

LabVIEW 開発者認定 (CLD) 認定プログラムと試験の概要

認定プログラムの概要

ナショナルインスツルメンツ LabVIEW 認定プログラムは、以下の 3 段階の認定レベルで構成されています。

- LabVIEW 準開発者認定試験 (CLAD)
- LabVIEW 開発者認定試験 (CLD)
- LabVIEW 設計者認定試験 (CLA)

各レベルの認定は、次のレベルの認定を受けるための前提条件になります。

CLAD 認定者は、LabVIEW 開発システムで提供されるコア機能と機能範囲を、幅広く理解しています。さらにそれらの知識を応用して、小規模な LabVIEW モジュールの開発、デバッグ、保守を行える能力を保有しています。CLAD 認定者の標準的な経験レベルは、LabVIEW 開発システムの使用経験がおよそ 6~9 か月です。

CLD 認定者は、中~大規模な LabVIEW アプリケーションの開発、デバッグ、拡張、保守の経験者です。CLD 認定者は、中~大規模の LabVIEW アプリケーション開発において累積で 12~18 か月程度の経験を持つプロフェッショナルです。

CLA 認定者は、複数の開発者を組織する環境での LabVIEW アプリケーションの設計に習熟しています。CLA 認定者は、プロジェクトの仕様を管理可能な LabVIEW コンポーネントに細分化するための技術的専門知識とソフトウェア開発経験を保有するだけでなく、プロジェクト管理ツールや構成管理ツールを効果的に活用してプロジェクトの完成を指揮するための経験を持ちます。CLA 認定者は、中~大規模の LabVIEW アプリケーション開発において累積で 24 か月程度の経験を持つプロフェッショナルです。



メモ : CLAD 認定は CLD 試験を受けるための前提条件です。CLD 認定は CLA 試験を受けるための前提条件です。それぞれの試験においてこの要件は例外なく適用されます。

LabVIEW 開発者認定 (CLD) 認定プログラムと試験の概要

試験の概要

対象製品：試験会場のコンピュータには、LabVIEW開発システム 8.2 以降がインストールされています。2007年10月からは、LabVIEW 8.5 以降がインストールされています。試験で使用するLabVIEWのバージョンについての詳細は、日本ナショナルインスツルメンツまで事前にお問い合わせください。使用されるLabVIEW環境に慣れていないという理由で試験の前後に特別に時間を設けることはいたしません。(LabVIEWプロフェッショナル開発システムで提供される機能の詳細については、http://www.ni.com/labview/ja/how_to_buy.htmを参照してください。)

試験時間：4 時間

試験形態：実技試験—アプリケーション開発

合格基準：75%

試験では、LabVIEW を使用した計測と自動化のアプリケーション開発における問題解決スキル、知識、経験がテストされます。試験内容はソフトウェア開発のみを対象とし、ハードウェアは一切含まれません。

LabVIEW に内蔵されたリソース（『LabVIEW ヘルプ』、サンプル、テンプレートなど）を試験中に使用してもかまいません。外部で作成した VI や資料等の持込みは禁止されています。

詳細なアプリケーション仕様が提供されます。仕様は、アプリケーションの一般的な要件と技術的な要件で構成されています。仕様に関する情報やサンプル問題は、トレーニングのサイトからダウンロードできます。

試験の解答はディスクに保存して試験監督官に提出していただきます。試験の完全性を維持するため、試験問題や解答のホッチキス取り外し、コピー、複製は禁止されています。既定に違反があると不合格となります。

LabVIEW 開発者認定 (CLD)
認定プログラムと試験の概要

お申し込みの流れ

1. 認定試験合意書に必要事項をご記入ください。
2. 米国でのお申し込みは、ナショナルインスツルメンツまで合意書をFAXにて送信した後、電話（米国（888）484-4436）またはオンライン（ni.com/training）にてお申し込みください。
3. 米国以外でのお申し込みは、最寄のナショナルインスツルメンツ営業窓口まで、試験のお申し込みと試験日程についてお問い合わせください。
4. 各国の通貨で、既定された認定試験の受験料をお支払いください。

