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。

1. LabVIEW 準開発者認定試験（CLAD）
2. LabVIEW 開発者認定試験（CLD）
3. LabVIEW 設計者認定試験（CLA）

各認定試験に合格することが、次のレベルの認定試験の受験資格となります。CLD は、CLAD 認定者のみが受験できます。CLA は、CLD 認定者のみが受験できます。これらの受験資格は、両試験において例外なく適用されます。

LabVIEW 準開発者認定試験（CLAD）は、LabVIEW 開発システムの基礎機能全般に関する理解度、およびその知識を適用した実技（小規模な LabVIEW モジュールの開発、デバッグ、および保守）の水準を認定する試験です。CLAD は、6～9 か月程度の LabVIEW 開発システムの使用経験を目安としています。

CLD は、中～大規模の LabVIEW アプリケーションの開発、デバッグ、拡張、保守の実技水準を認定する試験です。CLD は、1 年から 1 年半程度の中～大規模の LabVIEW アプリケーション開発経験をお持ちのプロフェッショナルの方を対象としています。

LabVIEW 設計者認定試験（CLA）は、複数の開発者による開発環境における LabVIEW アプリケーションの設計能力を認定する試験です。プロジェクト仕様を基に LabVIEW コンポーネントを開発するための技術力やソフトフェア開発能力に加え、プロジェクト管理ツールや構成管理ツールを効率的に活用したプロジェクト管理能力も問われます。CLA は、2 年以上の中～大規模の LabVIEW アプリケーション開発経験をお持ちのプロフェッショナルの方を対象としています。

試験の概要

試験会場のコンピュータには、LabVIEW 開発システム 2011 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。 システム環境をカスタマイズする必要がある場合は試験監督者にお申し出ください。試験を開始する準備が整ってから、試験問題を配布いたします。
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のでご了承ください。 LabVIEW 開発システムの機能の詳細については、LabVIEW 開発システムを参照してください。

試験時間: 4 時間
試験形式: 実技試験（アプリケーション開発）
合格基準: 70%

本試験は、LabVIEW を用いた計測/オートメーションアプリケーション開発における、問題解決能力、理解度、および経験を認定するためのものです。本試験は、ソフトウェア開発のみを対象としており、ハードウェアは使用しません。

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）は、試験中にご使用いただけます。 外部で作成された VI や資料などはご使用いただけません。

アプリケーションを作成する際には試験問題と共に配布された USB メモリスティックに保存された VI および制御器を必ず使用してください。メイン VI やその他のコンポーネントの名前、またはフォルダ階層は、変更しないでください。作成した VI および制御器は、すべて既存のフォルダ構造に保存してください。

アプリケーションの要件書が提供されます。要件書には、アプリケーションの一般要件および技術要件が記述されています。試験用紙を留めたホッチキスを外したり、試験問題や回答をコピーや複製することは**禁止されています**。これに違反した受験者は不合格となります。

試験終了後、配布された USB メモリスティックに解答を保存してください。解答が保存されたことを確認し、USB メモリスティックを試験監督官に提出してください。

試験内容

1. 設計概念
2. ユーザインタフェース（フロントパネル）の設計
3. ブロックダイアグラムの設計
4. プログラミング手法
5. サブ VI の設計
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

試験内容（概要）

トピック	項目
1. 設計概念	a. モジュール性、拡張性、可読性、保守性 b. 結合度、凝集度 c. 階層設計 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	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 i. 静的/動的 ii. 初期化時、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	a. データフロー b. 可読性の向上
4. プログラミング手法	a. データ要素 b. 関数とサブ VI c. プログラミング構造 d. データ構造 e. リファレンス、プロパティノード
5. サブ VI の設計	a. モジュール性、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、アイコン
6. デザインパターンの選択	a. 拡張性、保守性 b. 応答性、ブロック防止 c. デザインパターン i. シンプルステートマシーン 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	a. タイミング関数 b. タイミングメカニズム 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、機能的グローバル関数
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 d. エラー

LabVIEW 開発者認定試験（CLD）試験内容の詳細

1. 設計概念

a. モジュール性、拡張性、可読性、保守性

1. 以下の条件を満たす LabVIEW アプリケーション（VI）を開発する
   a) モジュール性 ー VI の機能がモジュールまたはサブ VI に分割されている
   b) 拡張性 ー ユーザインタフェースやブロックダイアグラムをほとんど変更せずに、より大規模なデータセットやより多くのプログラム状態に対応できる
   c) 可読性 ー 適切なドキュメント化およびプログラミングスタイルにより、第三者が VI を理解しやすい
   d) 保守性 ー モジュールやアプリケーションの本来の目的を変更せずに、VI を変更できる

b. 結合度、凝集度

1. 以下の要件を満たす LabVIEW モジュールを作成する
   a) 高凝集性 ー モジュールの目的が明確に定義され、公開されている
   b) 疎結合性 ー 機能を完成/補完するための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. 上述の方法を使用し、論理的階層により構成された LabVIEW VI を作成する

d. ファイル構造

1. VIのファイル構造は、ソフトウェアの階層構造を反映するように構成する
2. アプリケーションのフォルダを作成し、分かりやすい名前を付ける
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する
4. サブVIおよび制御器用に独立したフォルダを作成する

