテストは、これらの標準に従って構成され、分類されます。GCCをベースにするコンパイラの中には、C言語に多くの拡張を提供するものがあります。そのような拡張は、ISOで定義された標準ではありません。

なぜ、コンパイラのユーザーがコンパイラをテストする必要があるのか？

コンパイラは非常に複雑なプログラムで、多くのオプションがあります。コンパイラに起因するエラーは、思いも寄らないところでアプリケーションを破壊することがあり、デバッグも困難です。

たとえコンパイラサプライヤがコンパイラをテストしたとしても、ユーザー側の特定のユースケース（環境オプションとコンパイラオプションの組み合わせ）でテストされないでしょう。

セーフティクリティカルまたはセキュリティドメインのアプリケーションの開発者は、コンパイラが特定のユースケース下でアプリケーションエラーを引き起こさないことへの検証が必要です。

CおよびC++用のコンパイラが正しいことを証明するには何が必要？

簡単な答えはSuperTestです。もう少し詳しく言うと、言語標準で指定されている「動作」を検証するテストスイートが必要です。この動作をテストするには、テストスイート、コンパイラ、ホストマシン、ターゲットハードウェア、またはシミュレータを使用してテストを実行する必要があります。テストフレームワークは、テストの結果を収集し、レポートを生成します。



SuperTest はコンパイラの認定に必要な C および C++のすべてをテストしますか？

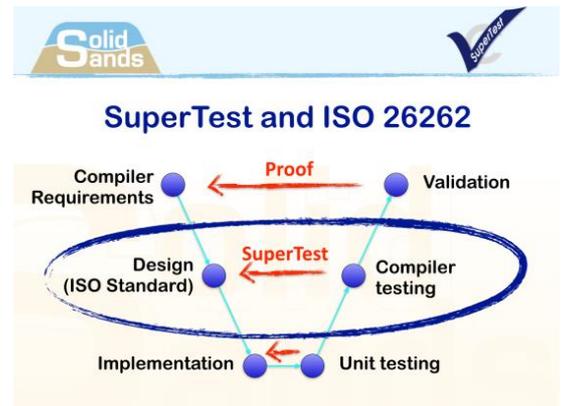
それが SuperTest の目標です。SuperTest が提供するテストは言語標準に従って構成されているため、テストと言語標準の間のマッピングを簡単に見つけることができます。言語実装には、「実装定義」の部分もあります。その場合、SuperTest はさまざまな選択肢を可能な限りカバーしようとします。C および C++で実装により定義された動作の多くは、基本型の算術サイズに関連しています。この特別なケースでは、SuperTest には 33 種類の異なる算術モデルに組み込んだ 33 種類のビット精度算術テストが含まれています。必要なバリエーションがまだ存在しない場合、Solid Sands 社はそのモデル用の新しい算術テストスイートを用意します。

標準的な C ライブラリーについてはどうですか？

はい、C ライブラリーの定義は言語標準の一部です。実際、これをコンパイラの検証と組み合わせることができます。その構成は同じにすることができます。

ISO 26262 等の機能安全規格の コンパイラのクオリフィケーション？

コンパイラがセーフティクリティカルなアプリケーション用のコードを生成するために使用される場合、コンパイラのクオリフィケーションが必要になることがあります。そのような場合、コンパイラはツールとして、ISO 26262 の Section 8.11 でツールクオリフィケーションの規則が定義されています。



さまざまな方法をツールの認定に使用できます。そのうちのツール仕様（この場合は言語標準）に対するテストが可能な技術の 1 つです。SuperTest は、言語標準に従って構築され、編成されているため、これに使用できます。さらに、ツールのクオリフィケーションを得るには、ユースケース（使用されるコンパイラオプションなど）、テストの設定、コンパイラのバージョンおよびその他の関連するパラメータの入念な証拠書類の提出が必要です。その証拠書類は、クオリフィケーションプロセスを繰り返すのに十分正確でなければなりません。

また、証拠書類には、アプリケーション開発者が遵守するユースケースを定義しなければなりません。コンパイラの障害が見つかった場合は、開発者がこれらの既知の障害に遭遇しないようにするための推奨または手順（緩和）も文書化する必要があります。

SuperTest を用いたクオリフィケーションを支援するパッケージやコンサルタントサービスを提供していますので、顧客は独自のユースケースで使用するコンパイラのクオリフィケーションを迅速かつ低価格で行えます。