試験全般に関するご質問はEメールをご利用ください。 trainingjapan@ni.com

LabVIEW 開発者認定 (CLD)
認定プログラムと試験の概要

試験内容

1. 設計の概念
2. ユーザインタフェースの設計手法
3. ブロックダイアグラムの設計手法
4. プログラミング手法
5. サブ VI の設計手法
6. デザインパターンの選択
7. タイミング
8. エラー
9. ドキュメント化
10. テスト

LabVIEW 開発者認定 (CLD)

認定プログラムと試験の概要

試験内容 (概要)

トピック	項目
1. 設計の概念	<ul style="list-style-type: none"> a. モジュール化、拡張性、可読性、保守性 b. 凝集度と結合度 c. 階層的な設計 d. ファイル構造
2. ユーザインタフェース (フロントパネル) の設計手法	<ul style="list-style-type: none"> a. カラースキーム b. オブジェクトのグループ化と整列 c. プロパティの設定 d. オブジェクトのカスタマイズ e. 状態管理 <ul style="list-style-type: none"> i. 静的/動的 ii. 初期化時とアプリケーション停止時 f. アイコンのデザイン
3. ブロックダイアグラムの設計手法	<ul style="list-style-type: none"> a. データフロー b. 可読性の向上
4. プログラミング手法	<ul style="list-style-type: none"> a. データ要素 b. 関数とサブ VI c. プログラミングストラクチャ d. データストラクチャ e. リファレンス、プロパティノード
5. サブ VI の設計手法	<ul style="list-style-type: none"> a. モジュール化と凝集度 b. フロントパネルのレイアウト c. コネクタペーンとアイコン
6. デザインパターンの選択	<ul style="list-style-type: none"> a. 拡張性と保守性 b. 応答性とブロック防止 c. デザインパターン <ul style="list-style-type: none"> i. シンプルステートマシン ii. ユーザインタフェースイベントハンドラ iii. キューメッセージハンドラ iv. 生産者/消費者 (データ) v. 生産者/消費者 (イベント) vi. 機能的グローバル変数
7. タイミング	<ul style="list-style-type: none"> a. タイミング関数 b. タイミングメカニズム <ul style="list-style-type: none"> i. イベントストラクチャのタイムアウト ii. 同期関数のタイムアウト iii. タイミングストラクチャ c. タイミング Express VI と機能的グローバル変数

LabVIEW 開発者認定 (CLD)
認定プログラムと試験の概要

8. エラー	a. エラー処理 b. エラーレポート
9. ドキュメント化	a. フロントパネル b. ブロックダイアグラム c. VIプロパティ
10. テスト	a. コードとドキュメント化のレビュー b. 機能 c. エラー

LabVIEW 開発者認定 (CLD)
認定プログラムと試験の概要

LabVIEW 開発者認定 (CLD) 試験内容の詳細

1. 設計の概念

a. モジュール化、拡張性、可読性、保守性

1. 以下の要件を満たす LabVIEW アプリケーション (VI) を作成します。

- a) モジュール化：VI の機能がモジュールまたはサブ VI に分離されている
- b) 拡張性：この VI で、より大規模なデータセットや追加のプログラム状態を扱う場合にも、ユーザインタフェースやブロックダイアグラムの変更がほとんど必要ないかまったく必要ない
- c) 可読性：適切なドキュメント化とプログラミングスタイルにより、VI 自体に関する情報がわかるようになっている
- d) 保守性：モジュールやアプリケーションの当初の目的を変更することなく、VI の変更が容易にできるようになっている

b. 凝集度と結合度

1. 以下の要件を満たす LabVIEW モジュールを作成します。

- a) 高凝集性：モジュールの目標が明確に定義され、公開されている
- b) 疎結合：モジュールの機能を完成/補完する他のモジュールへの依存度が最小限に留められている

c. 階層設計

1. 上記のテクニックを使用して論理階層を設計した LabVIEW VI を作成します。

d. ファイル構造

- 1. ソフトウェアの階層構造を考慮して、ファイルシステム内で VI を編成します。
- 2. アプリケーションのフォルダを作成し、わかりやすい名前を付けます。
- 3. メイン VI (トップレベル VI) を、アプリケーションフォルダ内でアクセス可能にします。
- 4. サブ VI や制御器用に独立したフォルダを作成します。