2. ユーザインタフェース（フロントパネル）の設計

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する
2. 色付けが必要な場合は、薄い色を使用し、鮮やかな色は避ける

3. 背景およびユーザー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 スタイルチェックリスト」トピックのガイドラインを使用する

b. オブジェクトのグループ化と整列
   1. 論理的に関連するユーザインタフェースオブジェクトは、配列、クラスタ、装飾体などを使用してグループ化する
   2. レイアウトに統一感を持たせるように、オブジェクトおよびそのラベルを配置する

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選択する

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する
   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することにより、アプリケーションの拡張性を向上させる

e. 状態管理
   1. オブジェクトの「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することにより、制御器の値や属性を設定する
   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う

f. アイコンのデザイン
   1. メイン VI（トップレベル VI）およびサブ VI のアイコンは、アプリケーションやモジュールの機能を表すようにデザインする
   2. メイン VI およびサブ VI の間で一貫性のあるアイコンデザインスキームを使用する

3. ブロックダイアグラムの設計

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、データフローを確立する
   2. エラー端子のない 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する

b. 可読性の向上
   1. 1024 x 768 の画像解像度に合わせてブロックダイアグラムを作成する
   2. ユーザが複数方向にスクロールしなくてもいいように、ブロックダイアグラムの大きさを制限する
   3. VI および関数が狭いスペースに密集し過ぎないように、等間隔に配置する。
   4. 一貫したスキームを使用し、VI および関数を等間隔に配置する
   5. ワイヤの屈折を避け、直線になるようにする
   6. 正しい端子に接続されていることが確認できるようにワイヤを配線する
   7. 左から右へ、上から下へとデータが流れるように、VI や関数を配線する
   8. ストラクチャやストラクチャ枠の下にワイヤが隠れないようにする
   9. ストラクチャ枠上にトンネルが重複しないようにする
   10. ブロックダイアグラムのセクションを区別するために色を使用することは避ける。システムカラーは必要な場合のみに使用する。

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する
   2. 制御器から直接読み取る。ローカル/グローバル変数やプロパティノードの使用は避ける。
   3. 表示器を直接更新する。ローカル/グローバル変数やプロパティノードの使用は避ける。
   4. ローカル変数は、制御器の更新に使用する。
   5. プロパティノードは、制御器および表示器の属性の設定に使用する。
   6. リファレンスは、サブ VI がフロントパネル上の制御器または表示器に影響する場合にのみ使用する。
   7. 競合状態を避けるため、ローカル/グローバル変数へのアクセスを制限する。
   8. タイプ定義データ要素を使用することにより、すべてのインスタンスで同じタイプを維持し、拡張性を向上する
   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、定数を使用する

b. 関数とサブ VI
   1. 関数の機能を最大限に利用する。なるべく少数の関数でタスクを実行する
   2. 入力でのデータ強制を避ける
   3. 可能な限りエラー端子を使用する
   4. 明示的に開いたリファレンスを閉じる

c. プログラミング構造
   1. VI に適切なプログラミングストラクチャを選択する
   2. 必要に応じてシフトレジスタを初期化する。シフトレジスタを初期化しない場合の影響を考慮する
   3. ストラクチャのネストの深さは、2 レベルに留める
   4. ワイヤを使用できない場所でデータフローを確立する場合は、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する
   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ことを避ける
   6. 拡張性を高めるため、可能な限りシーケンスストラクチャの代わりにケースストラクチャを使用する
   7. ストラクチャ枠のトンネルでデフォルト値を使用することは避ける
   8. ユーザインタフェースイベントを処理するには、イベントストラクチャを使用する
   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する

d. データ構造
   1. 論理的に関連するデータ要素は、適切なデータ構造にグループ化する
   2. ブロックダイアグラム上に作業用のデータ構造を作成して、ユーザインタフェースで操作が不要なデータやステート情報をグループ化して操作する

e. リファレンス、プロパティノード（リファレンスを取得する/閉じる）
   1. 同一 VI 内のオブジェクト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する
   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する
   3. プロパティノードの機能を認識し、適切に使用する
   4. 明示的に開いたリファレンスを閉じる

5. サブ VI の設計

a. 凝集性とモジュール性
   1. 明確に定義された単一機能を実行するようにサブVIを設計する
   2. サブVIのタスクを補う関数/サブVIを呼び出す

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する
   2. エラー入力クラスタとエラー出力クラスタを追加し、コネクタペーンに配線する
   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」ケースのセレクタ端子に配線する
   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する
   5. すべてのサブVIコードを「エラーなし」ケースに配置し、関数、サブVI、プロパティノードのエラー端子にエラーワイヤを確実に接続する
   6. さまざまなソースからのエラーを結合し、そのエラーをエラー出力端子を介して呼び出し側VIに渡す
   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する

