2015.04.17 V13.4

ガイオ･テクノロジー 定期セミナーCM1

# モジュール単体･カバレッジテスト 入門コース

ガイオ･テクノロジー(株)　営業部

# アジェンダ

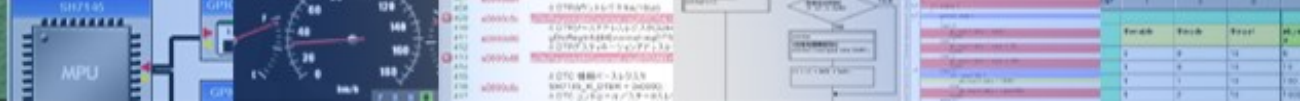
## ■【ガイオ・テクノロジー 定期セミナーCM1 概要】

ソフト品質改善のための、モジュール単体テスト、カバレッジテスト手法の入門コースです。
専用シミュレータ「カバレッジマスターwinAMS」を実際に操作して、モジュール単体自動テストを体験頂けます。

## ■【アジェンダ】

- 組み込みソフト品質向上をとりまく業界の動向と課題について
- 品質向上のための解決策について
- 「マイコンシミュレータ」を用いたソフト品質改善手法について
- 導入事例紹介
- モジュール単体テスト、カバレッジテストの概要と方法
- 　実習（1）：サンプルを使用した関数単体テストの基礎
- 　実習（2）：ポインタ変数を持つ関数の単体テスト
- 　実習（3）：STUB機能を使用したサブ関数の入れ替え
- 　実習（4）：CasePlayer2と連携した テストデータ作成機能を使う
- Q＆A

追加実習　：　MC/DCカバレッジ測定のための測定環境構築方法
（通常のコースには含まれません）



# 組み込みソフト品質向上への課題

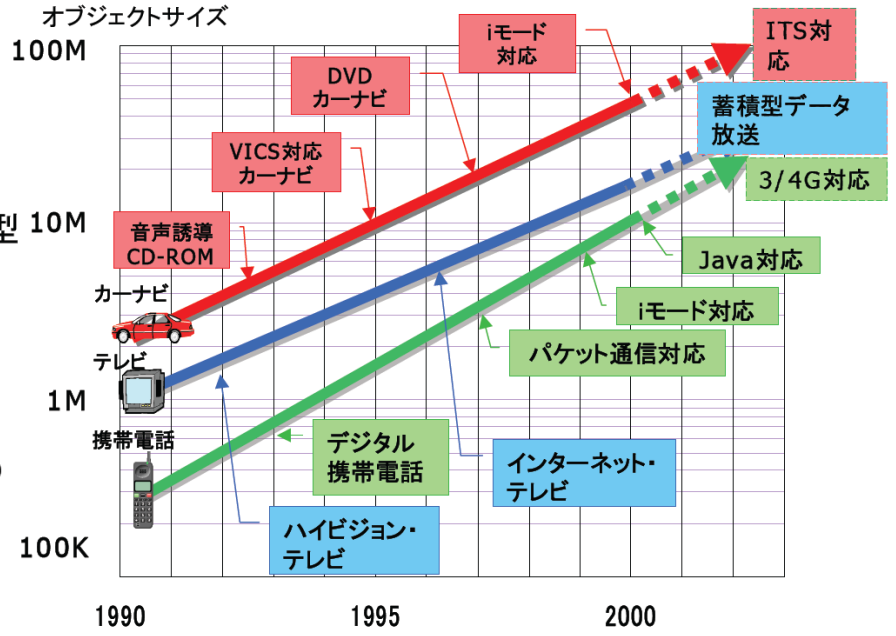
# 組み込みソフトウェア開発の現状

- **組み込みソフトウェアは大規模化、複雑化している**
  - 累乗的に大規模化、複雑化するソフトウェア規模
  - ソフトウェア規模に比例して増大する不具合件数

**市場不具合や開発リスクが激増!!**

規模例（SLOC*）

- 携帯電話
  - 500万
- 通信機能搭載型カーナビ
  - 300万
- 薄型テレビ
  - 60万
- HDD内蔵DVDレコーダ
  - 100万



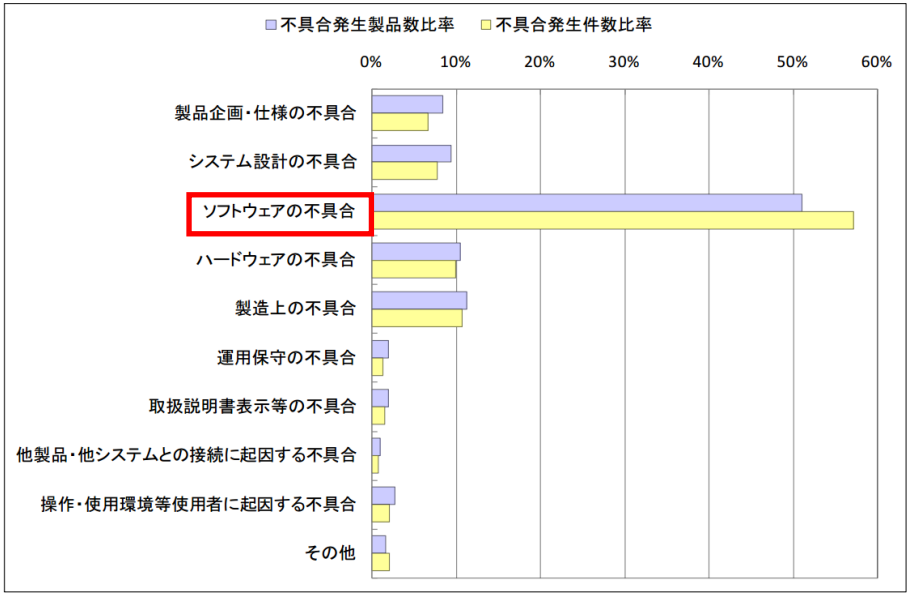

* SLOC: Source Lines of Code

出典：日経エレクトロニクス 2000 9-11(no.778)をベースに追加、修正。

# ソフト品質改善に対する意識・要求

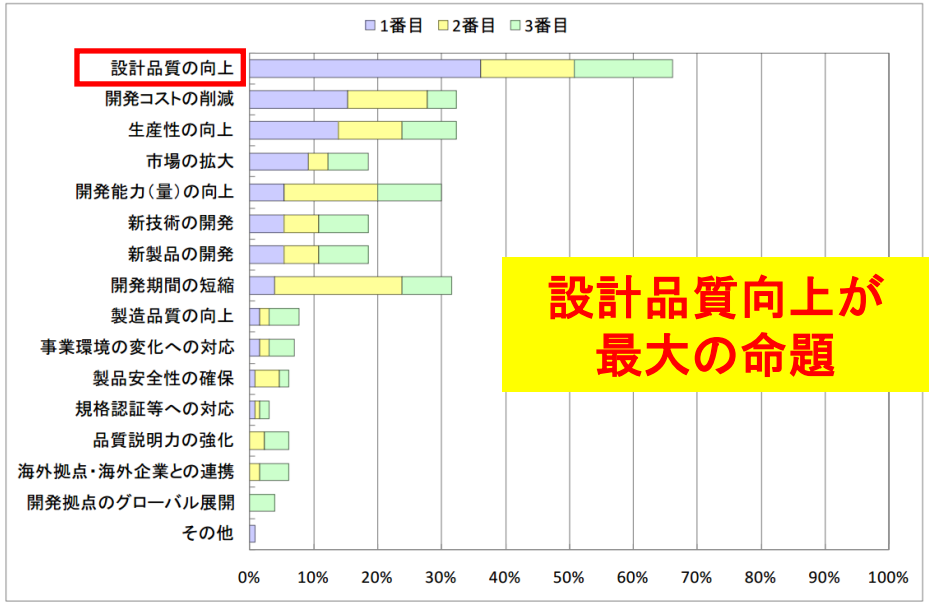
■組込みソフトが原因となる品質問題が非常に多い
■「設計品質の向上」が企業における最優先課題となっている

Q5-2 原因別の不具合発生製品数比率（N=74）と不具合発生件数比率（N=73）




製品出荷後の不具合の原因

Q2-6-I 組込み系ソフトウェア開発の課題




設計品質向上が最大の命題

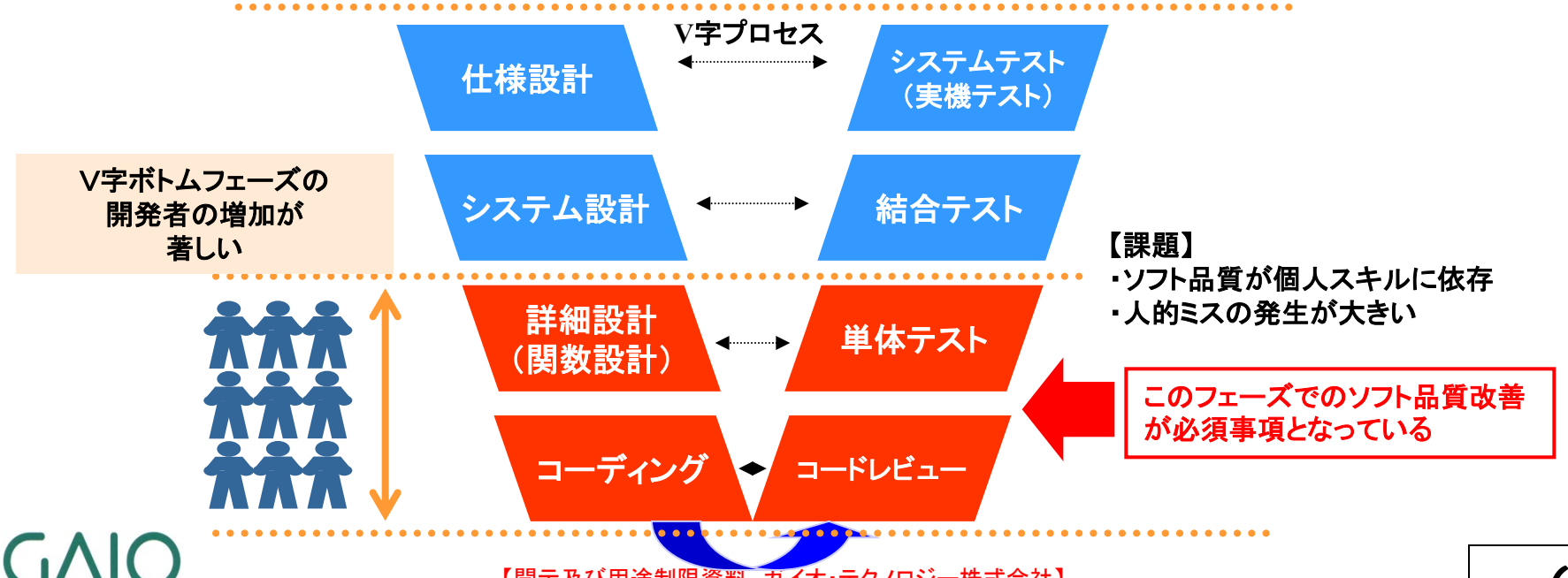
事業責任者層の課題意識

IPA 独立行政法人情報処理推進機構 Information-technology Promotion Agency, Japan

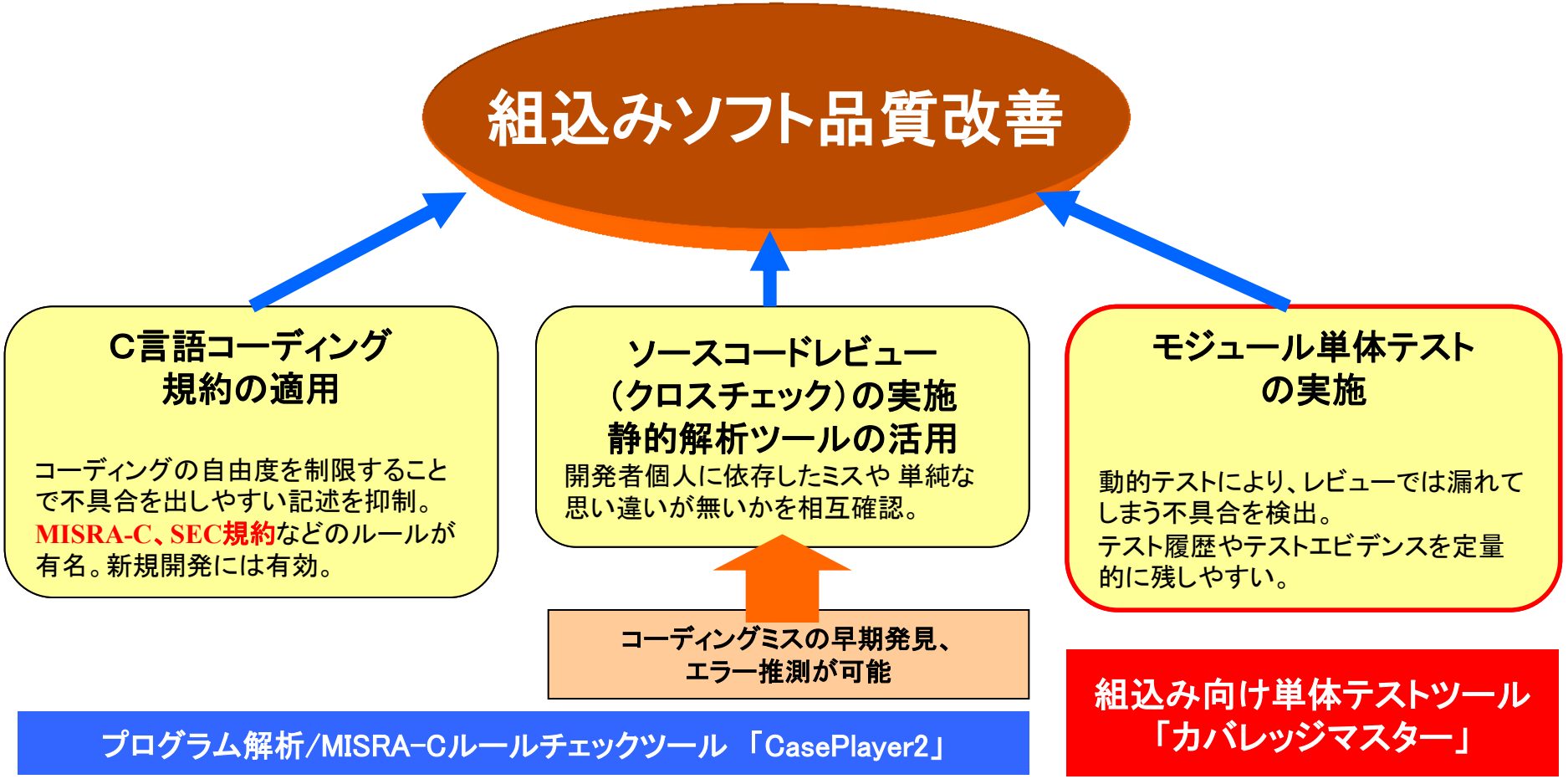
2012 年度「ソフトウェア産業の実態把握に関する調査」調査報告書

# 組込みソフトウェア開発の課題

- **ソフト開発者の人数を増員により ソフト品質にばらつきが出てしまう**
  - 組込み開発経験の浅い技術者も多く、ソフト品質が保てなくなる
  - システム全体のソフトウエアの全てを見通すことは不可能になっている
- **コーディング後のソフト品質確認のために「単体テスト」が重要となる**
  - コードレビュー、単体テストフェーズへの対策が必須



# 組込みソフト品質改善の方策

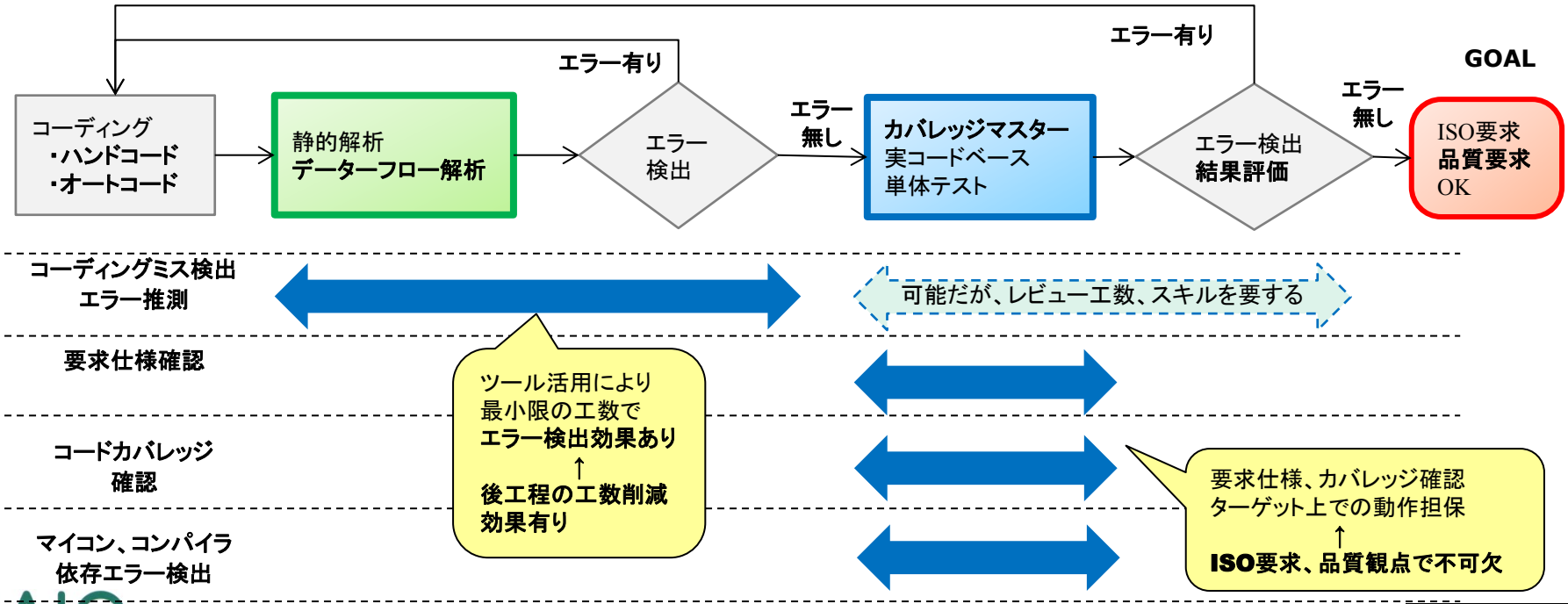


# （参考）静的解析と動的テストの活用

## ■ 静的解析エラー推測と 動的単体テストによるエラー検出の棲み分け

静的解析：　静的コードチェック、データフロー（ランタイム）解析、エラー推測
　　最小限のテスト工数で、コーディングミス、エラー推測が可能

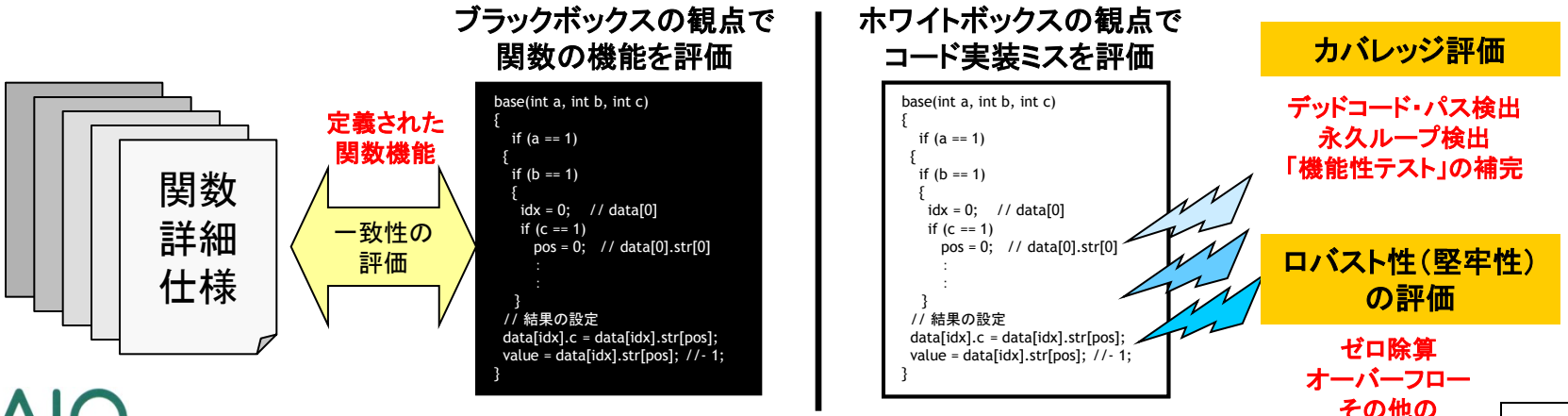
動的検証：　単体テストによる期待値評価＋コードカバレッジ計測
　　要求仕様テスト、ターゲット依存の問題点検出が可能 → 機能安全（ISO26262）対応に必須