2. ユーザインタフェース (フロントパネル) の設計手法

a. カラースキーム

- 1. 一貫性のあるシステムカラースキームを使用して VI のユーザインタフェースを作成します。
- 2. 必要に応じてパステルカラーを使用し、鮮やかな色は避けるようにします。
- 3. 背景およびユーザインタフェースオブジェクトのカラースキームには、『LabVIEW ヘルプ』の「LabVIEW Style Checklist (英語)」トピックのガイドラインを使用します。

LabVIEW 開発者認定 (CLD) 認定プログラムと試験の概要

- b. オブジェクトのグループ化と整列
 - 1. 配列、クラスタ、装飾体などを使用して、論理的に関連するユーザインタフェースオブジェクトをグループ化します。
 - 2. レイアウトの統一性と一貫性が得られるように、オブジェクトとそのラベルを整列させます。
 - c. プロパティの設定
 - 1. 使いやすさとパフォーマンスが向上するように、フロントパネルオブジェクトの設定を適切に選択します。
 - d. オブジェクトのカスタマイズ
 - 1. フロントパネルオブジェクトの外観を変更します。
 - 2. カスタム制御器のタイプ定義または指定タイプ定義を作成し、アプリケーションの拡張性を広げます。
 - e. 状態管理
 - 1. 「プロパティ」ダイアログボックスを静的に使用したりプロパティノードを動的に使用して、制御器の値や属性を設定します。
 - 2. アプリケーションのロード時、開始時、停止時に、初期化または制御値の設定を行います。
 - f. アイコンのデザイン
 - 1. アプリケーションやモジュールの機能を表わすように、メイン VI (トップレベル VI) およびサブ VI のアイコンをデザインします。
 - 2. メイン VI とサブ VI の間で一貫性のある、統一したアイコンデザインスキームを使用します。
3. ブロックダイアグラムの設計手法
- a. データフロー
 - 1. VI およびプロパティノードのエラー端子を使用して、データフローを確立します。
 - 2. エラー端子を持たない VI/関数には、シーケンスストラクチャを使用してデータフローを確立します。
 - b. 可読性の向上
 - 1. 1024 x 768 の画面解像度に合わせてブロックダイアグラムを作成します。
 - 2. ユーザのスクロールが 1 方向のみに限定されるように、ブロックダイアグラムのサイズを制限します。
 - 3. VI および関数を等間隔に配置します。小さな領域に多くの VI や関数が密集しすぎないようにします。
 - 4. 一貫したスキームを使用して、VI および関数を等間隔に配置します。
 - 5. ワイヤの屈折を避け、できるだけワイヤが直線になるようにします。
 - 6. 正しい端子に接続されたことがわかるように、ワイヤを配線します。
 - 7. 左から右、上から下のデータフローに従って VI および関数を配線します。

LabVIEW 開発者認定 (CLD) 認定プログラムと試験の概要

8. ストラクチャやストラクチャ枠の下に配線しないようにします。
9. ストラクチャ枠上のトンネルが重複しないようにします。
10. カラーを使用してブロックダイアグラムのセクションを区別することは避けます。必要であればシステムカラーのみを使用します。