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 スタイルチェックリスト」トピックのガイドラインを使用する
   2. 端子の種類として必須、推奨、任意を指定する
   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する
   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する

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、メインVIに最適なデザインパターンを選択する
   2. サブVIの必要機能を実現するために最適なデザインパターンを選択する
   3. 拡張性や保守性を制限しないアーキテクチャを使用する
   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する

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100 ms以内に応答し、適切な頻度で表示器を更新できるデザインパターンを使用する
   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する

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する
   2. 入力ステートに基づいて1つまたは複数の関数やサブVIを実行する必要のあるサブ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する
   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する
   4. 1つまたは複数のステートを保存するには、キューメッセージハンドラデザインパターンを使用する
   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は、生産者/消費者（データ）デザインパターンを選択する
   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つまたは複数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は、生産者/消費者（イベント）デザインパターンを選択する
   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を選択する

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できるようにするために最適なタイミング関数を使用する
   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適なメカニズムを決定する

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できるようにするには「待機」関数を使用する
   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」関数を使用する
   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」関数を使用することは避ける
   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」関数および「日付/時間を秒で取得」関数を使用する
   5. 「ティックカウント」関数を使用する場合は、所要時間を考慮する
   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する

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」ケースを使用する
   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ア関数のタイムアウト入力を使用する
   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する

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」Express VI を使用する
   2. 機能的グローバル変数で「ティックカウント」関数または「日付/時間を秒で取得」関数を使用することにより、タイミングサブ VI を作成する

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する
   2. 関数、サブ VI、プロパティノードのエラー端子に配線する
   3. 必要に応じて、複数ソースからのエラーを結合する
   4. 「一般エラー処理」VI でカスタムエラーコードを定義する
   5. エラー発生時に実行するアクションを決定する

6. 可能な場合はエラーを修正する処理を行う
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、実行中のすべてのループを確実に終了させる

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する

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、その機能を表す名前のラベルを付ける
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る
3. 必要に応じて操作手順を提供する
4. クラスタや装飾体にグループ化された制御器には、適切なグループ名を付ける

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する
2. 各プログラミングストラクチャの機能を簡潔にドキュメント化する
3. ワイヤ上のデータとデータフローの方向が分かるように、ワイヤにラベルを付ける
4. 定数に、その機能を表す名前のラベルを付ける
5. 自己ドキュメント化するプログラミング手法を使用する

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する
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する

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 スタイルチェックリスト」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する
2. 上述の一連の要件と『LabVIEW ヘルプ』の「LabVIEW スタイルチェックリスト」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する

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する
   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する
   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する
   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する
   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 100 ms 以内に応答することを確認する
   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する

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する
   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する
   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する

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：（正答率 70%） **28 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「アプリケーション開発と設計ガイドライン」トピックに沿っているか。
- 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。
    - 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。
- ワイヤが不要に屈折していないか。
- 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隠れていないか。
- エラー端子が配線されているか。
- リファレンスが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、仕様に定義さ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（100 ms）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 はドキュメント化されているか。
- 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、機能を表わす名前が付けられているか。
- VI に、機能を表わす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 開発者認定試験（CLD）準備資料

以下の試験準備資料をご活用ください。また、一部のトピックの復習には、LabVIEW 準開発者認定試験（CLAD）プログラム概要に掲載されている準備資料をご活用ください。

LabVIEW 開発者認定試験（CLD）準備：

- LabVIEW 開発者認定試験（CLD）準備用リソースキット（開発者認定試験プログラム概要、サンプルテスト、Web イベントへのリンクを掲載）

ナショナルインスツルメンツの講義/自習形式コース：

- LabVIEW 実践集中コース 1
- LabVIEW 実践集中コース 2
- LabVIEW 実践集中コース 3
- LabVIEW パフォーマンス（推奨コースには含まれていませんが、認定試験準備に非常に役立ちます）

LabVIEW 開発者認定試験（CLD）

以下の表は、LabVIEW 開発者認定試験（CLD）で課題として提供される可能性のあるシナリオを示したものです。これは、試験の概要を説明するためのものであり、実際の試験内容はここに記したものと異なる場合があります。

試験のシナリオ	**説明**
コーヒーメーカー	コーヒーメーカーは、お湯、コーヒー、およびカフェオレを作るための材料の保管、豆の粉砕、飲料の作成および抽出などの操作をシミュレートします。
電子レンジ	電子レンジは、手動メニューまたは自動レシピメニューをシミュレートします。
セキュリティシステム	セキュリティシステムは、複数のセキュリティゾーンの監視、アラームの設定/解除、不正侵入、バイパスなどの操作をシミュレートします。
空調システム	空調システムは、暖房、冷房、換気の自動操作をシミュレートします。
トレッドミル	トレッドミルは、速度、傾度、時間、走行距離の手動/自動操作をシミュレートします。
自動販売機	自動販売機は、商品の保管、購入、販売操作をシミュレートします（米国通貨を使用）。