# 潜在バグを無くし 信頼性を向上する
# モジュール単体テスト

# 単体テストとは？

- **ブラックボックスの観点で 関数の機能性を評価**
  - 各関数が仕様通りに作られていることを確認すること
  - ブラックボックスの観点で、関数仕様通りに設計されているかをテスト
  - 詳細設計から設定した引数/外部変数への入力値に対して 出力変数が正しいか？
- **ホワイトボックスの観点で コーディング時に作り込んだミスを検出**
  - カバレッジの指針を使用して 実装コード構造の間違いを検出
  - 不要コード、デッドコードの有無の確認
  - ゼロ割、NULLアクセスへの回避コードの有無、オーバーフローの発生の有無を確認



# （参考）ISTQBに定義された単体テスト技法

## ■ ISTQBの定義から引用：　単体テストの設計技法

- **仕様ベース テスト**
  関数仕様書を基にテスト項目を作成
  ブラックボックステスト / 関数への入力に対する出力を評価
- **構造ベース テスト**
  ソースコードの全ての構造を網羅して、これがすべて「動くこと」を確認するテスト
  カバレッジの指標（C0,C1,MC/DC）により、これを満たすテストを行う

- **欠陥ベース テスト**
  予測される潜在的な問題を中心にテストを設計する
- **経験ベース テスト**
  過去の問題経験からテストを設計する

# （参考）仕様ベーステスト/仕様網羅のテスト項目設計

## ■ 関数仕様を基にテスト項目を導出する

- 関数仕様を網羅するテストケースを設計する

例：　AddFunc()　関数仕様

【関数機能】

入力された2つの引数InA、InBの加算値を求め、戻り値として出力する

【入出力仕様】

入力

引数：　InA：　unsigned char型、有効レンジ　150以下

引数：　InB：　unsigned char型、有効レンジ　150以下

出力

戻り値：　unsigned char型

【例外条件】

InA、InBのいずれかが有効レンジ外の場合は、255（0xFF）を出力して終了

加算値が200を超える場合は、200を出力して終了

【API】

unsigned char AddFunc(unsigned char InA, unsigned char InB)

導出

**単体テスト設計の手法**

- 境界値分析
- デシジョンテーブルテスト
- 同値分割法
- 状態遷移テスト
- ユースケーステスト

入力テストケース例

| inA | | inB | |
|---|---|---|---|
| 149 | 境界条件 | 149 | 境界条件 |
| 150 | | 150 | |
| 151 | | 151 | |
| 255 | 最大値 | 255 | 最大値 |
| 0 | 最小値 | 0 | 最小値 |
| 75 | 代表値 | 75 | 代表値 |

## （参考）自動車機能安全規格　ISO 26262-6:　9 Software unit testing

Table 10 — Methods for software unit testing

| | Methods | ASIL A | ASIL B | ASIL C | ASIL D |
|---|---|---|---|---|---|
| 1a | Requirements-based test[a] | ++ | ++ | ++ | ++ |
| 1b | Interface test | ++ | ++ | ++ | ++ |
| 1c | Fault injection test[b] | + | + | + | ++ |
| 1d | Resource usage test[c] | + | + | + | ++ |
| 1e | Back-to-back comparison test between model and code, if applicable[d] | + | + | ++ | ++ |

Table 11 — Methods for deriving test cases for software unit testing

| | Methods | ASIL A | ASIL B | ASIL C | ASIL D |
|---|---|---|---|---|---|
| 1a | Analysis of requirements | ++ | ++ | ++ | ++ |
| 1b | Generation and analysis of equivalence classes[a] | + | ++ | ++ | ++ |
| 1c | Analysis of boundary values[b] | + | ++ | ++ | ++ |
| 1d | Error guessing[c] | + | + | + | + |

# （参考）構造ベーステスト/カバレッジ確認

## ■ 仕様を基に設計したテストケースでコード構造が網羅されるかを評価

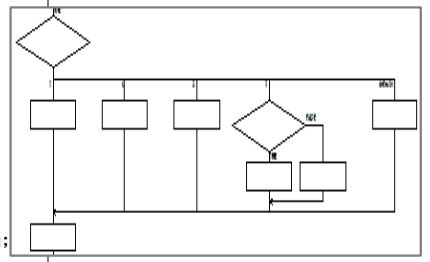
- カバレッジ（C0、C1、MC/DC）が100%になるかを確認
  - ➢ 100%の場合：　不要コード、デッドコードが無いことが確認できる
  - ➢ 未達の場合：　テストケースの不足か、または不要コード、デッドコードかを評価

**仕様を網羅するテストケース**

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | 引数 | gb_a | gb_b | gb_c | gb_d | gb_out | 戻り値 |
| 2 | 1 | 11 | 21 | 31 | 31 | 5 | 0 |
| 3 | 1 | 10 | 21 | 31 | 31 | 5 | 0 |
| 4 | 1 | 11 | 20 | 31 | 31 | 5 | 0 |
| 5 | 2 | 10 | 21 | 31 | 31 | 5 | 0 |
| 6 | 3 | 10 | 21 | 31 | 31 | 5 | 0 |
| 7 | 4 | 10 | 21 | 31 | 31 | 5 | 0 |
| 8 | 4 | 10 | 21 | 30 | 31 | 4 | 0 |

**コード構造を網羅するかを評価**

```
if( enable )
{
    switch( mode )
    {
    case 0:
        gb_result.data = input;
        break;
    case 1:
        gb_result.data = input * 10;
        break;
    case 2:
        gb_result.data = input * 100;
        break;
    case 3:
        if( input>100 )
            gb_result.data = 10000;
        else
            gb_result.data = input*100;
        break;
    default:
        gb_result.data = -1;
```



## （参考）自動車機能安全規格　ISO 26262-6:　9 Software unit testing

Table 12 — Structural coverage metrics at the software unit level

| Methods | | ASIL A | ASIL B | ASIL C | ASIL D |
|---|---|---|---|---|---|
| 1a | Statement coverage | ++ | ++ | + | + |
| 1b | Branch coverage | + | ++ | ++ | ++ |
| 1c | MC/DC (Modified Condition/Decision Coverage) | + | + | + | ++ |

# 単体テストの工程

## ■ 単体テスト設計指針の作成

- 単体テストがテスト担当者のスキルに依存しないように指針を設定
  ※次頁に参考例

## ■ 単体テスト設計

- 仕様ベースによるテストケース設計
  - 境界値分析、同値分割、状態遷移　など単体テストの導出手法を使用
- 構造ベースによるカバレッジ網羅のテスト設計
  - C0、C1、MC/DCなどカバレッジ指針を満たすテストを行う
- 欠陥ベース、経験ベースによるテストケースの追加

## ■ 単体テスト実行

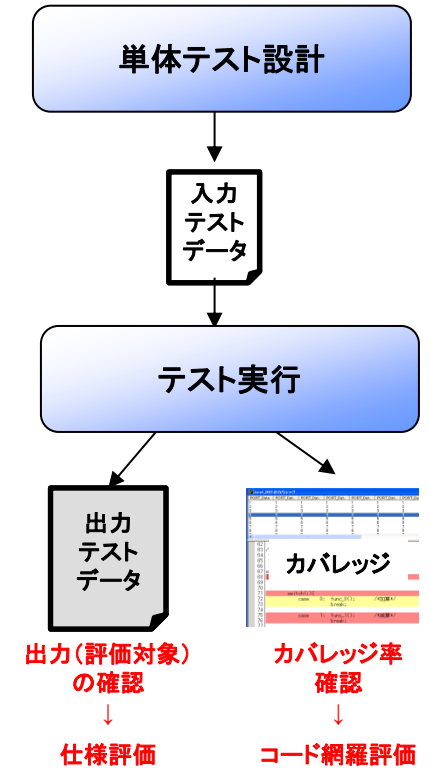
- 作成したテストデータを関数に与えて実行

## ■ 実行結果の評価

- 期待値との一致を確認、カバレッジ評価

ツールで
自動化が可能

**モジュール単体テストの概要**



# （参考）単体テストには作業フローのマニュアル化が重要

## ■ 単体テストを定着運用するためには・・・

- 単体テストの作業フローをマニュアル化する
- 担当者のスキルに依存しないテストが可能な様にテストを標準化する
  - ✓ 必ず行わなければならないテスト項目（必須項目）を明確に定義する
  - ✓ テスト設計に可能な限り担当者の判断が入らないようにテスト指針を設定する

4．方針
・関数単位で単体テストを実施する。
・テスト対象関数は例外を除き修正はしない。
・期待値については対象となるもののみ判定を行う。
・カバレッジは C1 レベルとする。

6．1テストデータ作成基準
・テストデータ作成する上での基準を以下に示す.
一般値：該当する変数の最大値、最小値、閾値、閾値±１、0、-1 以外の値。基本は 11 とする。
初期値：全ての期待値に入力データとして与える値。基本は 10 とし、他の入力データ及び他の出力データと
ならないデータを初期値として与える。
・不等号条件の場合は最大値、最小値、閾値、閾値±１、0、-1 (0xFFFFFFFF) の検証を行う。
※MPU の signed 及び unsigned によって動作に不正がないか確認する為。
・パステストは C1 網羅で行う。従って、例えば、
if (a==1 || b==1 || c==1) { ... }
のようなソースの場合、if 条件節の中の c==1 は、全く実行されない場合がある。


※単体テスト代行サービスにおける 指針（ガイドライン）の一例

6．1．2　分岐条件が数値、単独、不等号条件の場合
・ 分岐条件が数値、単独、不等号条件の場合は、最大、最小、閾値、閾値±１、0、-1 の検証を行う。

```
Test(int a, int b, int c)
	{
		if(a < 1)
		{
			out = 1;		//期待値　1
		}
		else
		{
			out = 0;		//期待値　0
		}
	}
```

例．分岐条件が数値、単独、不等号条件のソース

| 入力データ | | 期待値 |
|---|---|---|
| @a | @out | out |
| 0x7FFFFFFF | 10 | 0 |
| 0x80000000 | 10 | 1 |
| 1 | 10 | 0 |
| 2 | 10 | 0 |
| 0 | 10 | 1 |
| 0xFFFFFFFF | 10 | 1 |

例．分岐条件が数値、単独、不等号条件の場合の入力データ

# （参考）コントロールカバレッジ

## ■ コントロール（実行制御に対する）カバレッジ

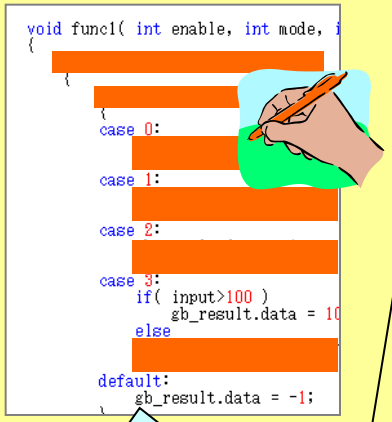
### C0：ステートメント（命令）カバレッジ

【指針】全ての実行文を1度実行すればOK

【保証できる内容】

・コーディングしたが実行されない実行文はない

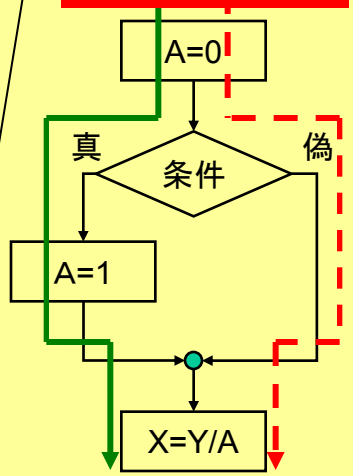
・存在してはいけない実行文はない




C0カバレッジ：

ソース行をマークして、全て塗りつぶす

C0では十分でない例



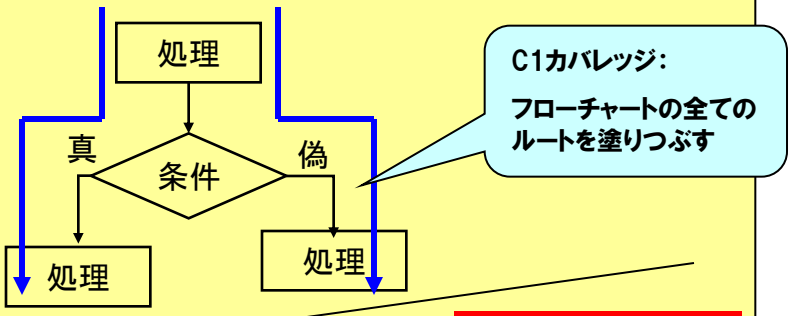

赤のパスには実行文がないので、C0ではテストされない

### C1：ブランチ（分岐）カバレッジ

【指針】全ての条件文の真/偽を実行すればOK

【保証できる内容】

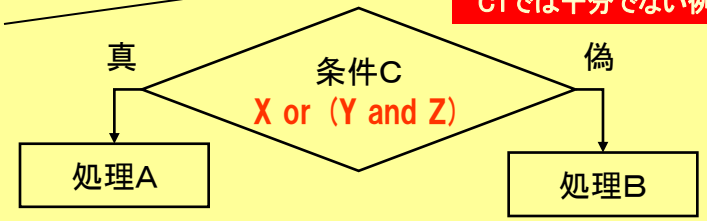
・存在してはいけない実行パスは存在しない




C1カバレッジ：

フローチャートの全てのルートを塗りつぶす

C1では十分でない例




複合条件では1つのTRUE/FALSEがテストされるのみ
全ての組合せがテストされる訳ではない

# （参考）コントロールカバレッジ（続）

## ■ コントロール（実行制御に対する）カバレッジ　（続き）

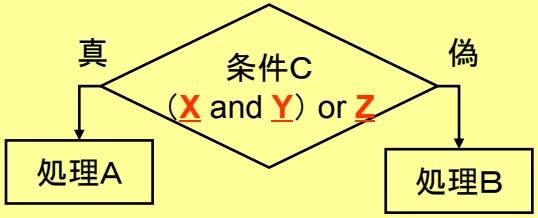
MC/DC:
全ての条件式のTRUE/FALSEの効果を確認する

### C2：コンディション（条件）カバレッジ

【指針】全ての条件文の条件式の組み合わせを1度実行すればOK

【保証できる内容】

・存在してはいけない条件判定式の要素がないこと




C2カバレッジ:
複合条件の全数組合せをテストする

全ての条件文の条件式の組み合わせを実行

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 条件式 X | F | F | F | F | T | T | T | T |
| 条件式 Y | F | F | T | T | F | F | T | T |
| 条件式 Z | F | T | F | T | F | T | F | T |

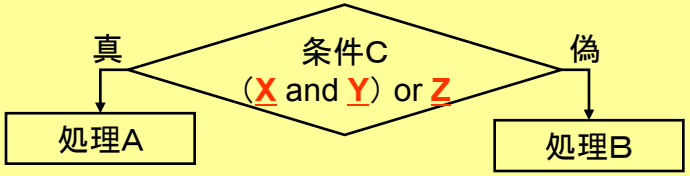
条件要素に対する網羅性は100%だが、
効果のないテストが含まれてしまう

### MC/DC：Modified Condition/Decision カバレッジ

【指針】全ての条件文の条件式の組み合わせのうち、意味のある組み合わせを全て1度実行すればOK

【保証できる内容】

・C2と同等の網羅内容（テストデータはC2より少ない）




他の条件式の真偽を固定して、1つの条件式の真偽を変化させたとき、全体の条件に変化があるものだけを実行する（↓色のついたデータ）

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 条件式 X | F | F | F | F | T | T | T | T |
| 条件式 Y | F | F | T | T | F | F | T | T |
| 条件式 Z | F | T | F | T | F | T | F | T |
| 論理 | F | T | F | T | F | T | T | T |