4. プログラミング手法

a. データ要素

1. 制御器、表示器、定数に対して適切なデータタイプを使用します。
2. 制御器から直接読み取ります。ローカル変数、グローバル変数、プロパティノードの使用は避けます。
3. 表示器を直接更新します。ローカル変数、グローバル変数、プロパティノードの使用は避けます。
4. ローカル変数は、制御器の更新に使用します。
5. プロパティノードは、制御器および表示器の属性変更で使用します。
6. リファレンスは、フロントパネル上の制御器や表示器がサブ VI から影響を受ける場合にのみ使用します。
7. 競合状態を回避するため、ローカル変数やグローバル変数へのアクセスを保護します。
8. すべてのインスタンスで同じタイプを維持して拡張性を向上させるため、データ要素をタイプ定義します。
9. ユーザインタフェースのアクセスが不要なデータ要素には、定数を使用します。

b. 関数とサブ VI

1. 関数の使用を最適化します。最小限の数の関数でタスクを実行します。
2. 入力でのデータ強制を避けます。
3. 使用可能な場合は、エラー端子を使用します。
4. 明示的に開いたリファレンスを閉じます。

c. プログラミングストラクチャ

1. VI に対して適切なプログラミングストラクチャを選択します。
2. 必要に応じてシフトレジスタを初期化します。初期化しないシフトレジスタの意味を理解します。
3. ストラクチャのネストの深さは 2 レベルまでにします。
4. ワイヤが使用できない箇所でデータフローを確立するには、単一フレームのフラットシーケンスストラクチャを使用します。
5. データやステート情報を渡す目的でシーケンスローカルを使用するのは避けます。
6. 拡張性を高めるために、可能な限りシーケンスストラクチャをケースストラクチャに置き換えます。
7. ストラクチャ上のトンネルでデフォルト値を使用することは避けます。
8. ユーザインタフェースイベントの処理には、イベントストラクチャを使用します。

LabVIEW 開発者認定 (CLD) 認定プログラムと試験の概要

9. タイミングイベントを確定的に処理するには、タイミングストラクチャを使用します。
 - d. データストラクチャ
 1. 論理的に関連するデータ要素を、適切なデータストラクチャにグループ化します。
 2. ブロックダイアグラム上に作業用のストラクチャを作成して、ユーザインタフェースでの操作が不要なデータ/ステート情報をグループ化して操作します。
 - e. リファレンス、プロパティノード (リファレンスの取得、リファレンスを閉じる)
 1. 同一 VI 内の属性を操作するには、内部的にリンクされたプロパティノードを使用します。
 2. フロントパネルオブジェクトへのリファレンスは、サブ VI からフロントパネルオブジェクトを更新する場合にのみ使用します。
 3. プロパティノードの意味を認識し、適切に適用します。
 4. 明示的に開いたリファレンスを閉じます。
- ### 5. サブ VI の設計手法
- a. 凝集とモジュール化
 1. 明確に定義した単一機能を実行するようにサブVIを設計します。
 2. サブVIのタスクを補う関数/サブVIを呼び出します。
 - b. フロントパネルとブロックダイアグラム
 1. 入力、出力、作業用データセクションが明確に定義されるようにサブVIのユーザインタフェースを設計します。
 2. エラー入力クラスタとエラー出力クラスタを追加し、これらをコネクタペーンに配線します。
 3. エラー入力端子を、ケースストラクチャの「エラー」および「エラーなし」ケースのセレクト端子に配線します。
 4. エラーケースは、上流のエラーを確実に通過するようにします。
 5. すべてのサブVIコードを「エラーなし」ケースに配置し、エラーワイヤを関数、サブVI、プロパティノードのエラー端子に確実に接続します。
 6. さまざまなソースからのエラーを結合し、そのエラーを呼び出し側VIにエラー出力端子を介して渡します。
 7. 「4. プログラミング手法」に記載されているプログラミング手法を使用します。
 - c. コネクタペーンとアイコン
 1. コネクタペーンの作成には、『LabVIEW ヘルプ』の「LabVIEW Style Checklist (英語)」トピックのガイドラインを使用します。
 2. 端子の種類として必須、推奨、任意を指定します。