SuperTest は IEC 61508 や ISO 26262 の認定を受けていますか？

SuperTest 自体は、IEC 61508 または ISO 26262 のクオリフィケーションを取得することはできません。なぜならクオリフィケーションは、特定のユースケース（特定のコンパイラと特定のコンパイラのユースケースとの組み合わせ）に対してのみ行うことができるためです。

SuperTest は、スタンドアローンや Q-Kit の一部として、複数のコンパイラクオリフィケーションプロジェクトで使用されています。SuperTest は、テストケースがコンパイラの（言語）仕様にどのように対応しているかを明確に文書化しているので、そのように使用することができます。

SuperTest のテストはどのようなもの？

ポジティブとネガティブの 2 種類のテストがあります。ポジティブテストには正しい C プログラムのみが含まれているため、正常にコンパイルされる必要があります。ネガティブテストの場合は診断が必要であり、コンパイルに失敗すると予想されます。

すべてのテストプログラムに `def.h` というファイルが含まれています。このファイルは、テストプログラムによって使用されるいくつかのマクロを定義します。

Example 1, a simple positive test:

```
/*
 * (c) Copyright 2015 by Solid Sands B.V.,
 * Amsterdam, the Netherlands. All rights reserved.
 * Subject to conditions in the RESTRICTIONS file.
 */

#include "def.h"

MAIN{
    CVAL_HEADER("compound literal (sizeof incomplete array)");
    CVAL_VERIFY(sizeof((int []) {1, 2, 3, 4}) == 4 * sizeof (int));
}
```

このテストは、`sizeof ()` 演算子を使用してサイズの比較を実行します。比較の左側に複合リテラルがあります。この場合はサイズが不明な整数配列です。サイズは 4 つの要素のイニシャライザリストによって決まります。したがって、サイズは 4 つの整数のサイズです。右側では、制御値は、整数のサイズに 4 を掛けて計算されます。

C99 Standard (ISO/IEC 9899:1999), Section 6.5.2.5, Paragraph 4:

A postfix expression that consists of a parenthesized type name followed by a brace-enclosed list of initializers is a compound literal. It provides an unnamed object whose value is given by the initializer list.



括弧で囲まれた型名とそれに続く波括弧で囲まれた初期化子の並びで構成される後置式を複合リテラル (compound literal) と呼ぶ。複合リテラルは、初期化子並びで与えられる値をもつ名前のないオブジェクトを与える

C99 Standard (ISO/IEC 9899:1999), Section 6.5.2.5, Paragraph 5:

If the type name specifies an array of unknown size, the size is determined by the initializer list as specified in 6.7.8, and the type of the compound literal is that of the completed array type.

型名が大きさの分からない配列を指定している場合、6.7.8 で規定するとおりに初期化子の並びによって大きさが決まる。さらに、この場合の複合リテラルの型は完全にされた配列型とする。

Example 2, a simple negative test:

```
/*
 * (c) Copyright 2015 by Solid Sands B.V.,
 * Amsterdam, the Netherlands. All rights reserved.
 * Subject to conditions in the RESTRICTIONS file.
 */

int v [2][ ][1] = { {{3}, {2}}, {{1}, {0}} };

#include "def.h"

MAIN{
    CVAL_HEADER("Only 1st dimension may be omitted.");
    CVAL_VERIFY(0);
}
```

不完全な型の配列は構築できないので、複数の次元を持つ配列は最初の次元では未知の境界しか持てません。したがって、このテストはコンパイル時に失敗することが予想されます。

C99 Standard (ISO/IEC 9899:1999), Section 6.2.5, Paragraph 20:

An array type describes a contiguously allocated nonempty set of objects with a particular member object type, called the element type (since object types do not include incomplete types, an array of incomplete type cannot be constructed).

配列型 (array type) は、要素型 (element type) と呼ぶ特定のメンバオブジェクト型をもつ空でないオブジェクトの集合を連続して割り付けたものを表す(オブジェクト型は不完全型を含まないので、不完全型の配列は作ることができない)。



C99 Standard (ISO/IEC 9899:1999), Section 6.2.5, Paragraph 22:
An array type of unknown size is an incomplete type.

大きさの分からない配列型は、不完全型とする。

C99 Standard (ISO/IEC 9899:1999), Section 6.7.5.2, Paragraph 4:
If the size is not present, the array type is an incomplete type.