2つのデータ組み合わせで、各条件式を評価→ Z X Y

C2の網羅性を維持して、テストパターン数を抑えることができる

# 単体テストの効果例/実施例

# モジュール単体テストの効果例1

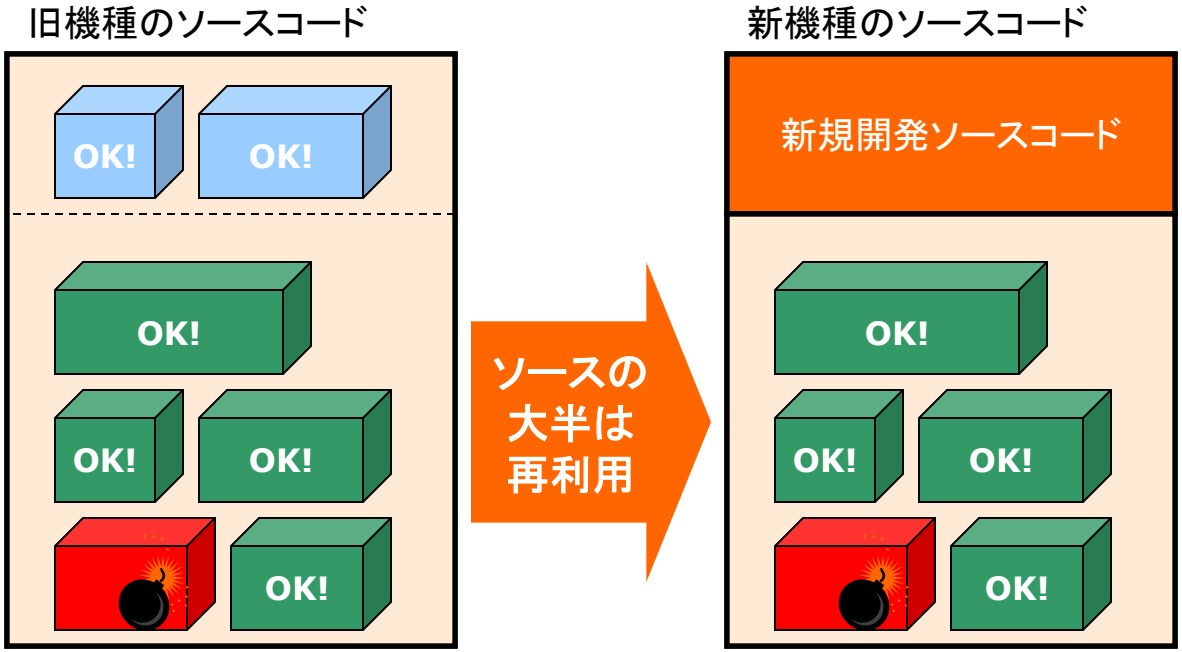
■ ソフト設計の上流工程での単体テストは 潜在バグを未然に防ぐ

- 上流工程での「ホワイトボックステスト」で「想定外」条件を検証
- 下流工程のテスト項目を減らし、問題要因の発見を容易にする

# モジュール単体テストの効果例2

■ **関数モジュールの再利用において 製品品質を確保可能にする**

- 単体テスト未実行の関数の再利用は 他の製品へ不具合を及ぼす
- モジュールの再利用においては 単体テストは必須となる




一般的な組み込み開発では、新規開発するソースは、少ない

ほとんどは、ソースを再利用して開発される

再利用するソースに 不具合が潜在していると・・

次の設計にも そのまま影響してしまう！

# モジュール単体テストの効果例3

■ テストの結果を定量的に図りやすく 品質の指標が明確になる

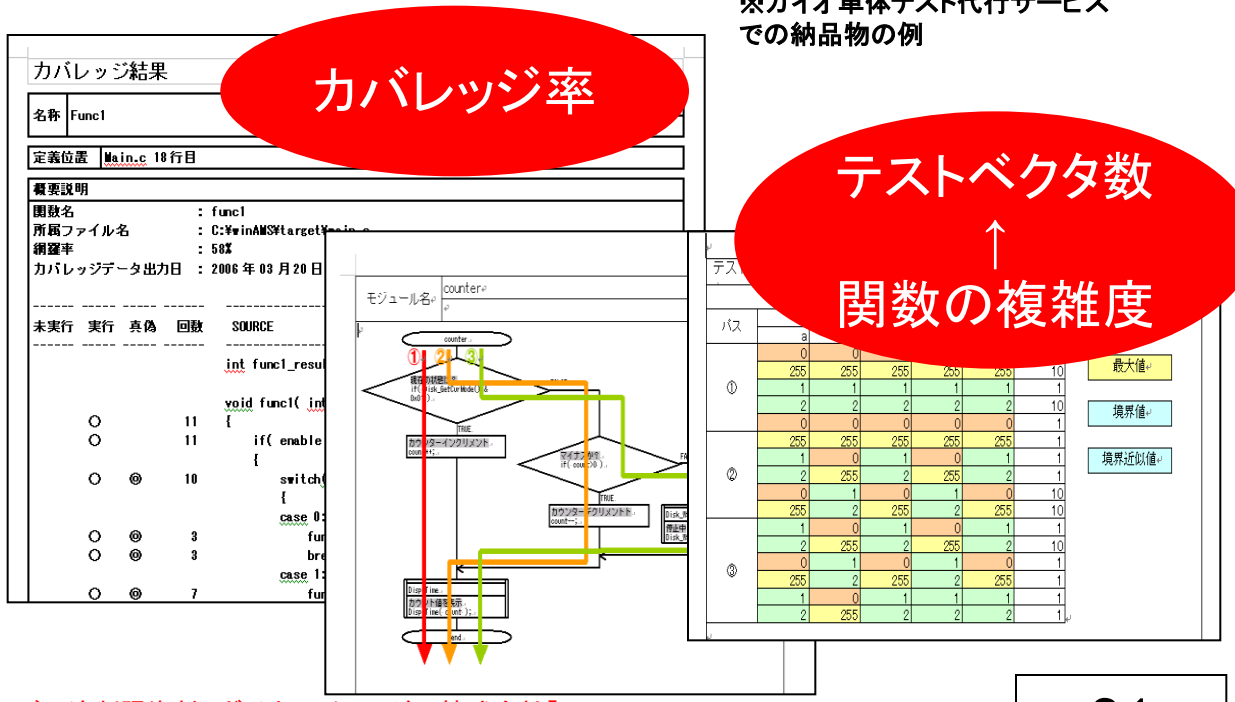
要素分析表： 変数に対してどの様な値を与えてテストしたかを示す

テストベクターの個数：

カバレッジ率： C1/MCDCなど、ソースコードのテスト網羅率を示す

| | | 入力要素 | | | | | |
|---|---|---|---|---|---|---|---|
| | | A | B | C | D | E | F |
| | | 第1引数 y | 第2引数 count | 第2引数 countが示すメモリデータ | disp_set2 の 復帰値 | disp_set2 の引数 count | グローバル変数 flag |
| 入力値 | 1 | 1 | short型の領域を示すアドレス | 0xaaaa | 6 | 0 | 0 |
| | 2 | 2 | | 0xffffe | 0 | 1 | 1 |
| | 3 | 5 | | 0xffff | | 0xaaaa | |
| | 4 | 9 | | | | 0xffffe | |
| | 5 | 10 | | | | 0xffff | |
| | 6 | 0 | | | | | |
| | 7 | | | | | | |
| | 8 | -5 | | | | | |
| | 9 | 15 | | | | | |

要素分析表




カバレッジ率

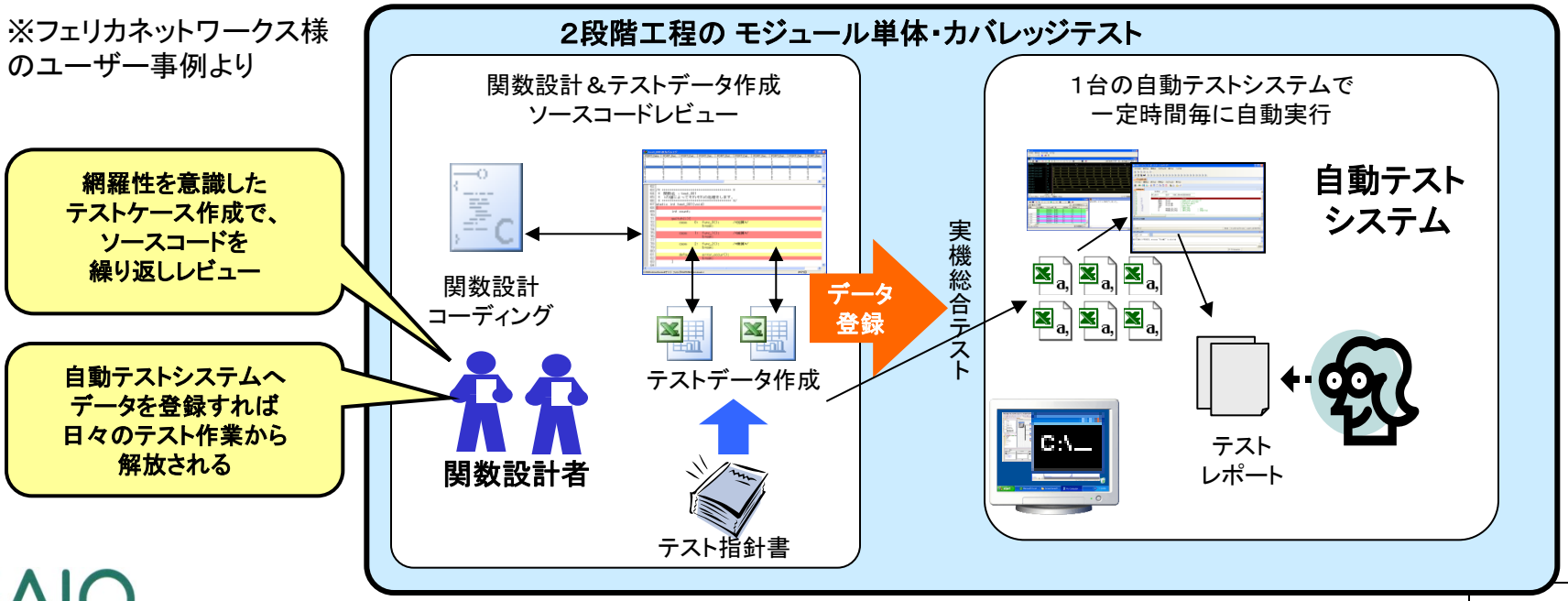
テストベクタ数
↑
関数の複雑度

※ガイオ単体テスト代行サービス
での納品物の例

# モジュール単体テストの効果例4

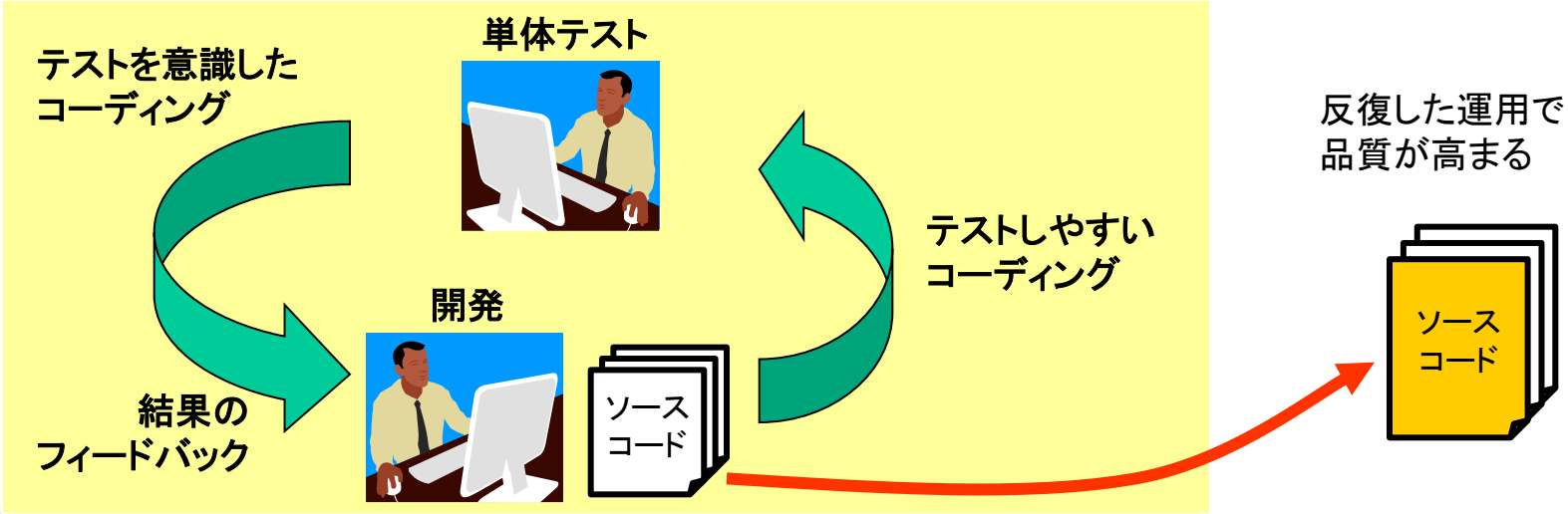
- **単体テストのテストデータ設計工程で ソースの不具合を発見する**
  - 「網羅性」を意識したテストケース作りの過程で ソースの潜在バグを発見することが多い
    - 型の間違い、符号有無の間違い、パスの確認から不要コードを発見　など
- **自動テストシステムで 工数をかけない反復テストの実施が可能**
  - 全ての単体テストを自動実行することで、ソース修正による影響（デグレ）を発見し易い

※フェリカネットワークス様
のユーザー事例より



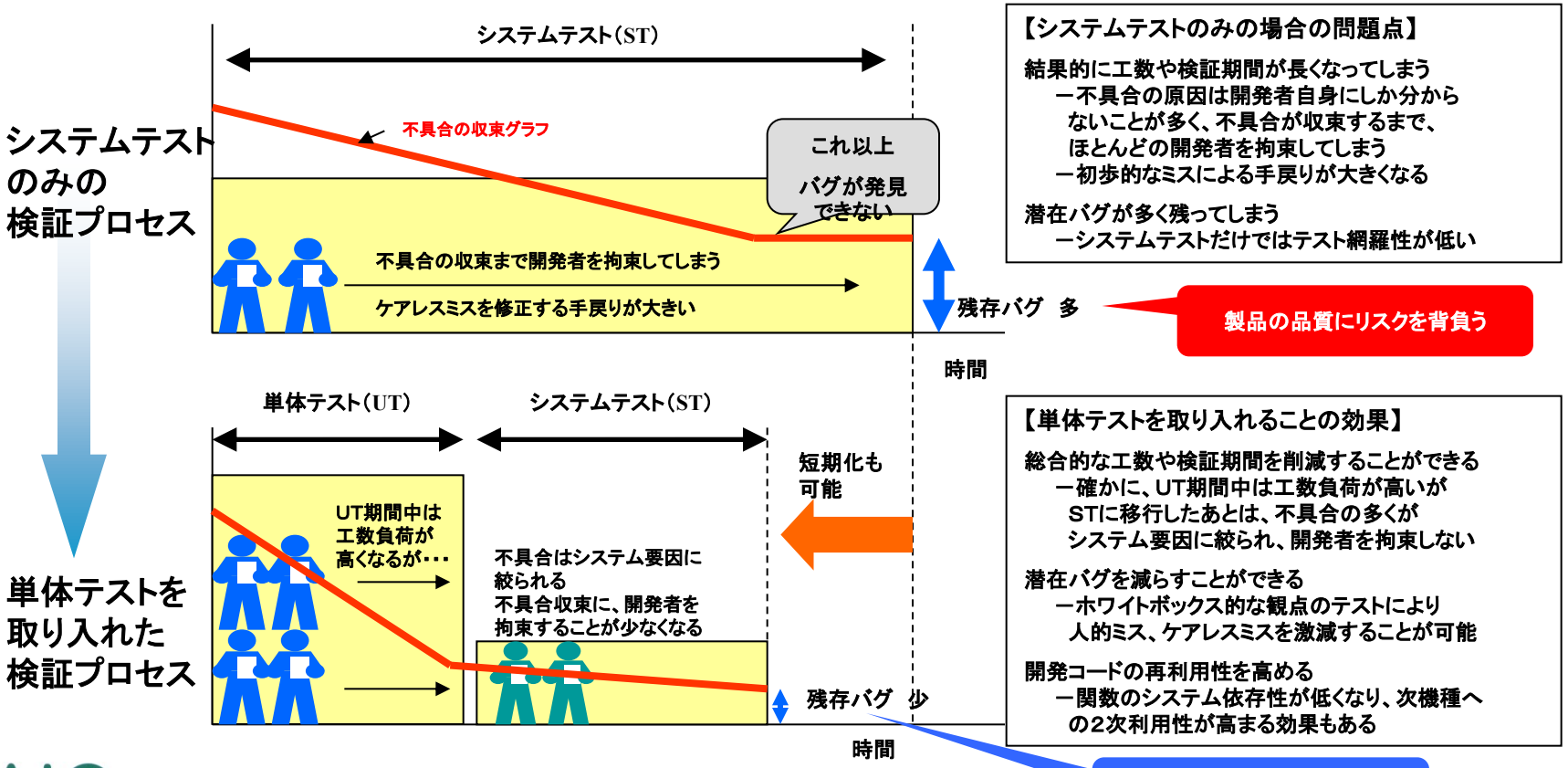
# 単体テストの定着運用による効果

- **単体テストを「開発プロセス」化、定着運用することでソフト品質が高まる**
  - 1つのプロジェクトへの適用での 即時効果よりも、
    数回のプロジェクトへの運用で、再利用されることにより効果が高まる
- **単体テストの実施は 次のソフト設計に良い影響を与える**
  - 単体テスト実施で得た不具合情報を設計にフィードバック
  - テストしやすいコーディング　← 品質の高いソフトの基本条件
  - テストを意識したコーディング　← 網羅性を意識したソフト設計に

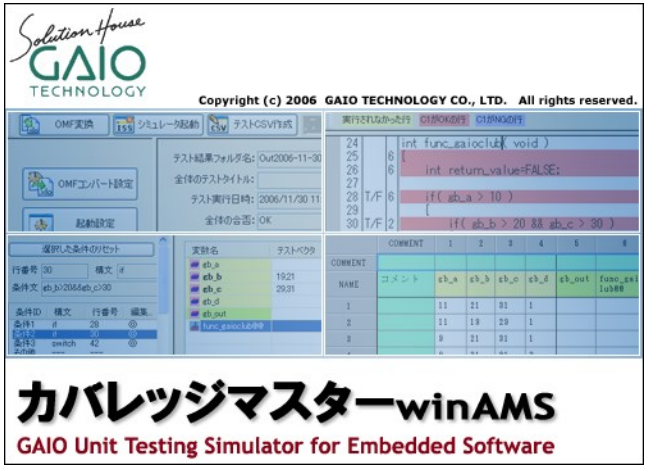


# （参考）単体テストの有無による比較

■ 単体テスト導入により 総合的な工数と残存バグの削減が可能
- HDDレコーダ開発会社の効果事例（※ユーザーヒアリングに基づく）

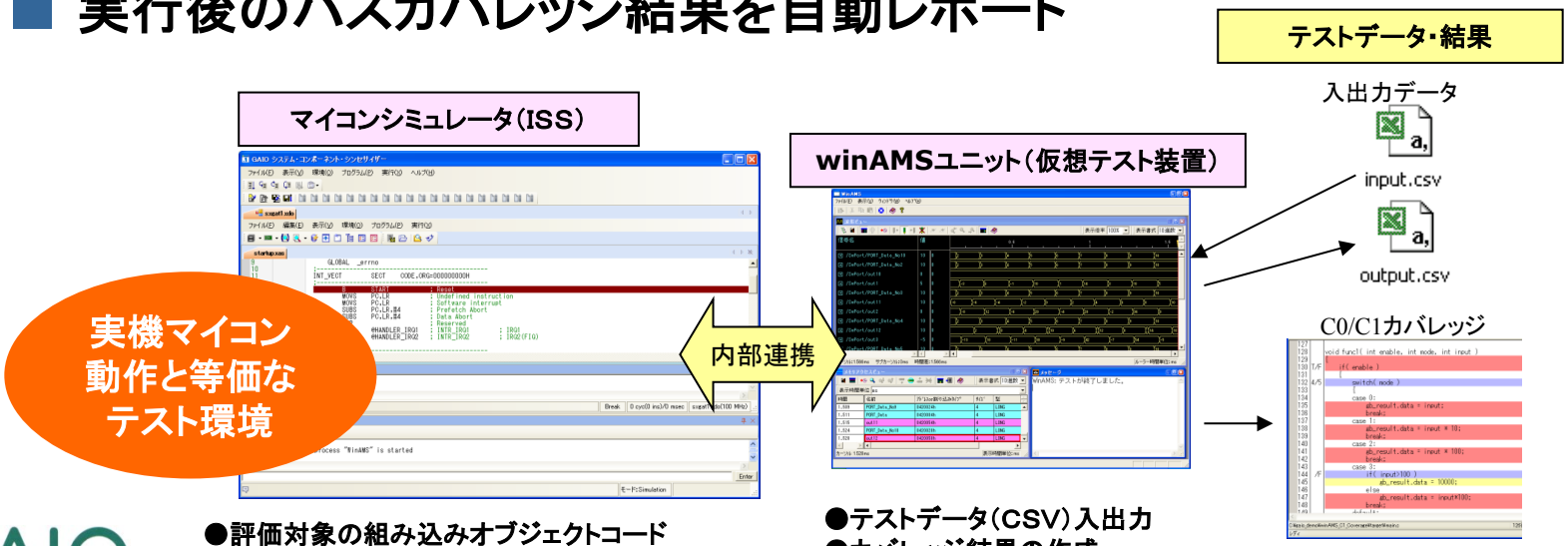


# 単体テストを自動化する
# カバレッジマスターwinAMS



# ガイオ カバレッジマスター winAMS 特長

- 組込みソフトをターゲットとした 単体テストツール
- マイコンシミュレータ（ISS）を使用して「実機コード」をテスト
  - マイコン仕様、コンパイラなどの、ターゲットへの実装時の問題を含めた評価が可能
  - 単体テストのソースコード、ビルド環境は、製品開発のものをそのまま適用可能
- テストドライバ作成、ソースコードの変更は不要
- テストデータ（関数、変数名）は全てCSVファイルで入出力
- 実行後のパスカバレッジ結果を自動レポート