LabVIEW 開発者認定 (CLD) 認定プログラムと試験の概要

3. メイン VI およびサブ VI のアイコンは、それらの機能を特定するようにデザインします。
4. メイン VI とサブ VI の間で一貫したアイコンスタイルを使用します。

6. デザインパターンの選択

a. 拡張性と保守性

1. プロジェクトの仕様を分析して、メイン VI に最も適したデザインパターンを選択します。
2. サブ VI の必要機能を実現するために最も適したデザインパターンを選択します。
3. 拡張性や保守し易さを制限しないアーキテクチャを使用します。
4. ステートやデータを管理するには、タイプ定義されたデータストラクチャを使用します。

b. 応答性とブロック防止

1. ユーザインタフェースの操作に 100ms 以内に応答し、適切な頻度で表示器を更新するデザインパターンを使用します。
2. CPU リソースの 100% を占有しないデザインパターンとデザインテクニックを使用します。

c. デザインパターン

1. ユーザインタフェースをポーリングするトップレベル VI には、シンプルステートマシンデザインパターンを選択します。
2. 入力状態に基づき 1 つ以上の関数やサブ VI を実行する必要があるサブ VI 内では、シンプルステートマシンデザインパターンを選択します。
3. ユーザインタフェースの動作に素早く応答する必要がある場合には、ユーザインタフェースイベントハンドラデザインパターンを選択します。
4. 1 つ以上の状態を保存するには、キューメッセージハンドラデザインパターンを使用します。
5. 別のデータ消費ループと並列して、生産者ループでデータ/ステートの生成と保存を行う場合には、生産者/消費者 (データ) デザインパターンを選択します。
6. 生産者ループのユーザインタフェースイベントに応答し、1 つ以上の生産者ループに対するイベントの処理を延期する場合には、生産者/消費者 (イベント) デザインパターンを選択します。
7. パフォーマンスとアクセス制御を向上させるため、サブ VI ではグローバル変数ではなく、機能的グローバル変数デザインパターンを選択します。

LabVIEW 開発者認定 (CLD) 認定プログラムと試験の概要

7. タイミング

a. 実行とソフトウェアタイミング

1. システム上の VI の実行速度を制御し、プロセッサが外部イベントやシステムタスクに応答可能にするために最も適したタイミング関数を決定します。
2. ソフトウェア動作やタスクを、設定された時間内または設定された時間後に実行するために最も適したメカニズムを決定します。

b. タイミング関数

1. ループの実行速度を制御し、プロセッサが外部イベントやシステムタスクに応答可能にするには、「待機 (Wait)」関数を決定します。
2. ループの実行速度を制御し、複数ループを同期させるには「次のミリ秒倍数まで待機 (Wait Until Next ms Multiple)」関数を使用します。
3. ソフトウェア動作のタイミングを合わせる目的で「待機 (Wait)」関数を使用することは避けます。
4. ソフトウェア動作のタイミングを合わせるには、「ティックカウント (Tick Count)」関数および「日付/時間を秒で取得 (Get Date/Time in Seconds function)」関数を使用します。
5. 「ティックカウント」関数を使用する場合は、時間の折り返しを考慮します。
6. 「日付/時間を秒で取得」関数を使用する場合は、システムクロックを変更するとエラーが発生することをユーザに警告します。

c. タイミングメカニズム

1. ソフトウェア動作のタイミングを合わせるには、イベントストラクチャの「タイムアウト」ケースを使用します。
2. ソフトウェア動作のタイミングを合わせるには、キュー関数やノーティファイ関数のタイムアウト入力を使用します。
3. 確定的なソフトウェアタイミング動作を実行するには、タイミングストラクチャを使用します。

d. タイミング Express VI と機能的グローバル変数

1. ソフトウェア動作のタイミング制御には、「待機時間(Elapsed Time)」Express VI を使用します。
2. 機能的グローバル変数デザインパターンを使用し、「ティックカウント」関数または「日付/時間を秒で取得」関数を使用して、タイミングサブ VI を作成します。