大きさの指定がない場合、その配列型は不完全型とする。

Example 3, a positive optimization test:

```
/*
 * (c) Copyright 2015 by Solid Sands B.V.,
 * Amsterdam, the Netherlands. All rights reserved.
 * Subject to conditions in the RESTRICTIONS file.
 */

#include "def.h"

int test(int j, int k, int recurs){
    int x = 0;
    int result = 0;
    if (recurs)    { test(0, 0, 0); } /* prevent inlining */
    bb2:
    if(j == 0)    { x = 1;          goto bb12; }
    else {
        bb4:
        if(k == 0) {                goto bb13; }
        else      { k = 0;          goto bb12; }
    }
    bb12:
    if(x == 0)    { result = 20; goto bb2; }
    else         { result = 10; goto bb4; }
    bb13:
    return result;
}

MAIN{
    CVAL_HEADER("irreducible flow and constant propagation");
    CVAL_VERIFY(test(1, 1, 0) == 20);
    CVAL_VERIFY(test(0, 1, 0) == 10);
}
```



このテストは、コンパイラがコードを最適化しようとするときのコンパイラの正確性を検証することを目的としています。このテストは、非可約なループを含むように特別に設計されています。これはまれですが、コンパイラは変換前にそのようなループを正しく識別しなければなりません。

SuperTest を使用するには？

SuperTest は商用ソフトウェアですので、Solid Sands 社からライセンスを取得する必要があります。次に SuperTest をインストールして、コンパイラとターゲットまたはシミュレータを使用するように SuperTest を設定する必要があります。そして SuperTest を実行します。

SuperTest のインストールについて

SuperTest は移植性が高く、Mac OS や Windows、あらゆる Unix や Linux にインストールすることができます。*nix システムでは、SuperTest をインストールするためにいくつかの開発ツール（ホスト C コンパイラや Perl など）が必要です。Windows では、SuperTest は Cygwin 環境で実行するのが最も簡単です。SuperTest のインストールはほんの数分で済みます。SuperTest は、インストールディレクトリにのみインストールされ、ホストシステムには他の変更は行われません。したがって、異なるバージョンの SuperTest を同じシステムに並べてインストールすることもできます。

SuperTest の設定について

テストを実行するには、SuperTest のテストドライバが、実行するテスト、コンパイラを実行する方法、コンパイルされたプログラムを実行する方法を知っている必要があります。これらはすべて、ユーザーが設定可能ないくつかのファイルとスクリプトで定義されています。トップレベルの設定ファイルには、コンパイラの名前、コンパイルとリンクのフラグ、実行するテストのリスト、LOG ディレクトリの名前などが定義されています。次に、テストのコンパイルとテストの実行のための別々のスクリプトがあります。テストドライバは、コンパイルと実行をコマンドラインから実行できることと、コンパイルとテストの実行の成功または失敗に関するフィードバックを得る必要があります。

SuperTest のテスト実行について

SuperTest が設定されると、設定はテストドライバに渡されて実行されます。通常は、一連のテストをまとめて実行しますが、デバッグ時には特定のテストケースを 1 つ実行することもできます。SuperTest には、プロセスを高速化するために、現在のマシンまたはマシンのネットワーク上でも並行してテストを実行するオプションがあります。



テスト結果について

SuperTest はすべてのテスト実行の詳細なログを保持します。SuperTest の 'log-report' コマンドを使用すると、テスト結果を読みやすく包括的なレポートにすることができます。閲覧を容易にする、ASCII レポートと HTML レポートの両方を生成することができます。

テストにかかる時間について

これは、テストリストのサイズ、コンパイラの種類、およびターゲットへのテストのロードと実行にかかる時間によって異なります。テスト自体は一般的に短く、実行に時間がかかりません。ホストネイティブコンパイラの C99 検証では、約 5000 のテストプログラムを実行するのに 10 分もかかりません。

コンパクトなターゲットでも使用できるか？

はい。多くのテストプログラムは数キロバイト未満のコードにコンパイルされ、使用するランタイムリソースは僅かです。もちろん、SuperTest にはいくつかの非常に大きなテストが含まれており、コンパイルして実行を確認されますが、これらは簡単にスキップすることもできます。

SuperTest のテストには、通常、テストが失敗した場合にわかりやすい診断を生成する診断ライブラリーへの呼び出しが含まれています。ターゲットのリソースに制約がある、または I/O 機能を持たないなどの場合、診断ライブラリーは完全にスキップすることもできます。

Testing your compilers really isn't that difficult



富士設備工業株式会社 電子機器事業部 www.fuji-setsu.co.jp

(c) Copyright 2017 by Solid Sands B.V., Amsterdam, the Netherlands
SuperTest™ is a trademark of Solid Sands B.V., Amsterdam, The Netherlands.