【開示及び用途制限資料　ガイオ・テクノロジー株式会社】

# 関数への入出力テスト方法（仕組み）

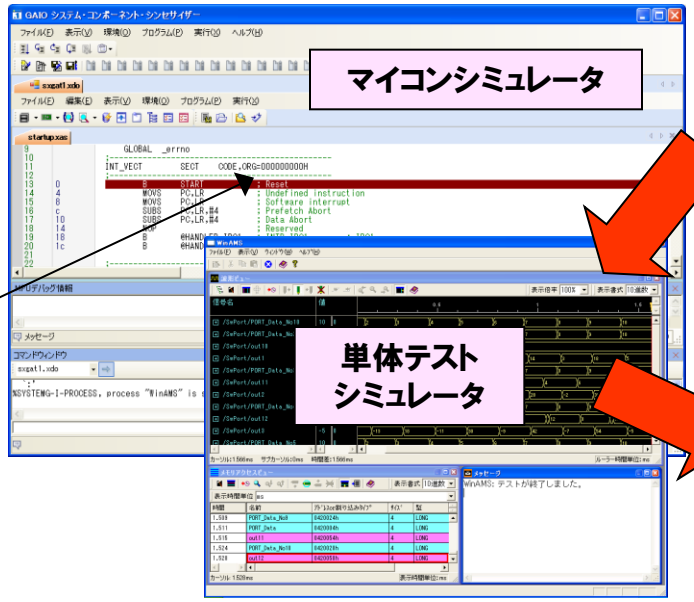
## ■ 関数を単体実行する仕組みをシミュレータ本体に実装

- クロスコンパイルした 組み込みオブジェクトをそのまま使用
- 関数名、変数名、テストデータをCSVで入出力
- 期待値との照合結果を自動レポート

**試験対象のソースコード**

```
base(int a, int b, int c)
{
   if (a == 1)
  {
    if (b == 1)
    {
      idx = 0;    // data[0]
      if (c == 1)
        pos = 0;   // data[0].str[0]
        :
        :
    }
  // 結果の設定
    value = data[idx].str[pos]; //- 1;
}
```

・組み込みソースをそのまま使用
・コンパイラも現在お使いのものを
　利用可能
・バッチ処理で自動テスト




**テスト入力データ　CSV**

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | mod | base | base単体 | 3 | 1 |
| 2 | #COMMENT | @a | @b | @c | value |
| 3 | | 0x20 | 0 | 1 | -1 |
| 4 | | 0x1f | 1 | 2 | 10 |
| 5 | | 0x1e | 1 | 1 | 100 |
| 6 | | 0xff | 1 | 2 | 200 |

・対象の関数名
・入力変数名＆入力データ
・出力変数名＆期待値

**テスト結果出力　CSV**

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | mod | base | base単体 | 3 | 1 | |
| 2 | #COMMENT | @a | @b | @c | value | |
| 3 | | 0x20 | 0 | 1 | -1 | OK |
| 4 | | 0x1f | 1 | 2 | 16?(10) | NG |
| 5 | | 0x1e | 1 | 1 | 100 | OK |
| 6 | | 0xff | 1 | 2 | 200 | OK |

・出力変数名＆変数結果出力
・期待値との比較結果（OK or NG）

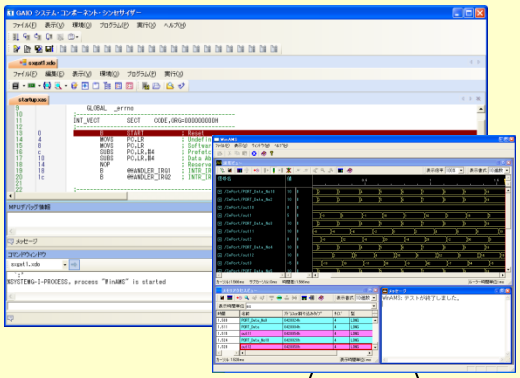
**カバレッジマスターwinAMS パッケージ**

# C0/C1カバレッジ自動レポート（標準機能）

■ 「カバレッジマスター winAMS」C0/C1カバレッジ結果を自動レポート

- 条件分岐によるコードパスの網羅テストで品質を保証
- 「winAMS」により テスト実行後に 実行したソース行を色表示
- 組み込みソフトのC0/C1カバレッジの自動テストを実現

**winAMS　シミュレータ**

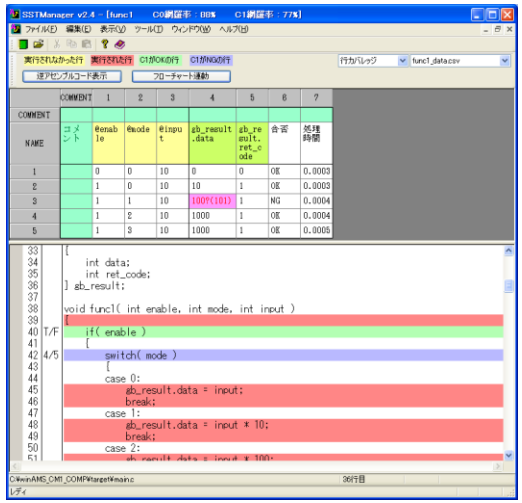


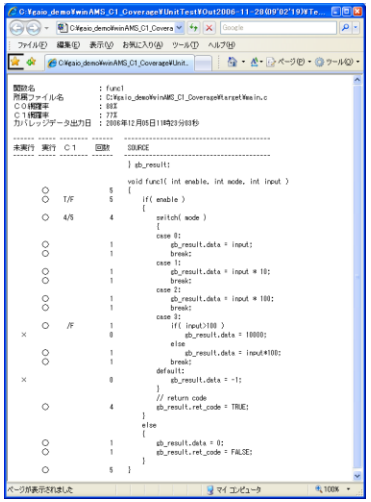
入出力CSV
データ



実行後

**C0/C1カバレッジ結果表示**





クリックしたテストデータによる実行ソース行を色表示

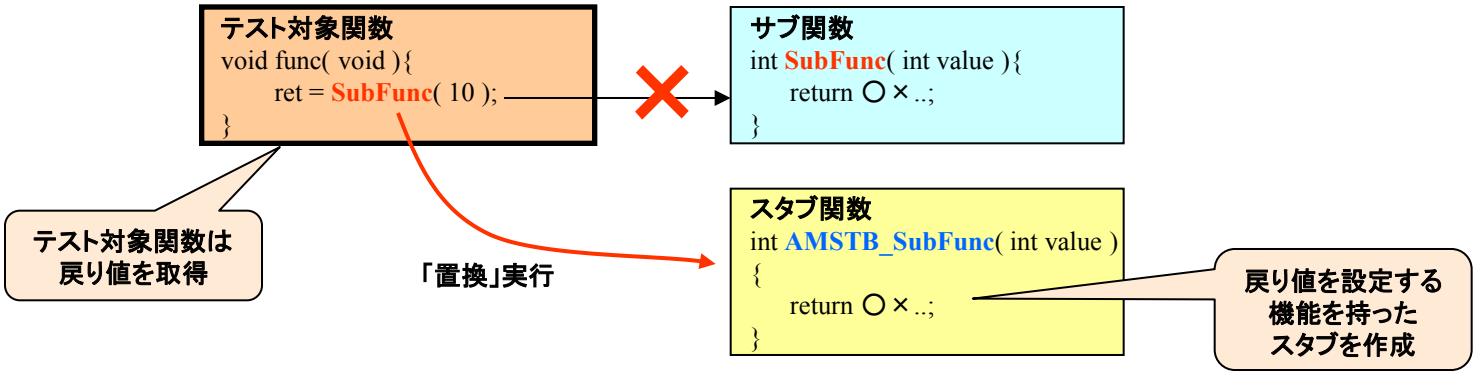
# サブ関数スタブ作成機能（標準機能）

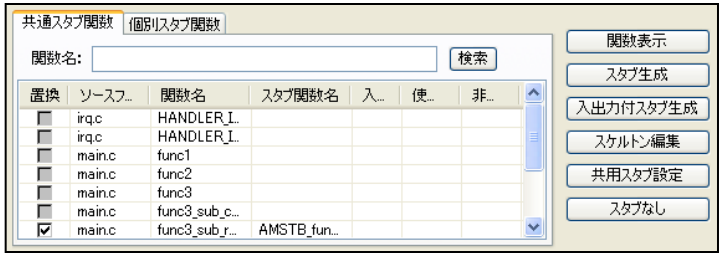
## ■ 単体テストのフェーズでは 関数を独立させて単体で評価する

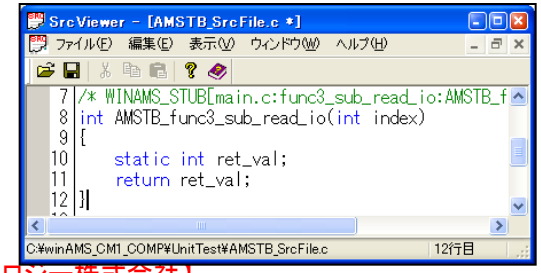
- テスト対象関数とサブ関数の依存関係を無くすためにスタブ関数を作成

## ■ スタブ関数作成・管理機能をサポート

- 元のソースコード修正なしで サブ関数の入れ替えが可能
- 関数リスト上の「置換」スイッチで スタブのON/OFFを簡単に切り替え可能
- テスト対象の関数呼出し部分のコーディングは、一切修正不要



# ポインタ変数対応・その他機能（標準機能）

## ■ ポインタ変数・引数の実体を自動割り付け

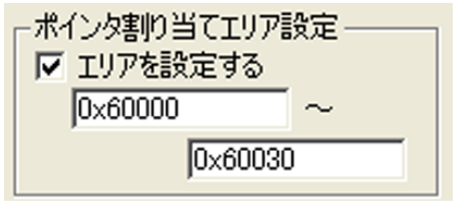
- 変数の実体を伴わないポインタ変数・引数の場合でもそのままテスト可能
- 入力データCSVの変数名に「$」を付けるだけで 実体を自動割り当て
- 割り付けエリアは MPUのメモリモデルから 自由に指定可能

**$**

```
int func1( char *data )
```

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | mod | func1 | タイトル | 2 | 1 |
| 2 | #COMMENT | $func1 @data | func1 @data[0] | func1 @@ | |
| 3 | | 1 | 0x1f | 0 | |
| 4 | | 1 | 0x20 | 1 | |
| 5 | | 1 | 0x21 | 0 | |

ポインタ変数の実体を作るための割り当てエリアを指定




## ■ CSVファイルに「～」で 期待値の範囲指定も可能

- レンジ内に入ることを期待値とする場合
- 浮動小数点変数に対する有効数字範囲指定

| | | |
|---|---|---|
| 2 | value | data[0].c |
| 3 | 100~200 | 2 |
| 4 | 100~200 | 2 |

# アセンブラ混在表示（標準機能）

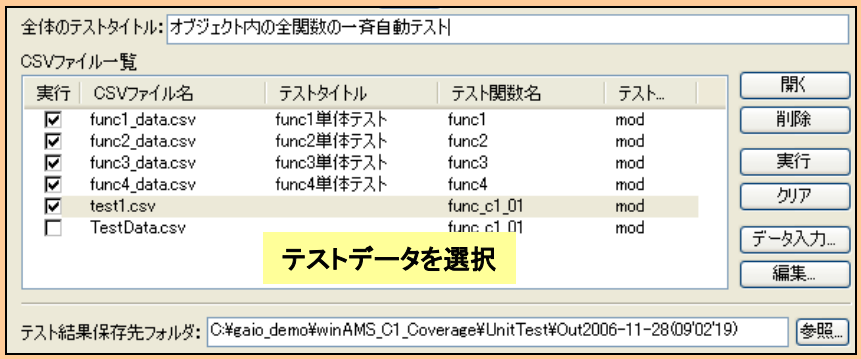

■コンパイラが生成したデバッグ情報を基にソースと生成コードの対応を混在表示

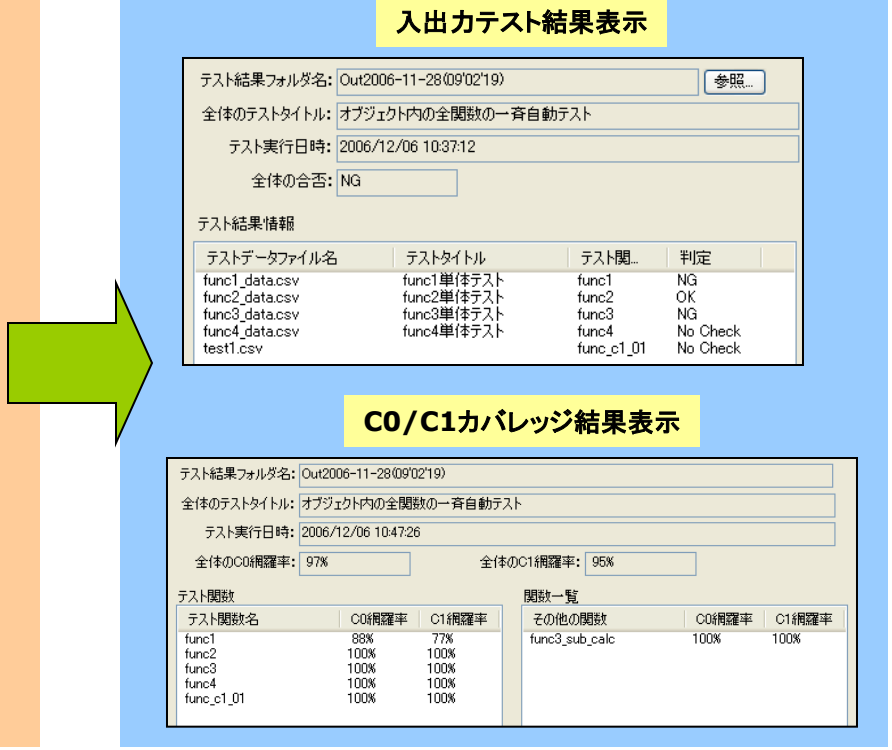
クロスコンパイラの最適化等により、Cソースと生成されたマイコンコードの不一致が起こる場合がある為、混在表示で影響を判断出来ます。

# 選択テストデータを一斉自動実行（標準機能）

## ■ 管理ツールSSTManagerで選択したデータを自動実行

- 指定したCSVデータ（関数）を自動バッチ実行
- 結果表示まで 全て自動実行




ダブルクリックで各結果詳細表示へ

# 機能安全規格ISO26262/IEC61508
# ツール認証について

# 機能安全ツール認証をTÜV SÜDより取得

- **第三者認証機関であるドイツのテュフズード（TÜV SÜD）より機能安全ツール認証を取得**
  - ISO26262（自動車）、IEC61508（機能安全メタ企画）にて取得
  - ツール認証取得製品　（2013/6/21に 取得）
    - カバレッジマスターwinAMS/ゼネラル　V3.6.x：　単体テストツール
    - CasePlayer2　V5.6.x：　プログラム解析ツール

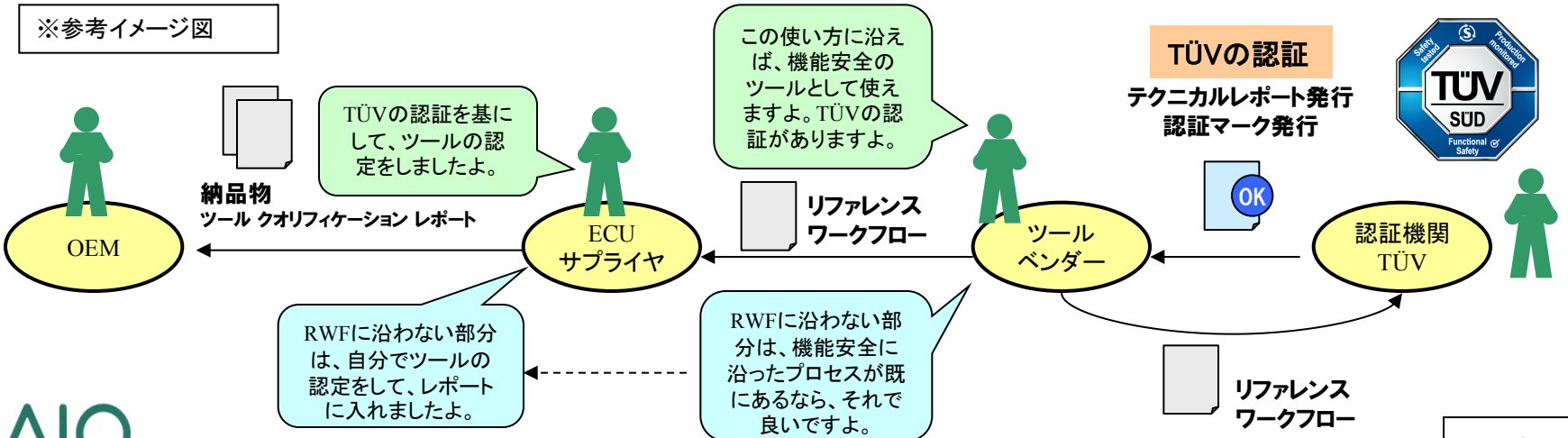
- **結合テスト向け「関数/コールカバレッジ計測機能」もツール認証対象に**
  - 単体テスト機能に加え、結合テストで求められる「関数/コールカバレッジ」計測機能も、ツール認証の対象に含まれます。

- **TCLの設定によらず適用可能**
  - 最も煩雑なツール認定作業が必要となるTCL3に設定した場合にも適用されます
    - ユーザ様は、TCLによらずツール認定に関わる作業を省略することが可能です

# リファレンスワークフロー（RWF）とは？

- **ISO 26262に適合した静的解析、単体テストを行うのに必要なガイオツールの使用方法（ワークフロー）を定義した文書**

- **TCLによらずツール認定に関わる作業を省略可能**
  - ISO26262は開発に用いるソフトウェアツール毎に設定するTCL (Tool Confidence Level)に応じた開発ツール認定レポートの作成を ユーザーに要求します
  - 「カバレッジマスターwinAMS/ゼネラル」「CasePlayer2」はTÜV認証取得済み
  - 最も煩雑なツール認定作業が必要となるTCL3に設定した場合にも適用可能



