# ABI テスト スイート

## よく寄せられる質問 (FAQ) v1.0.0

文責：スニル・スリヴァスタヴァ
2014 年 8 月 12 日

1.どの clang/LLVM のバージョンが、ABI テスト スイートと合わせて使用できますか?

当文書執筆段階では、ABI テスト スイートは svn/trunk revision #213127 を使用したものが検証済みです。このバージョンは、v3.5 の分岐に先立ったリビジョンとなります。clang v3.5 の正式版のリリース時に、本テスト スイートとの検証を行う予定です。clang v3.4 を使用した場合、いくつかの不具合があるため、テストの中にはパスできないものもあります。これらの不具合は v3.5 で修正されています。

2.この ABI テスト スイートは、clang の将来のバージョンでも使用可能となる予定ですか?

はい。これまでと同様に、clang の将来のバージョンで ABI に変更があった場合、本テスト スイートが変更点を検出します。つまりは、それがテスト スイートの目的です。このような場合、ABI を以前の状態に戻すか、テスト スイートをアップデートして変更を反映する必要があります。

3.ABI テスト スイートには、x86 32 ビットと x86 64 ビットの 2 つの構成があります。その他の構成を追加するにはどうしたらいいですか?

残念ながら、それについての有効な解決策はありません。本テスト スイートは、IA64 ABI 仕様と、C レベル レイアウトを定義するベース ドキュメントの両方に依存します (ベース ルール、基底部規則)。このため、任意の基底部規則のセットについて IA64 ABI をテストできるテスト スイートを作成することは極めて困難です。SN Systems では、最も一般的に使用されている 2 つの基底部規則を採用することにしました。

4.現在サポートされている構成の 1 つに対して新しいテストを追加するにはどうしたらいいですか？

既存のテストと似たやり方で追加できるテストもありますが、(本テスト スイートで実行されるテストの大多数である) オブジェクト レイアウト テストは、自動テスト ジェネレータによって自動生成されていることにご注意ください。現在、SN Systems では、テスト ジェネレータをオープン ソース コミュニティに提供することができません。提供できない、ということは当社にとってマイナスとなると認識しています。そのため、将来的に、clang テクノロジを基にテスト ジェネレータを再実装し、オープン ソース コミュニティへ提供できるよう、検討中です。

5.自動テスト ジェネレータの役割は何ですか?

テスト ジェネレータは、C/C++ ソースを読み込み、標準的なコンパイラのオブジェクト レイアウトを実行します。その際は、通常のオブジェクトコードを生成する代わりに、C/C++ ソース コードを出力して、実行するレイアウトを生成したばかりのレイアウトでテストします。これは、1 構成につき一度実行され、レイアウト ルールがそれぞれの要件を満たすよう変更されます。複数の構成をまとめるときは、ABISELECT マクロを使用します。

6.リクエストがあった構成について、自動テスト ジェネレータを実行していただくことはできますか?

いいえ。SN Systems では、x86 32 ビットおよび x86 64 ビットの構成を提供し、コミュニティ ユーザーの皆様が、これらの共通プラットフォーム上で ABI の品質を保てるようにしています。可能性のあるすべての構成をサポートすることはできませんが、将来的には他の構成を追加することがあるかもしれません。

7.このテスト スイートで使用できる g++ のバージョンは何ですか?

g++ コンパイラはまだテストされていません。理論上は問題なく動作するはずですが、g++ と clang 間でわずかな違いが見られる場合があります。clang は、レイアウトの互換性においては g++ を規範としてきましたが、clang と g++ は分化してしまったようです。

8.このテスト スイートは他の IA64-ABI コンパイラとともに使用できますか?