8. エラー

a. エラー処理

1. アプリケーションの開始時にエラークラスタを初期化します。
2. 関数、サブ VI、プロパティノードのエラー端子に配線します。
3. 必要に応じて、複数ソースからのエラーを結合します。

LabVIEW 開発者認定 (CLD) 認定プログラムと試験の概要

4. 「一般エラー処理 (General Error Handler)」VI でカスタムエラーコードを定義します。
 5. エラー発生時に実行するアクションを決定します。
 6. 可能であれば、エラーを修正する処理を行います。
 7. 致命的なエラーの発生時には、エラーワイヤや「エラー処理」ケースを直接使用して、実行中のすべてのループを確実に終了させます。
- b. エラーレポート
1. エラーや警告をレポートするには、「エラー処理」VI やエラーダイアログを使用します。

9. ドキュメント化

- a. ユーザインタフェース (フロントパネル)
1. ユーザインタフェースオブジェクトに、その機能を表わす名前のラベルを付けます。
 2. ユーザインタフェース制御器に、簡潔なヒントラベルを付けます。
 3. 必要に応じて、操作手順を提供します。
 4. クラスタや装飾体にグループ化された制御器に、適切なグループ名を付けます。
- b. ブロックダイアグラム
1. メイン VI およびサブ VI の全体的なアルゴリズムを、簡潔にドキュメント化します。
 2. 各プログラミングストラクチャを、その機能の簡潔な説明も含めてドキュメント化します。
 3. ワイヤにラベルを付け、ワイヤ上のデータとデータフローの方向を示します。
 4. 定数に、その機能を表わす名前のラベルを付けます。
 5. 自己ドキュメント化するプログラミング手法を使用します。
- c. VI プロパティ
1. メイン VI の「VI プロパティ」に、その VI の全体的な目的を簡潔にドキュメント化します。
 2. サブ VI の「VI プロパティ」に、その VI の全体的な目的、端子の説明、データタイプを簡潔にドキュメント化します。

LabVIEW 開発者認定 (CLD) 認定プログラムと試験の概要

10. テスト

a. コードとドキュメント化のレビュー

1. 上述の一連の要件と『LabVIEW ヘルプ』の「LabVIEW Style Checklist (英語)」トピックに対し、ユーザインタフェース、ブロックダイアグラム、プログラム設計が適合しているかどうかレビューします。
2. 上述の一連の要件と『LabVIEW ヘルプ』の「LabVIEW Style Checklist (英語)」トピックに対し、ドキュメント化が適合しているかどうかレビューします。

b. 機能

1. メイン VI を開いたときにすべてのサブ VI がリンクされ、壊れた矢印に変化しないことを確認します。
2. アプリケーションデータファイルへのアクセスに相対パスが使用されていることを確認します。
3. テストデータを使用して VI を実行し、各サブ VI が必要な出力を生成することを確認します。
4. 統合したアプリケーションが指定された機能を満たすかどうかテストします。
5. アプリケーションが CPU リソースの 100% を占有せず、ユーザインタフェースの操作から 100ms 以内に応答することを確認します。
6. アプリケーションの初期化時とシャットダウン時に、フロントパネルの制御器および表示器が適切に設定されているかどうかテストします。

c. エラー

1. ユーザインタフェースの制御器から意図しないデータが入力された場合、アプリケーションがエラーを生成するかどうかテストします。
2. エラーがサブ VI で適切に処理され、それぞれの発呼者に渡されるかどうかテストします。
3. エラーが適切に処理され、ユーザにレポートされるかどうかテストします。

LabVIEW 開発者認定 (CLD)
認定プログラムと試験の概要

LabVIEW 開発者認定試験 (CLD) 評価基準

本実技試験は、合計 40 点満点で以下の項目に関して採点されます。

- プログラミングスタイル： **15 点**
- 機能： **15 点**
- ドキュメント化： **10 点**
- 合格基準： 正解 75% **30 点**