# ユーザーにツール認定に必要な資料を提供

## ■ GAIOユニットテストツールセット リファレンスワークフロー（RWF）

- 保守契約を頂いているユーザー様からの要求に基づいてガイオが提供
  - 提供先を管理されたペーパードキュメントで提供致します　（複写、転送はできません）
- 機能安全に対応する静的解析、単体テストを行うために
  ユーザー様は、RWFに沿ったプロセス（テスト手順）を構築する必要があります

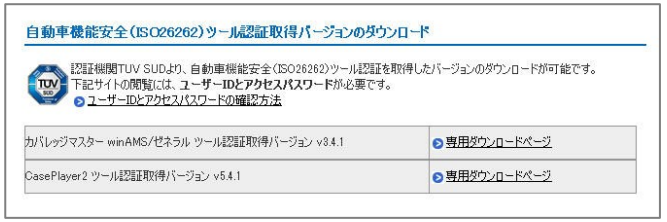
# 認証バージョンの提供とバージョンアップについて

## ■ ISO26262/IEC61508認証バージョン専用WEBページを設置

機能安全（ISO26262/IEC61508）ツール認証取得バージョン

- カバレッジマスターwinAMS/ゼネラル：v3.6.x
- CasePlayer2：v5.6.x

ガイオがアップデートバージョンで再認証を取得するまで、
上記認証取得バージョンをいつでもダウンロード可能




## ■ 認証バージョンに対するバグフィックス等は認証の範囲内で個別対応

機能安全に影響を与えないアップデートは、リビジョンアップにて対応

## ■ 再認証は機能安全へ与える影響とお客様のご意見を考慮し対応を決定

※お客様自身で最新版のプライベート認定を行う場合は、資料提供などを協力致します

**認証バージョンの**
**リビジョンアップ**
**（v3.6.x ／ v5.6.x）**

**認証の範囲内で個別対応**
**専用WEBページにて提供**

TÜV SÜD
2012/06
認証済み

**関数/コールカバレッジ**
**IEC61508対応を含む**

**機能追加を含む**
**最新版**
**（v3.X ／ v5.X）**

**未認証**

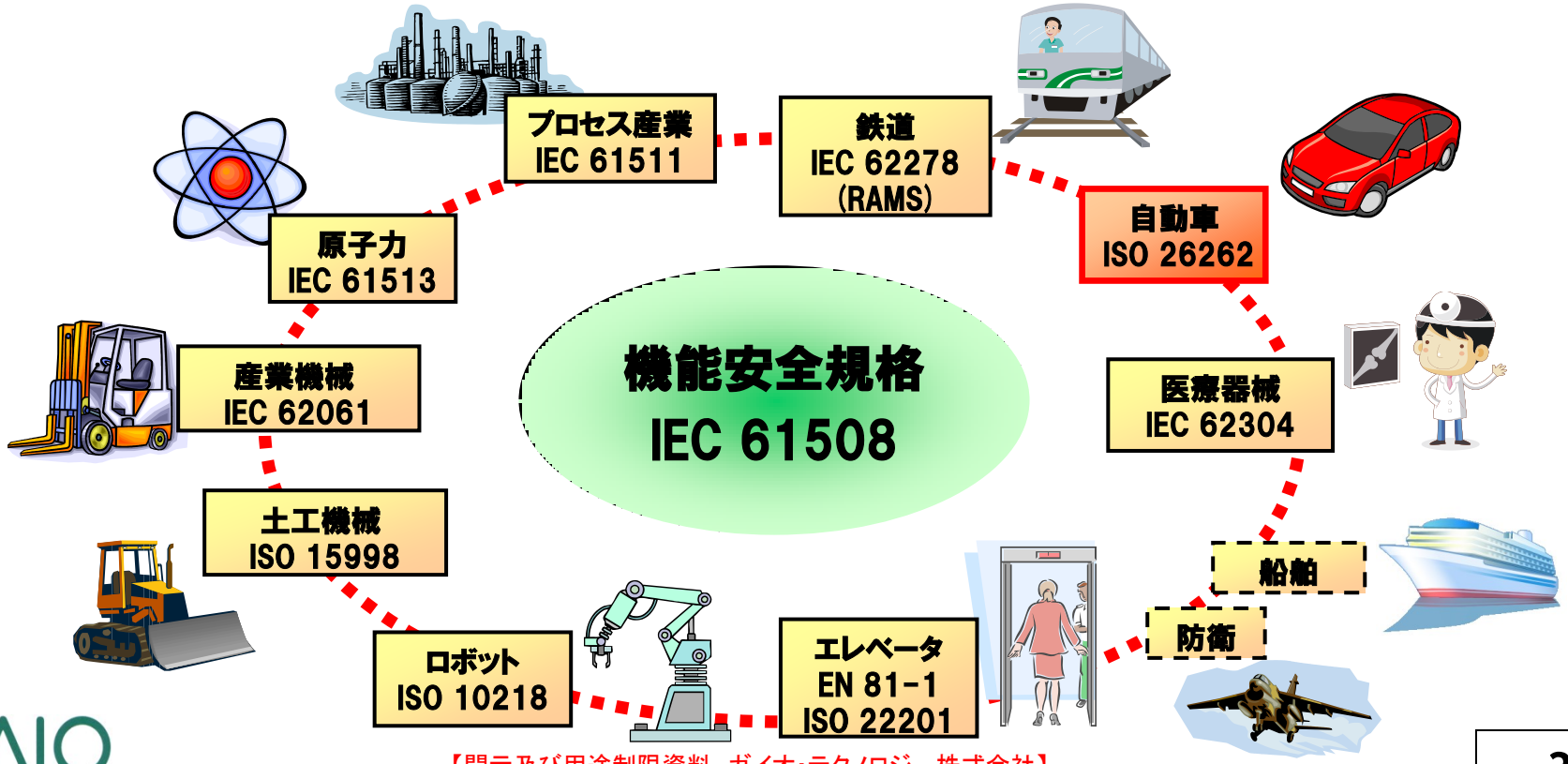
**通常のアップデートで提供**
**再認証はお客様のご意見を考慮して**
**対応時期を決定**

GAIO TECHNOLOGY

# （参考）機能安全規格 IEC61508 / ISO26262

## IEC61508（メタ規格）のもと各分野ごとに機能安全規格が派生

- 自動車機能安全（ISO26262）はもとより、
  IEC61508のツール認証により 広範囲な機能安全のテストに使用可能



# 機能安全（IEC61508 / ISO26262）への カバレッジマスターの優位性について

# 自動車機能安全規格ISO26262の推奨事項

- **ソフトウエアの構造網羅テスト、カバレッジテストに関する項目が規定**
  自動車安全度水準ASILに応じて、ステートメントカバレッジ、ブランチカバレッジ、MC/DCカバレッジの計測が推奨されている
- **「単体テストの環境はターゲット環境に可能な限り合わせること」を推奨**
  カバレッジマスターwinAMSは「**実装コードがそのまま動く単体テスト**」が可能な業界唯一のツール

**IEC 61508 META-STANDARD**
EN 5012x(鉄道)
IEC 60601 1-4 (Medical)
IEC 62304(医療機器ソフトウェア)
IEC 61513 (原子力)
IEC 61511 (プロセス産業)
ISO EN 12100 (Machinery)
**ISO 26262 (自動車)**

- Part1:Glossary of Terms
- Part2:Management of Functional Safety
- Part3:Concept Phase
- Part4:System Development
- Part5:Hardware Development
- **Part6:Software Development**
- Part7:Production and Operation
- Part8:Supporting Processes
- Part9: ASIL- and safety-oriented analyses

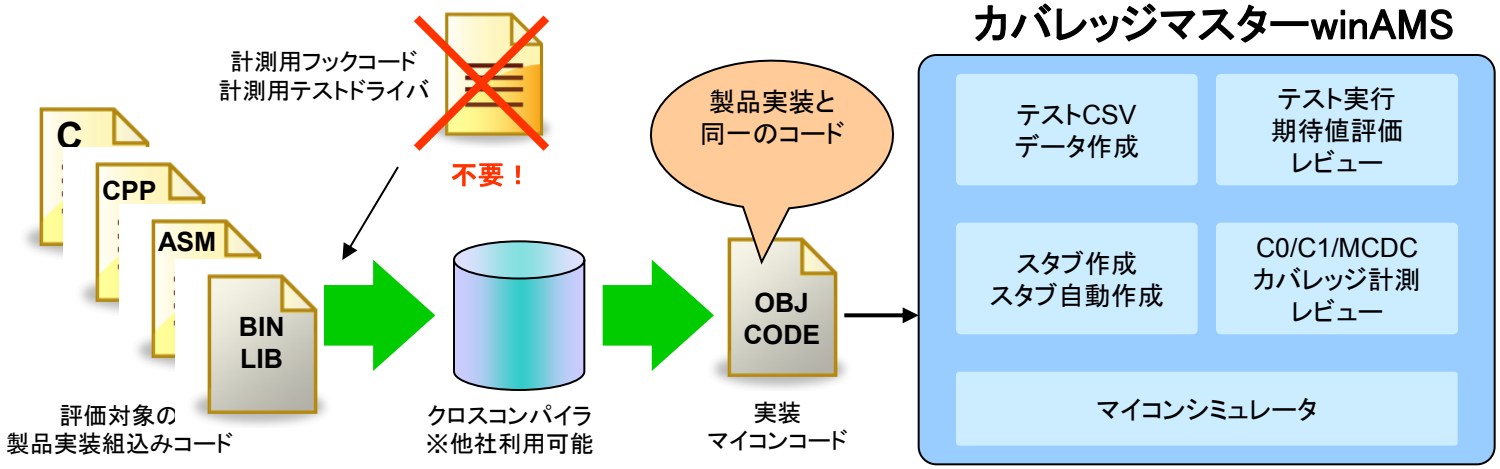
**6-9　ソフトウェア単体テスト**
**構造網羅テスト**
**ステートメントカバレッジ**
**分岐カバレッジ**
**,MCDCカバレッジ**
機能レベルでの欠陥投入
モデルベーストテスト
MIL,SIL,PIL、HIL
**モデルとコードの一致性テスト(BTBテスト)**

**6-10 ソフトウェア統合とテスト**
**モデルとコードの一致性テスト(BTBテスト)**
モデルベーストテスト
SIL,PIL,HIL
欠陥投入テスト

# フックコードを使わない「実コード」単体テスト

## ■ 単体テスト、カバレッジ計測用フックコード追加なしで実行可能

- 計測用コード挿入のない、製品実装コードそのものを実行して単体テストを実行
- 計測の仕組みはコードではなく、シミュレータ本体に機能実装
- マイコン依存問題、クロスコンパイラコード展開、最適化処理を含めた高信頼性テスト
  - ターゲットと異なる要素を全て排除した 最も信頼性が高いテストとなる

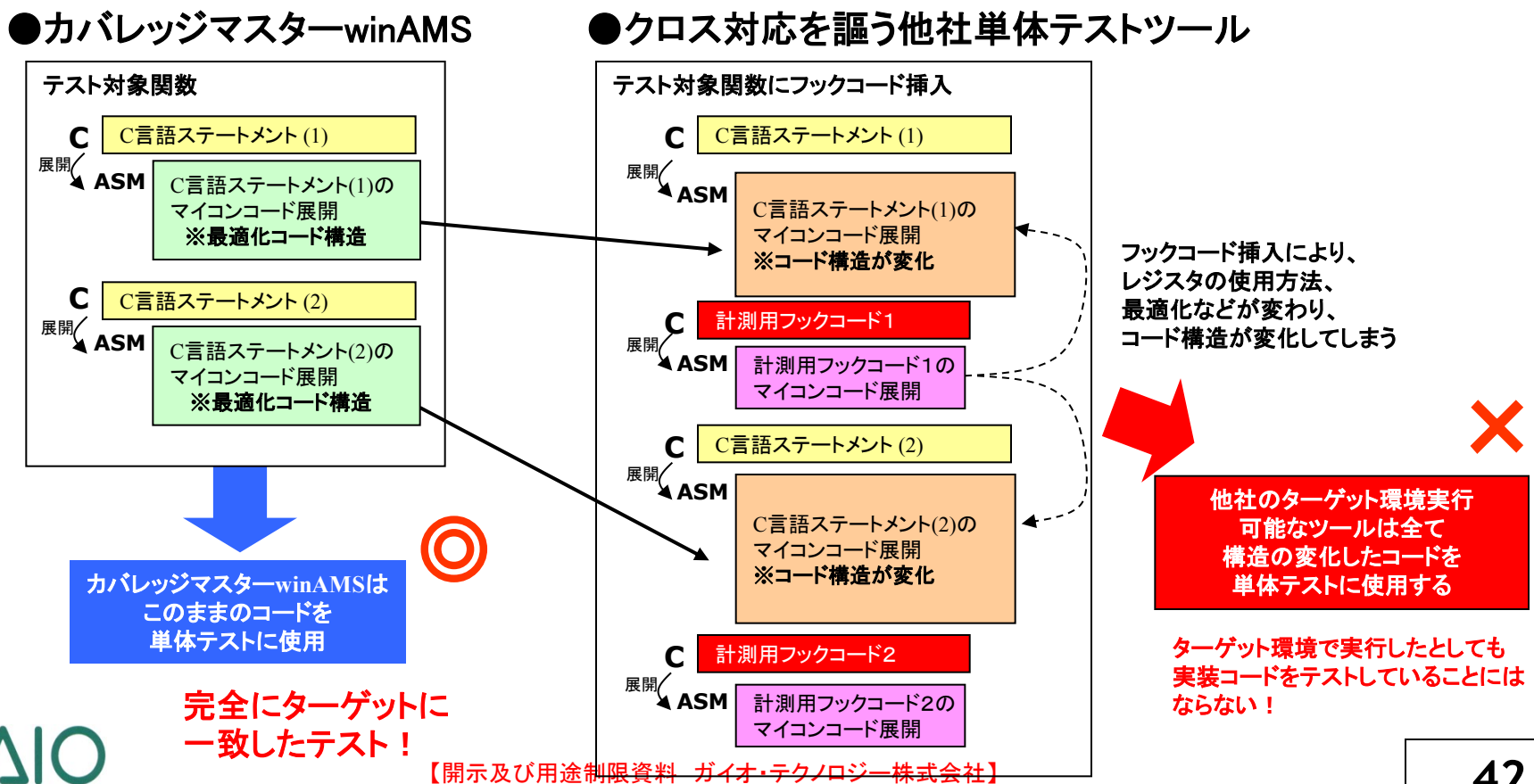



**実装コードがそのまま動く単体テスト**

# 他社単体テストツール　フックコード挿入の影響

## ■ ターゲット環境での実行を行う他社ツールは コード構造を変化させる

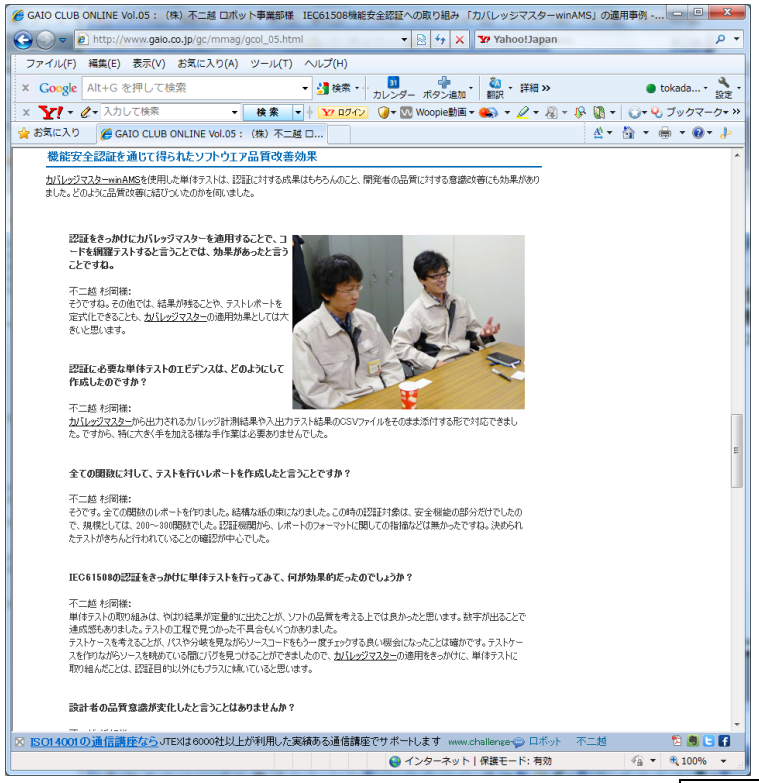
ターゲット環境で実行したとしても、実装コードそのものをテストしていることにはならない！



# 機能安全認証IEC61508 への適用事例

■ （株）不二越 ロボット事業部様の　IEC61508機能安全認証への取り組みと単体テストツール「カバレッジマスターwinAMS」の適用事例

– GAIO CLUB / 組込み開発技術者のための技術情報Vol.05 (2011/1/24)に掲載

# 導入実績･発表事例
# （参考資料）

# カバレッジマスターwinAMSの導入実績/事例

## ■ 導入実績

自動車業界、鉄道、デバイス関連、通信機器、空調関連、メカトロ機器、プリンター複写機、FA機器、計器関連、エネルギー関連、交通機器関連、等々

## ■ 多数のユーザ事例を発表

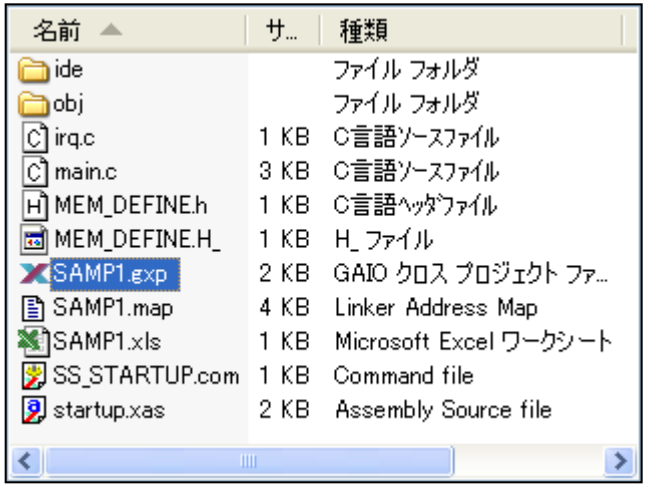
・不二越（株）　：2011/01 GAIO CLUB ONLINEに事例紹介
・パナソニック（株）　：2009/06「ガイオ品質向上セミナー」にて発表
・パイオニア（株）　：2008/02 組込み情報誌「ガイオ倶楽部」記事にて発表
・ソニーLSIデザイン（株）　：2007/11「単体試験の賢い選択・活用セミナー」にて発表
・日産自動車（株）　：2006/08「組込みソフト単体テスト成功事例セミナー」にて発表
・アイシン精機（株）　：2006/08「組込みソフト単体テスト成功事例セミナー」にて発表
・フェリカネットワーク（株）　：2005/12 組込み情報誌「ガイオ倶楽部」記事にて発表