他のコンパイラが clang の ABI レイアウトと完全に一致している場合にのみ使用できます。そのため、おそらく、いくつかのエラーが発生するでしょう。

9.自分のコンパイラを使用して、不可避な非互換性が原因で 23 個のエラーが発生した場合、これらをどう解決したらいいでしょうか。

このような「小さな」不一致は、以下のような方法で解決できます。

- 解決策 1：失敗したテストをメモしておいて、次の実行でこれらのエラーを無視します。
- 解決策 2：失敗したテストを skip_list（以下の項目 13 を参照）に追加して、これらを XFAIL としてマークします。テストを XFAIL としてマークすることは、テストを無視することとは異なりますのでご注意ください。その後、XFAIL テストがパスし始めると、これはテスト失敗としてフラグされます。
- 解決策 3：失敗しているテストの実行可能ファイルを実行して、その出力結果を「Golden Master」（項目 14 を参照）として保存し、その後のすべての実行結果をこの出力と比較します。
- 解決策 4：独自の checker を作成します。ほとんどのテストで最後の RUN 行は、以下のようになります。

```
runtool %t2%exeext | checker "TEST PASSED"
```

「checker」は「grep」にデフォルト設定されますが、ユーザーが独自のフィルタを作成して、その入力を「Golden Master」として格納させるか、入力結果を「Golden Master」と比較させることもできます。エラー メッセージには、ファイル名と行番号が示されるため、checker はどのファイルをチェックしているかを常に把握できます。弊社では、このように高度な checker はまだ書けていませんが、将来的には書くことができるのではないでしょうか。

- 解決策 5：コンパイラの動作に合わせてテストを修正する checker を作成します。
- 解決策 6：ローカル コピーで、該当するテストを修正または削除します。

10.テストをターゲット マシンで実行しなければならない理由は何ですか?

テスト スイートは、生成されたアセンブリ、オブジェクト コード、または clang IL を精査してレイアウトを検証します。ただし、これを行うには、複数のプラットフォームについて、アセンブリ言語パーサーまたは IL ダンプ パーサーを作成しておく必要があります。弊社では、その作業を行う必要のない方法を選択しました。オブジェクト レイアウトには、コンストラクタとデストラクタの vtable のように、静的にテストを行うことが難しいものがあります。VTT や vtable は静的にテストでき、その一部はコンストラクションまたはデストラクションの段階で使用されますが、それにはランタイムの点検か、またはコード エミュレータのようなものが必要になります。

11.静的アサートを使用しないのはなぜですか?

静的アサートは、アセンブリ コードの読み込みより weak です。また、静的なチェックではエラーをより迅速に検出できますが、1 つのファイルで静的チェックに 1 つでも失敗すると、エラー チェックが停止されてしまいます。そのため、最初のエラー以外が検出されない可能性があります。testsuite.h 内には、NO_STATIC_CHECKS マクロが定義されています。この定義を解除すると、可能な限り静的チェックが使用されます。

12.一意の ifdef にテストを含めなかったのはなぜですか? 他のテスト スイートでは、ユーザーが選択的に削除できるよう、このような措置がとられています…。

一部のケースにおいては、そうしたほうが好ましかったかもしれません。ただし、その場合は複雑な微分グラフが必要となりますし、微分ツリーの途中にあるクラスは削除できないなど、メカニズムが複雑化します。とは言え、この措置も将来的には採用される可能性があります。

13.テスト ファイルのサフィックスに .x と .xpp を使用するのはなぜですか?

各構成の各ファイル単位で、特定のテストを XFAIL としてマーク付けするメカニズムを模索していました。これは、lit では提供されないメカニズムです（ただし、lit は lit.local.cfg 経由でディレクトリ全体を無視できます)。そこで、skip_list メカニズムを考案しました。skip_list は、トップレベルの python runner で指定できます。実行開始後、lit.site.cfg はすべての .x および .xpp ファイルを対応する .c および .cpp ファイルにコピーし、

skip_list にあるファイルについては、XFAIL 行を追加します。その後、テストは .c または .cpp ファイル上で実行されます。この「コピー」処理には時間がかかりますが（1 分強）、新しいディレクトリにつき 1 度しか実行されません。その後の実行でも、skip_list にないファイルが再度コピーされることはないため、その後のオーバーヘッドはそれほど問題になりません（後から考えると、skip_list ではなく xfail_list と呼ぶべきでした）。

14.テスト実行可能ファイルの「Golden Master」の結果を生成するにはどうしたらいいですか?

実行可能ファイルで「–v」オプションを使用するか、「-v –v」オプションを使用すると詳細が出力されます。