採点時には、以下の点が考慮されます。

プログラミングスタイル：

- アプリケーションは、『LabVIEW ヘルプ』の「Development Guidelines (英語)」に沿っているか。
- VI は以下の点を満たしているか。
 - 可読性
 - 拡張性
 - 保守性
 - 不必要に複雑でない
- VI は適切な方法で作成されているか。
 - LabVIEW のフレームワークやデザインパターンを使用しているか。
 - ステートマシンが使用されているか。
 - VI は階層的な構造になっているか。
 - 複数回実行されるコードはサブ VI 化されているか。
 - ステートの定義に、タイプ定義された列挙型制御器が使用されているか。
- 不要な一時変数を使用していないか。
- フロントパネル制御器のデータのタイプ、範囲、形式/精度は適切か。
- データは、配列やクラスタなど適切なデータストラクチャにグループ化されているか。
- ストラクチャのネストが深くなりすぎていないか。2 レベルまでに抑えられているか。
- 初期化やクリーンアップ以外の目的でシーケンスストラクチャを使用していないか。
- ローカル変数やグローバル変数を使用しているか。
 - ローカル変数は、制御器の更新に使用される場合がある。
 - 競合状態を回避するため、グローバル変数は保護されているか。
- 表示器を更新するためにプロパティノードの値が使用されていないか。
- フロントパネルとブロックダイアグラムのレイアウトは適切か。

LabVIEW 開発者認定 (CLD) 認定プログラムと試験の概要

- ブロックダイアグラムは小さい領域に密集しすぎていないか。
- ワイヤの不要な屈折はないか。
- オブジェクトやワイヤが重なっていないか。
- ワイヤが、ストラクチャやストラクチャ枠の後ろにきていないか。
- エラー端子は配線されているか。
- リファレンスは適切に閉じられているか。
- メモリとパフォーマンスが最適化されているか。

機能:

- 実行ボタンは壊れていないか。
- VI は、仕様で挙げられた要件を適切に実行するか。
- ブール値の入力と出力の論理は適切か。
- VI は、指定された反応時間 (100ms) 以内にユーザ入力に反応するか。
- VI/サブ VI は CPU リソースの 100% を使用しないか。
- ファイル入出力は正しく実装されているか。
- エラー発生時にアプリケーションは停止するか。

ドキュメント化:

- ファイル→VI プロパティを選択すると、VI のドキュメントが表示されるか。
- サブ VI のドキュメント化は行われているか。
- ワイヤに適切なラベルが付けられているか。
- 機能はドキュメント化されているか。
 - ブロックダイアグラム
 - メインおよびネストされたストラクチャ
- フロントパネル上の制御器と表示器に、機能を表わすような名前が付けられているか。
- VI に、機能を表わすようなアイコンがデザインされているか。
- 定数はドキュメント化されているか。
- フロントパネル上の制御器に、関連するヒントラベルが付けられているか。
- トップレベル VI に、デフォルト以外のアイコンが使用されているか。
 - すべてのサブ VI に一貫したアイコンデザインが使用されているか。

LabVIEW 開発者認定 (CLD)
認定プログラムと試験の概要

LabVIEW 開発者認定試験 (CLD) 準備資料

以下の試験準備資料もご活用ください。

- [ナショナルインスツルメンツ LabVIEW 中級 I および II コース](#):
 - 講師によるトレーニングコース
 - 自習形式トレーニング
- [ナショナルインスツルメンツ CLD 準備コース \(オンライン\)](#)
- CLD 実技試験サンプル問題
 - [LabVIEW 開発者認定試験 \(CLD\) サンプル問題—自動洗車機コントローラ](#)
 - [LabVIEW 開発者認定試験 \(CLD\) サンプル問題—セキュリティシステム](#)
 - [LabVIEW 開発者認定試験 \(CLD\) サンプル問題—信号機](#)