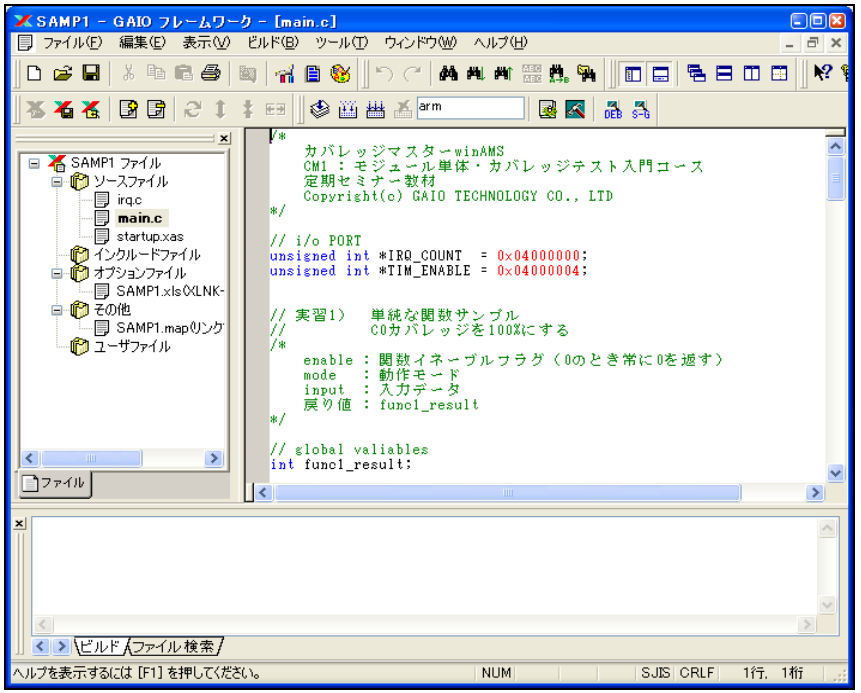
# さあ、始めましょう！
# 実習準備
# 評価ソースサンプルを確認

# 評価ソースサンプル プロジェクトを開く

## ■ ガイオ統合開発環境プロジェクトを開きます

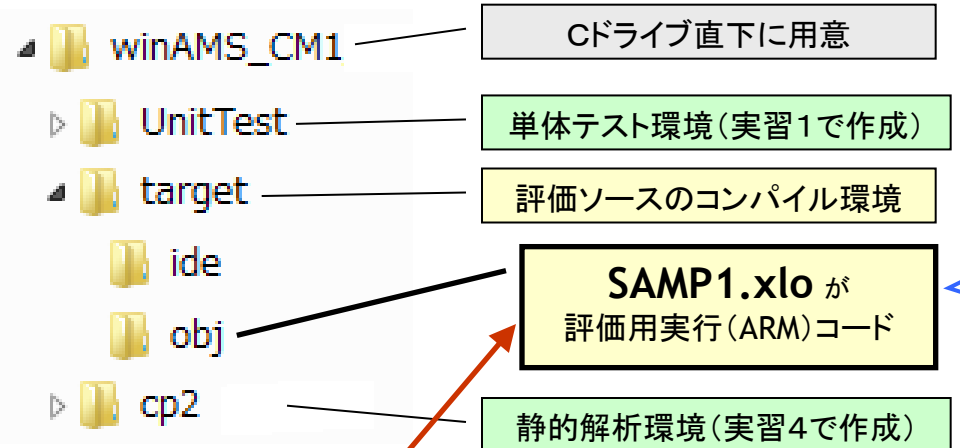
1. C:¥winAMS_CM1¥target フォルダを開きます
2. 「SAMP1.gxp」をダブルクリックして開発環境を立ち上げます
3. 「ビルド」メニューから「リビルド」を選択して、コンパイルします。



# 実習環境について

## ■ 実習教材のファイル構成




**【参考】**
**他社製（半導体メーカー製）コンパイラで**
**生成したオブジェクトファイルの場合**

（例）ルネサス製HEW環境の場合
・debugモードでコンパイル
・「*.abs」（ELFフォーマット）が生成される


**他社製コンパイラで生成したオブジェクトファイルは**
**カバレッジマスターの「OMF変換」機能で**
**SAUFフォーマット（*.xlo）に自動変換**

# 実習1
# 評価ソースサンプルの
# C0網羅率を100%にする
# (テストデータはEXCELで作成)

カバレッジマスターの基本機能の学習

# 実習1-1：テストプロジェクトを新規作成

## ■ カバレッジマスター用テストプロジェクトを新規作成します

1. デスクトップ「SSTManager」を起動します。
   ※「スタート」→「(すべての)プログラム」→「システムシミュレータ(winAMS)」→「SSTManager」を起動します。
2. 「ファイルメニュー」→「プロジェクト新規作成」を選択します。
3. ダイアログを以下のように設定して、「OK」を押します。


← プロジェクト名のフォルダができます。
← プロジェクト名のフォルダが作成されます
「C:¥winAMS_CM1」を選択

← 「systemg.exe」を選択

← 「ARM」を選択

← 「ARM7DM Little Endian」を選択

← XASS(GAIOクロス開発環境)

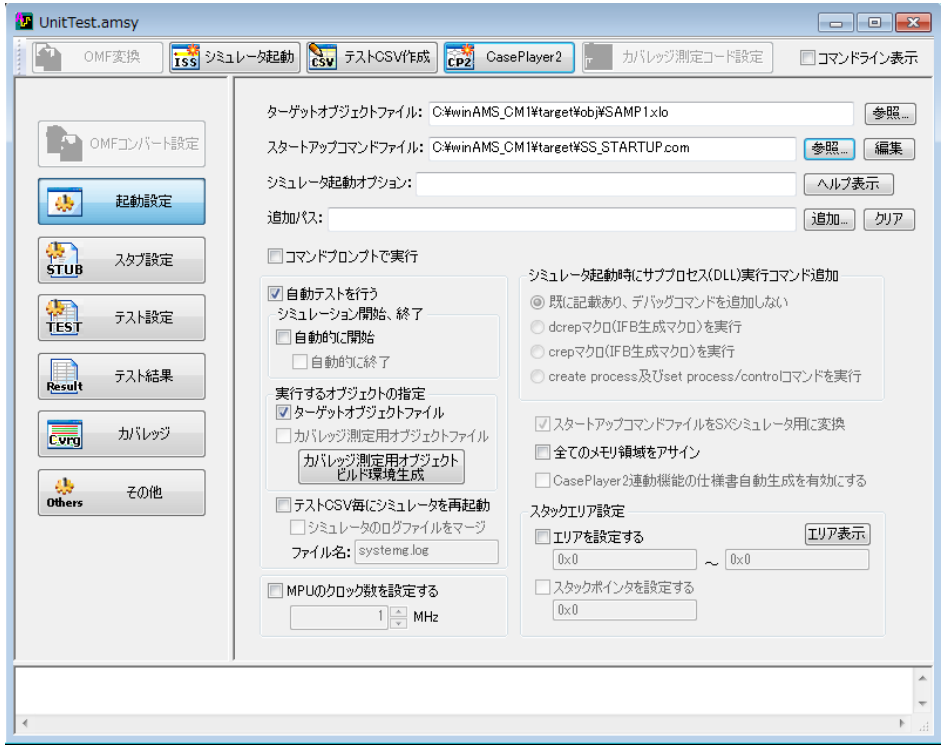
← winAMSを選択

← 実習1～3では空欄
※実習4で使用します。

# 実習1-2：起動設定

■ テスト対象ターゲットオブジェクトと、シミュレータ起動コマンドファイルを指定

1. オブジェクトファイルに、「C:¥winAMS_CM1¥target¥obj¥SAMP1.xlo」を指定します。
2. スタートアップコマンドファイルに、「C:¥winAMS_CM1¥target¥SS_STARTUP.com」を指定します。
3. 「自動テストを行う」にチェックを付けます。

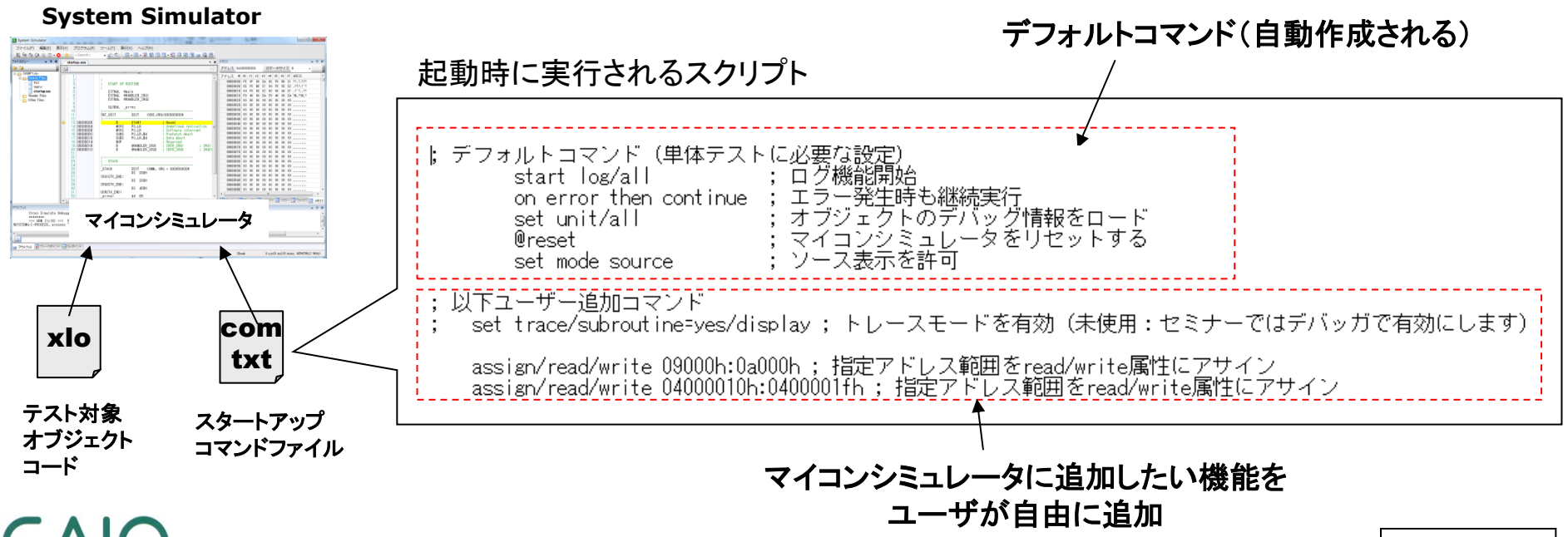


# (参考)スタートアップコマンドファイルとは？

## ■ マイコンシミュレータの起動時に実行されるスクリプトファイル

- マイコンシミュレータの設定内容をテスト事情に合わせて変更するために使用
  - メモリ領域の属性を変更（ROM領域→RAM領域）
  - レジスタ変化待ちなどのループをブレイクする操作　など

※通常の関数テストには必要なし



# （参考）ユーザー向け技術サポート情報

- **WEBサイトに技術サポート情報を掲載**
  - 「ユーザーサポート」トップページからアクセス →【ユーザー向け技術サポート情報】
- **カバレッジマスターwinAMS/ゼネラル 向けノウハウ情報**
  - **FAQ（よくある質問）**
  - **マクロ（シミュレータコマンド）の使い方　※上級者ユーザー向け**

**http://www.gaio.co.jp/support/user/tech_paper.html**


# 実習1-3：単体テストで網羅率を100%に

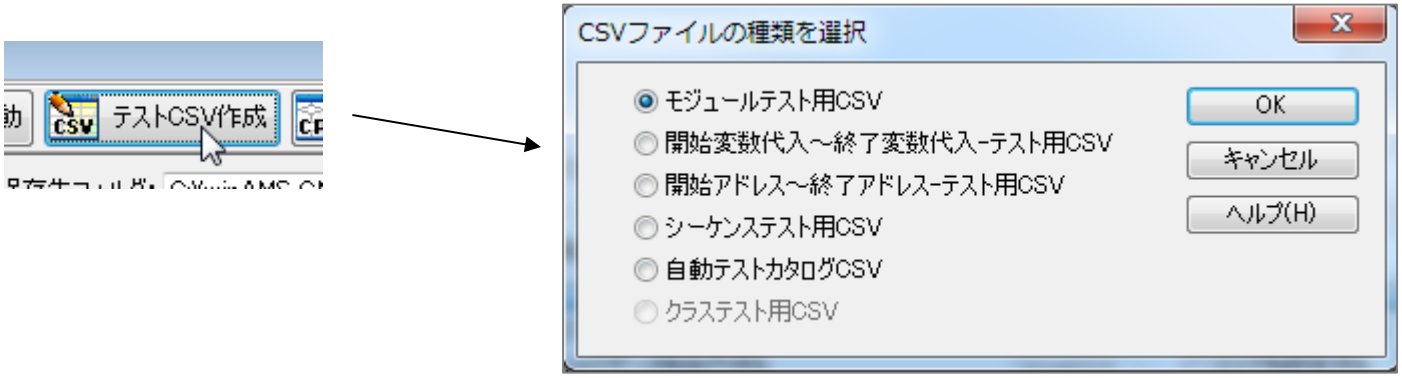
- **実習1は サンプルソースに対して、C0カバレッジを100%にするデータ作成を、EXCELで行います。**
  - サンプルソース（main.c）の func1() を単体テストします

```
// グローバル構造体変数
struct ST_PARAM
{
      int data;
      int ret_code;
} gb_result;

void func1( int enable, int mode, int input )
{
      if( enable )
      {
            switch( mode )
            {
            case 0:
                  gb_result.data = input ;
                  break;
            case 1:
                  gb_result.data = input * 10;
                  break;
            case 2:
                  gb_result.data = input * 100;
                  break;
            case 3:
                  if( input >100 )
                        gb_result.data = 10000;
                  else
                        gb_result.data = input *100;
                  break;
            default:
                  gb_result.data = -1;
            }
            // return code
            gb_result.ret_code = TRUE;
      }
      else
      {
            gb_result.data = 0;
            gb_result.ret_code = FALSE;
      }
}
```

# 実習1-4：func1()テスト用CSV雛形を作成

- テスト対象は 関数func1()です　※「実習1-1」のソースコードを参照
- func1()の入力データと出力データの変数名を確認します
  - 入力変数【3個】：int enable, int mode, int input　← 全て引数
  - 出力変数【2個】：gb_result.data, gb_result.ret_code　← グローバル変数(構造体)

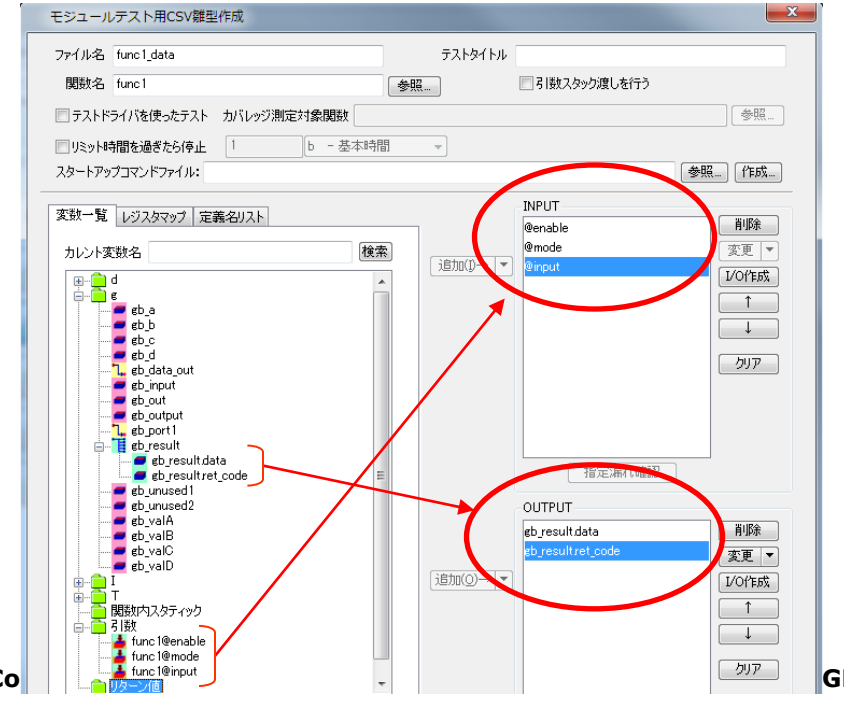
- モジュールテスト用CSV作成機能を使用して 入出力変数名を指定
  1. 上部の「テストCSV作成」ボタンを押します
  2. 「モジュールテスト用CSV」を選択して「OK」を押します



# 実習1-4：func1()テスト用雛形を作成（続き）

■ **モジュールテスト用雛形作成機能で、テスト関数名・変数名などを作成**

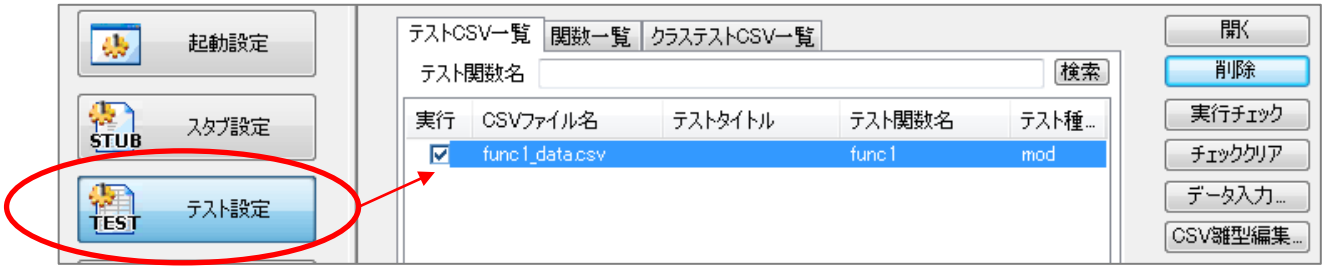
1. ファイル名「func1_data」、関数名「func1」を指定　（テストタイトルは任意）
2. 変数INPUT/OUTPUT ツリービューから 変数を選択して登録
   - INPUTに　func1@enable, func1@mode, func1@input　(@は引数の意味)
   - OUTPUTに　gb_result.data, gb_result.ret_code
3. 「OK」を押すと、CSVファイルが生成される



# 実習1-5：func1()　生成された雛形を確認

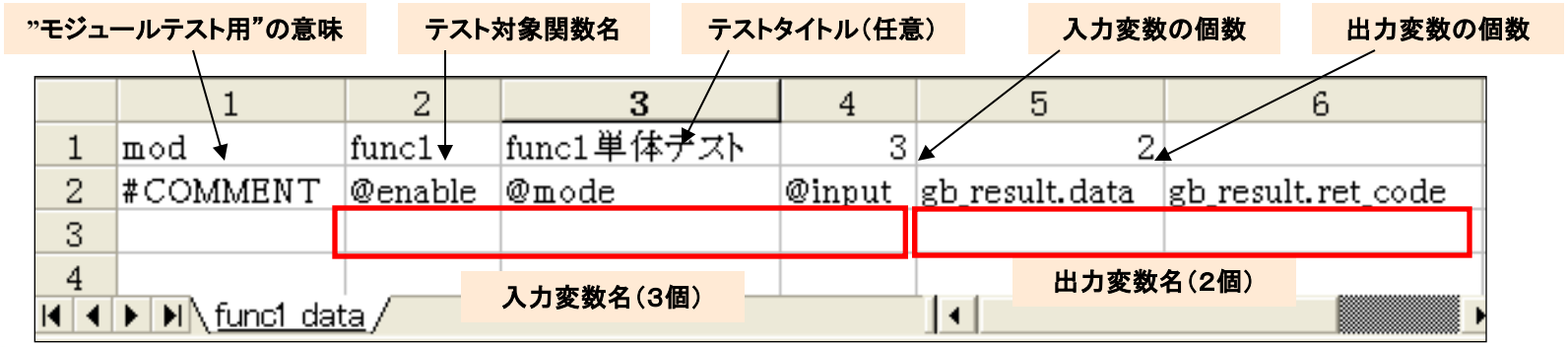
## ■ テスト用CSVファイルは「テスト設定」/「CSVファイル一覧」に表示されます

1. ウインドウ左の「テスト設定」ボタンを押して画面を切り替えます
2. 「CSVファイル一覧」の「func1_data.csv」をダブルクリックしてExcelを立ち上げます




## ■ 生成された雛形の内容を確認します

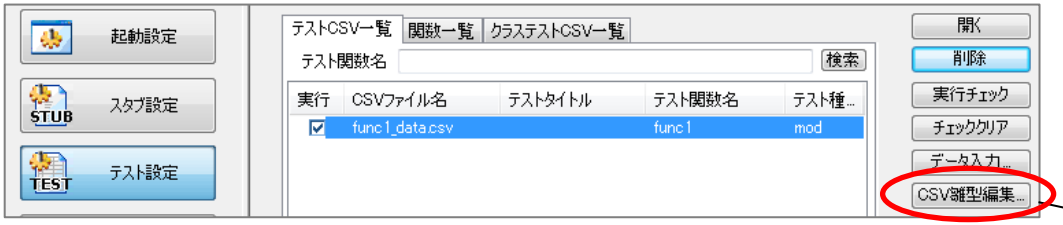
– ※雛形の内容は、エクセル上で編集しないでください　→次頁参考情報へ

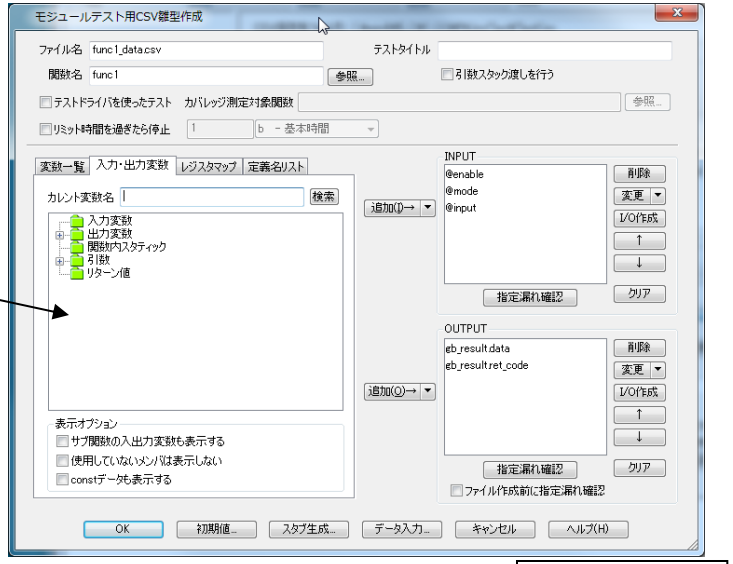


# （参考）CSVファイルの登録変数を変更する場合

- **「テスト設定」にてCSVファイルを選択し、「CSV雛形編集」ボタンを押す**
  - モジュールテスト用CSV雛形作成　の画面で再設定を行う
  - 「OK」ボタンで編集内容をCSVファイルに書き込む
    ※テストケースを設定した後であっても、テストケースは保存されます。

【注意】CSVファイルの変数等の雛形部分を、Excelなどで直接編集できますが、
正しい変数名やフォーマットである必要があります



# 実習1-5：func1()雛形にテストデータを追加

■ テストケースを追加します　CSVの1行が1回のテストに相当します

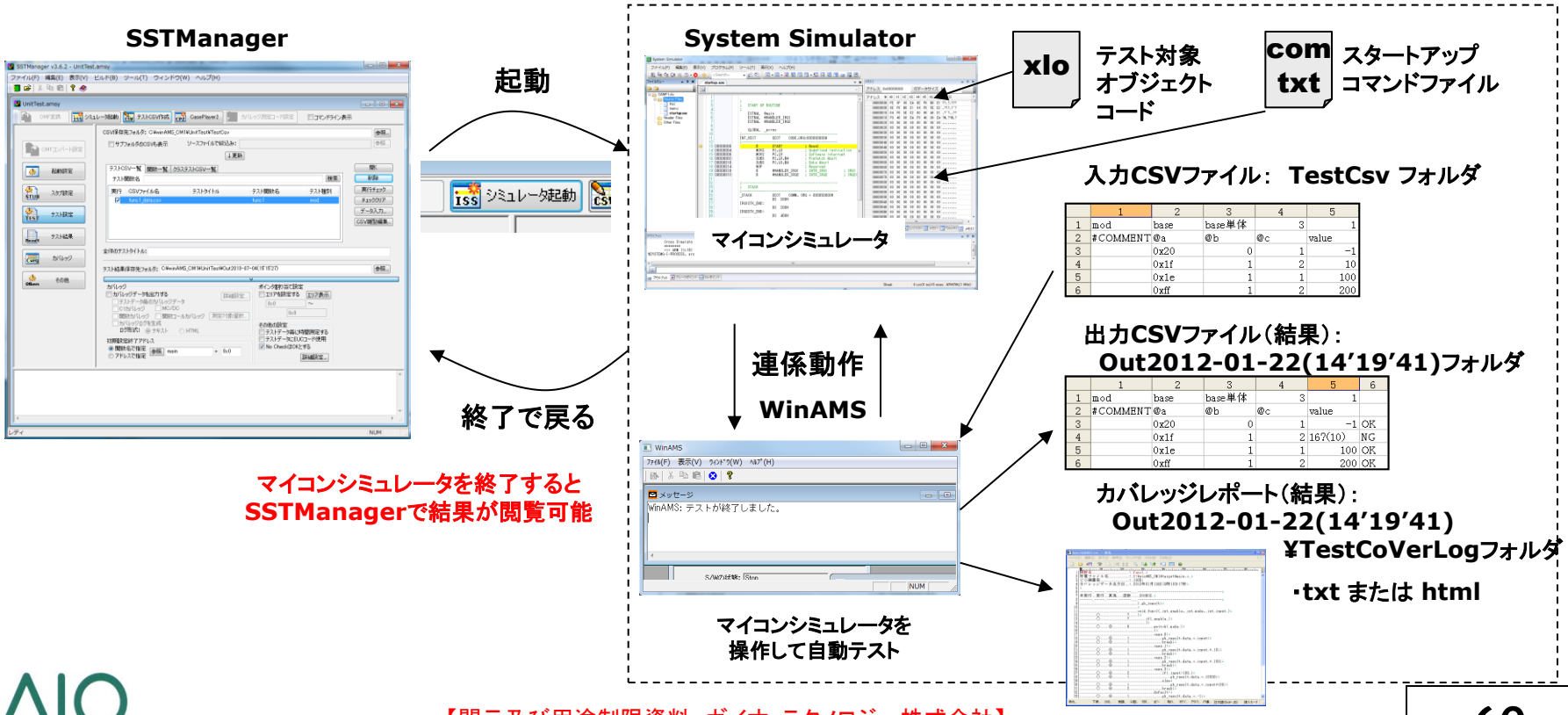
1. 3～7行目のINPUT変数にテストデータを加えます（5回分のテストケース）
2. gb_result構造体のメンバーには「期待値」を入れますが、今回は空欄にしておきます。
3. このCSVファイル「func1_data」を上書き保存します
4. EXCELを閉じます　（必ずファイルを閉じて下さい）

ここに入力

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | mod | func1 | | 3 | 2 | |
| 2 | #COMMENT | @enable | @mode | @input | gb_result.data | gb_result.ret_code |
| 3 | | 0 | 0 | 10 | | |
| 4 | | 1 | 0 | 10 | | |
| 5 | | 1 | 1 | 10 | | |
| 6 | | 1 | 2 | 10 | | |
| 7 | | 1 | 3 | 10 | | |
| 8 | | | | | | |

# （参考）カバレッジマスターの起動構成

■ テスト実行時には、SSTManagerから以下のモジュールが起動
- マイコンシミュレータ（ウインドウ名：System Simulator）
- 単体テストシミュレータ（ウインドウ名：WinAMS）



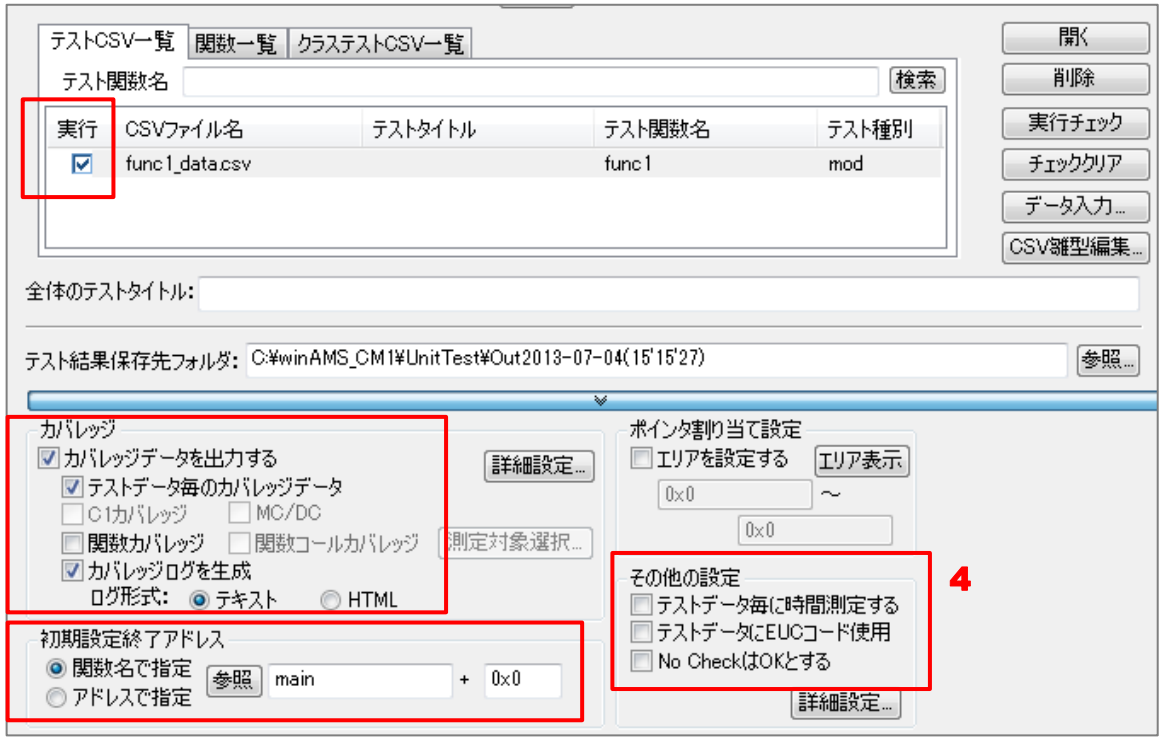

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | mod | base | base単体 | 3 | 1 |
| 2 | #COMMENT | @a | @b | @c | value |
| 3 | | 0x20 | 0 | 1 | -1 |
| 4 | | 0x1f | 1 | 2 | 10 |
| 5 | | 0x1e | 1 | 1 | 100 |
| 6 | | 0xff | 1 | 2 | 200 |

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | mod | base | base単体 | 3 | 1 | |
| 2 | #COMMENT | @a | @b | @c | value | |
| 3 | | 0x20 | 0 | 1 | -1 | OK |
| 4 | | 0x1f | 1 | 2 | 16?(10) | NG |
| 5 | | 0x1e | 1 | 1 | 100 | OK |
| 6 | | 0xff | 1 | 2 | 200 | OK |

# 実習1-6：func1()テスト実行

## ■ 実行のための設定を追加します

1. 「テストCSV一覧」で「func1_data.csv」の実行チェックボックスをONにします
2. 「カバレッジ」欄のチェックボックスを下図の様に設定する
3. 「初期設定終了アドレス」を「main」＋「0x0」にします
4. 「その他の設定」のチェックボックスを全てOFFにします

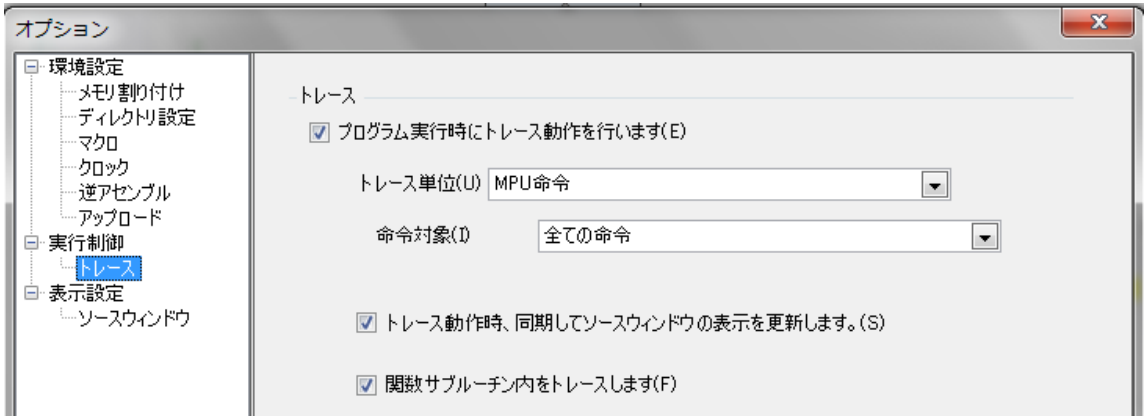


# 実習1-6：func1()　トレースモードを使う

## ■ 実行箇所をカーソル表示する「トレースモード」をONにします

- SystemSimulatorの「トレースモード」をONにした場合：
  - 実行時間に負荷がかかる（実行が遅くなる）
  - プロジェクトフォルダに、実行ログファイルが生成される（systemg.log）
    - ➢ ログファイルは、シミュレータの実行エラーのトラブルシューティングに利用

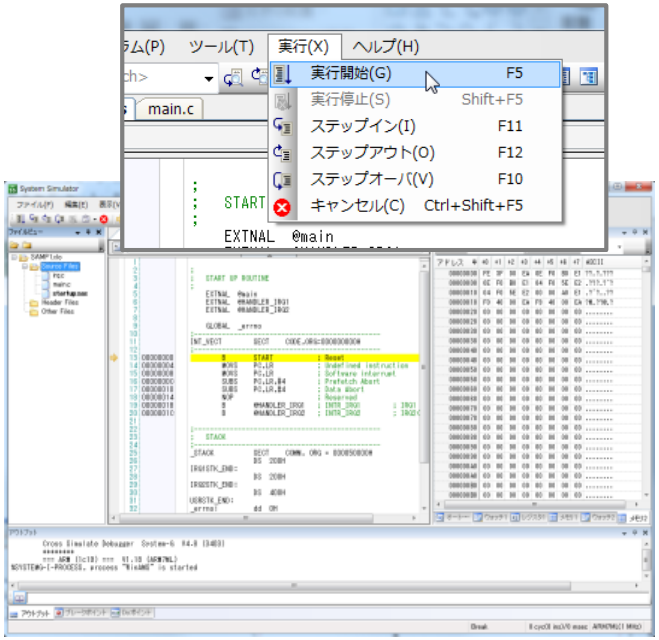
1. ウィンドウ上部の「シミュレータ起動」ボタンを押します
2. マイコンシミュレータ（ISS）とwinAMSユニットが起動します
3. SystemSimulatorの「ツール」メニューから「オプション」を選択します
4. 下図の通り、トレースオプションを全てONにします



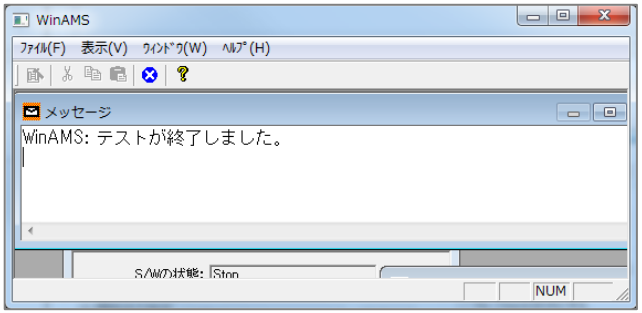
# 実習1-6：func1()テスト実行（続き）

## ■ テストをSystemSimulator（デバッガ）で実行をします

1. winAMSユニットウインドウから「メッセージ」ウインドウを最大化します
2. マイコンシミュレータの「実行」メニューから「実行開始」を選択します




マイコンシミュレータ（ISS）




winAMSユニット

# 実習1-6：func1()テスト実行（続き）

## ■ シミュレータを終了します

1. winAMSユニットのメッセージに「WinAMS：テストが終了しました」が表示されます
2. マイコンシミュレータ（ISS）の「ファイルメニュー」→「終了」を選択します
3. 終了確認ダイアログで「OK」を押します

## ■ 結果CSVファイルを確認します

1. SSTManagerウインドウ左側の「テスト結果」ボタンを押して
   テスト結果を表示します

テスト結果情報

| テスト結果CSVファイル | テストタイトル | テスト関数名 | 判定 |
|---|---|---|---|
| func1_data.csv | | func1 | No Check |

2. 「func1_data.csv」をダブルクリックします
3. 結果CSVファイルが開きます

※入力CSVとファイル名は同一ですが、別のフォルダに生成されています。

# 実習1-7：func1()単体テスト結果の確認

## ■ テスト結果を確認します

- 出力変数「gb_result.data, gb_result.ret_code」に関数実行結果が入ります
- 入力ファイルに期待値を設定すれば、この結果と照合して「OK/NG」をレポートします

入力データがそのままコピーされます

実行結果

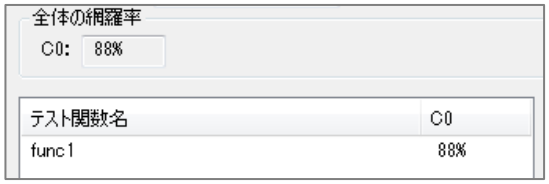
期待値は設定していないので「チェックしていない」の意味

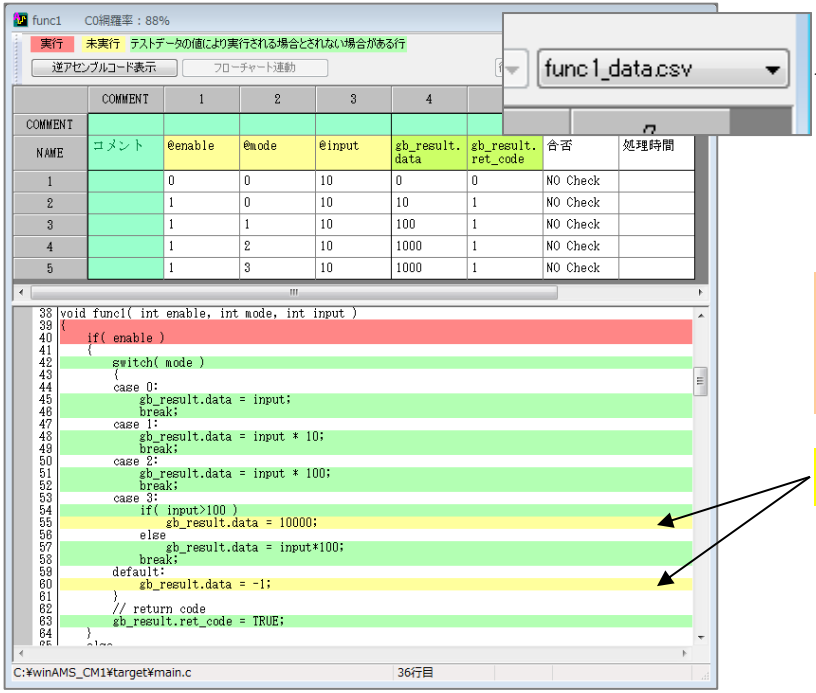
| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| 1 | mod | func1 | | 3 | 2 | | |
| 2 | #COMMENT | @enable | @mode | @input | gb_result.data | gb_result.ret_code | |
| 3 | | 0 | 0 | 10 | 0 | 0 | NO Check |
| 4 | | 1 | 0 | 10 | 10 | 1 | NO Check |
| 5 | | 1 | 1 | 10 | 100 | 1 | NO Check |
| 6 | | 1 | 2 | 10 | 1000 | 1 | NO Check |
| 7 | | 1 | 3 | 10 | 1000 | 1 | NO Check |

# 実習1-8：func1()カバレッジの確認

## ■ パスカバレッジ結果を確認します

1. 左側の「カバレッジ」ボタンを押します。
2. C0網羅率 「88%」が確認できます。
3. 「func1」をダブルクリックして カバレッジビューを表示します
4. 網羅率100%に不足しているテストケースを確認します


func1_data.csvを選択
**※必ずEXCELは閉じてから**

**<不足データ>**
**●input > 100　のケース**
**●modeが0～3以外（default）のケース**

**黄色の行が、テスト未実行の行**

※白の行(色なし)は、実行するアセンブルコードがない行
カバレッジ換算からは外されます

# 実習1-9：func1()再テスト

■ **網羅率を100%にするために テストケースを追加して再テストします**

1. 「実習1-5」の入力データに下の2つのケースを追加します
   - 「テスト設定」で「func1_data.csv」を編集
   - ※前回の実行結果を期待値として設定

期待値を入力
※「99」は実行結果が異なる設定

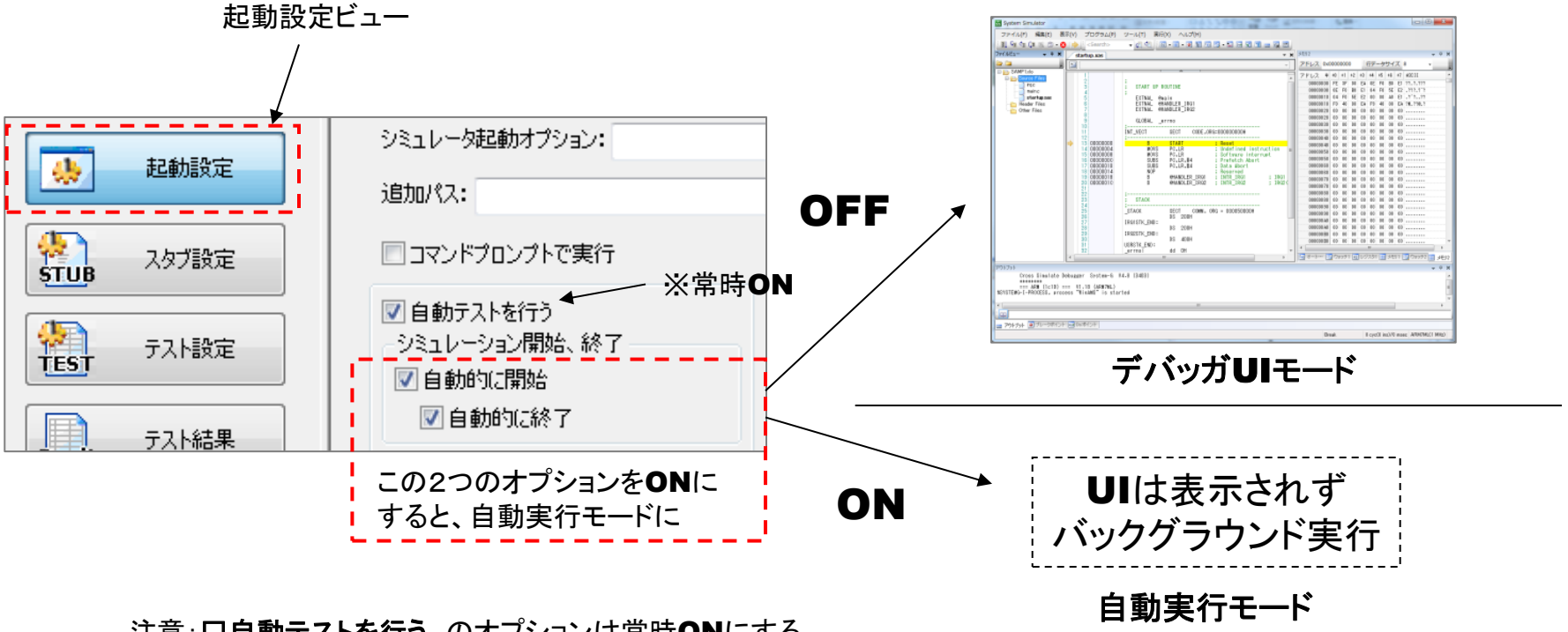
カバレッジを100%にする追加データ

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | mod | func1 | | 3 | 2 | |
| 2 | #COMMENT | @enable | @mode | @input | gb_result.data | gb_result.ret_code |
| 3 | | 0 | 0 | 10 | 0 | 0 |
| 4 | | 1 | 0 | 10 | 10 | 1 |
| 5 | | 1 | 1 | 10 | 99 | 1 |
| 6 | | 1 | 2 | 10 | 1000 | 1 |
| 7 | | 1 | 3 | 10 | 1000 | 99 |
| 8 | | 1 | 3 | 200 | | |
| 9 | | 1 | 4 | 10 | | |

# （参考）自動実行モードへの切り換え

## ■ 実行モードの切り替えが可能

- デバッガUIモード：　デバッガGUIを使用してマニュアル操作
- 自動実行モード：　バックグラウンドで実行（操作不要で起動するだけで結果が表示される）

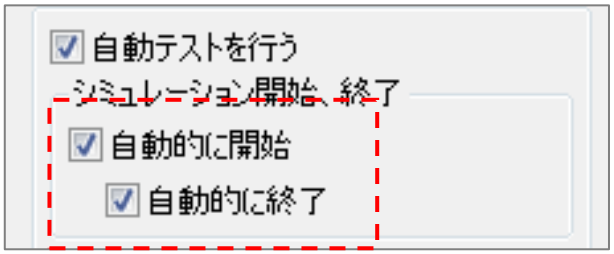



注意：**□自動テストを行う**　のオプションは常時**ON**にする
このオプションは、単体テスト自体の**ON/OFF**を行うもの

# 実習1-10：自動実行モードでfunc1再テスト

## ■ 自動実行（UIを使用しない）モードで、func1を再テストします

1. 「起動設定」ビューの「シミュレーション開始終了」の2つのオプションをONにします
   - 自動実行モードに切り替わります




2. 「シミュレータ起動ボタン」でテスト実行を再度行います
   ※シミュレータ実行の前に、入力CSVファイルは必ず閉じて下さい。
3. 「実習1-8」のパスカバレッジの網羅率が「100%」になっていることを確認します

# （参考）CSVファイルにコメントを記載する方法

## ■ CSVファイルにテスト項目などのコメントを記述することが可能

- #COMMENT の列に「;」（セミコロン）を入れると その行はコメント扱い
- テストケースに使用されている再右列の右側に「;」（セミコロン）を入れてコメント記載可能

| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| 1 | mod | func1 | | 3 | 2 | | |
| 2 | #COMMENT | @enable | @mode | @input | gb_result.data | gb_result.ret_code | |
| 3 | | 0 | 0 | 10 | 0 | 0 | ;異常条件 |
| 4 | | 1 | 0 | 10 | 10 | 1 | ;正常条件 |
| 5 | | 1 | 1 | 10 | 100 | 1 | |
| 6 | | 1 | 2 | 10 | 1000 | 1 | |
| 7 | | 1 | 3 | 10 | 1000 | 1 | |
| 8 | ; | テストケースを追加↓ | | | | | |
| 9 | | 1 | 3 | 200 | | | ;追加(2012/7/6) |
| 10 | | 1 | 4 | 10 | | | ;追加(2012/7/6) |

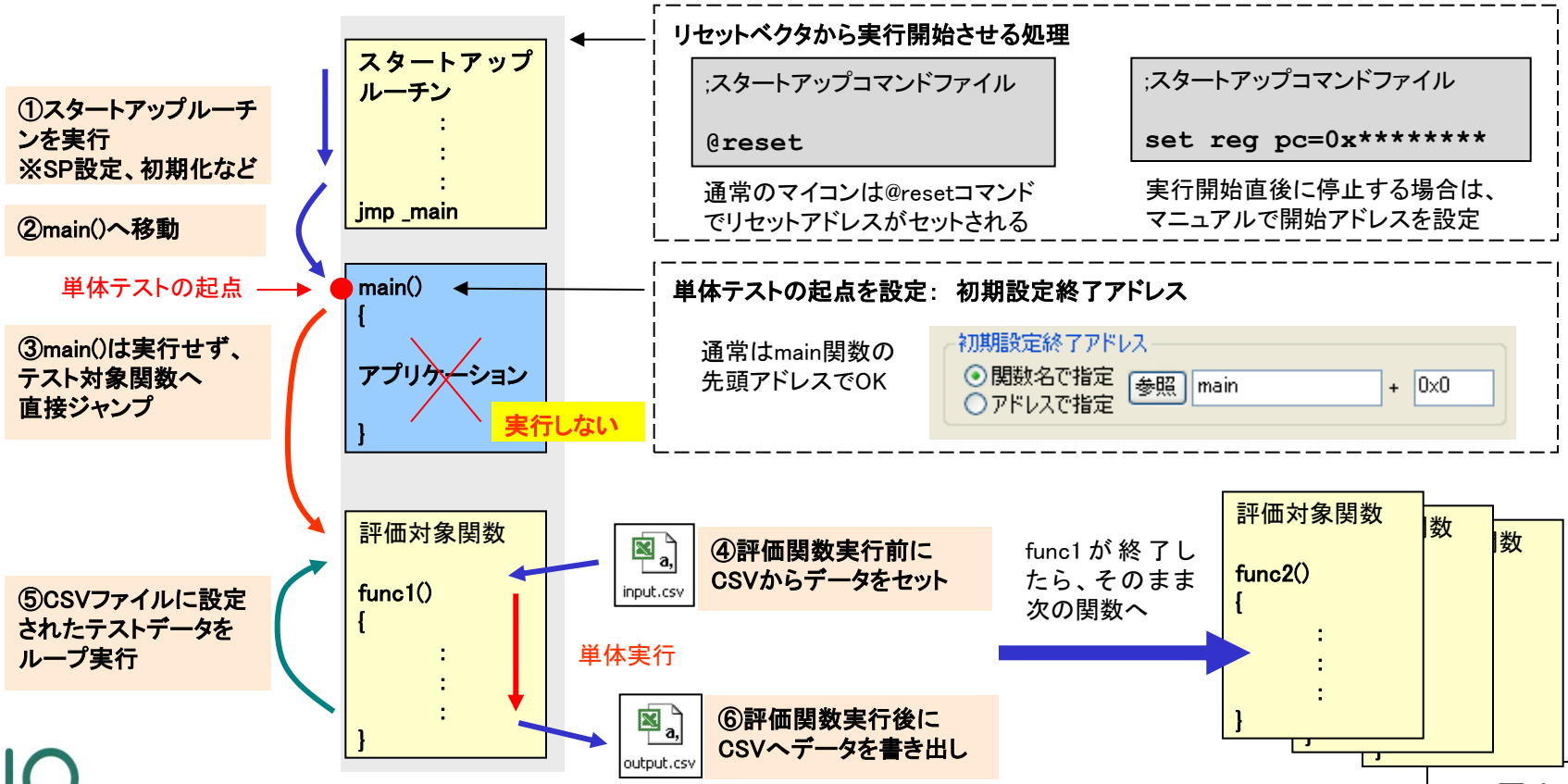
「;」（セミコロン）を入れると その行はコメント扱い

再右列の右側に「;」（セミコロン）
を入れてコメント記載可能

# （参考）:マイコンリセットからの単体テストへの動作

## ■ マイコンシミュレータ(ISS)は 実マイコンと同じ手順で動作する

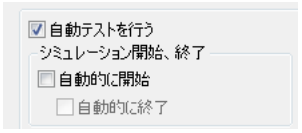
- 各関数を単体で実行するためには、スタートアップルーチンを動作させる必要がある
  - スタックポインタ設定、グローバル変数初期化などの main()実行前の動作



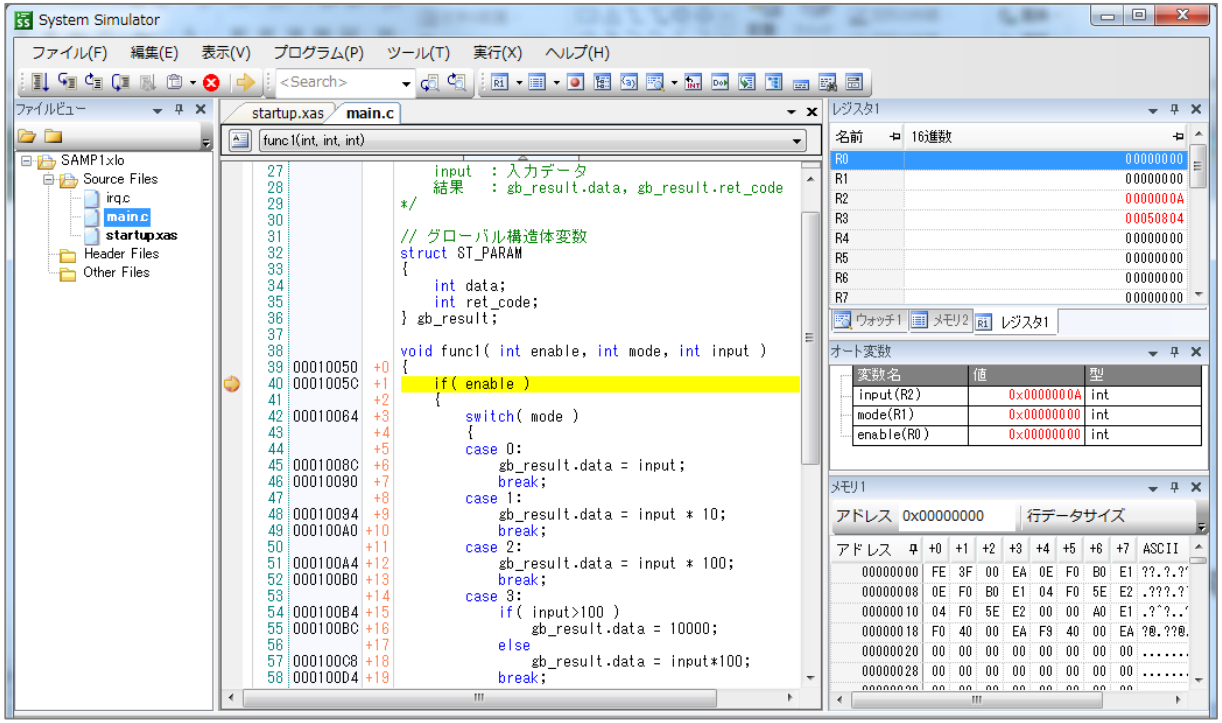
# （参考）：マイコンシミュレータのデバッグ機能

## ■ マイコンシミュレータのソースコードデバッグ機能を活用する

- 単体テストで問題が発生した場合、ソースコードデバッグで検出が可能
  - 自動実行モードを外す：「シミュレーションを自動的に開始」をオフにする
  - 対象関数にブレークポイントを置く（注意：関数の先頭行にはブレークは設定不可）
  - ウオッチウインドウに変数をドラッグ＆ドロップで登録可能




自動実行モードを外す
とデバッガ**GUI**の
使用が可能



# 実習2
# ポインタ変数を持つ関数の単体テスト

ポインタの実体をカバレッジマスターの機能で
自動割り当て

# 実習2-1：ポインタ変数を持つ関数の単体テスト

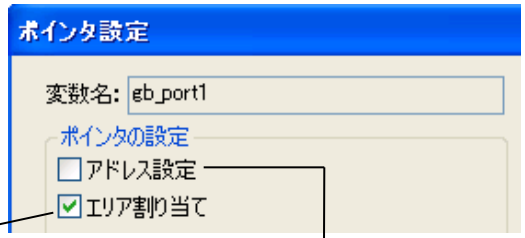
■ 実習2は サンプルソースを使用して、ポインタ変数を持つ関数へのデータの与え方を学習します。

– サンプルソース（main.c）の func2()を単体テストします。

```
// 実習2） ポインタ変数引数の関数サンプル
//           C0カバレッジを100%にする

char *gb_port1 ;  // i/oポート入力
int  *gb_data_out; // 結果

void func2( int mode, int *data_in )
{
     if( *gb_port1 & 0x00000001 ) // 最下位ビットが1のとき
     {
          switch( mode)
          {
          case 0:
               *gb_data_out= *data_in ;
               break;
          case 1:
               *gb_data_out= *data_in * 10;
               break;
          case 2:
               *gb_data_out= *data_in * 100;
               break;
          case 3:
               if( *data_in > 100 )
                    *gb_data_out= 10000;
               else
                    *gb_data_out= *data_in *100;
               break;
          default:
               *gb_data_out= -1;
          }
     }
     else
     {
          *gb_data_out= 0;
     }
}
```

# （参考）ポインタ変数にはアドレスとテストケースが必要

■ （例）char *gb_port1;
- ①ポインタ変数gb_port1のアドレスを設定する
- ②ポインタ変数の指す実体にテストケースを入れる




自習ではこの方法を使用

## ポインタの自動割り付け機能を使用する方法

①＄マーク　→　エリア割り当て
（シミュレータにポインタを自動割付の機能がある）

②添え字で実体を指定

| | A | B | C |
|---|---|---|---|
| 1 | mod | func2 | |
| 2 | #COMMENT | $gb_port1 | gb_port1[0] |
| 3 | | 1 | 0x1 |
| 4 | | 1 | 0x0 |

ポインタに割り付ける
実体の個数を入力

②テストケースを設定

※0を入力すると
ポインタはNULLになる

## （参考）ポインタにアドレスを直接指定する方法

①ポインタ変数を直接指定

②添え字で実体を指定

| | A | B | C |
|---|---|---|---|
| 1 | mod | func2 | |
| 2 | #COMMENT | gb_port1 | gb_port1[0] |
| 3 | | 0x10000000 | 0x1 |
| 4 | | func1 | 0x0 |

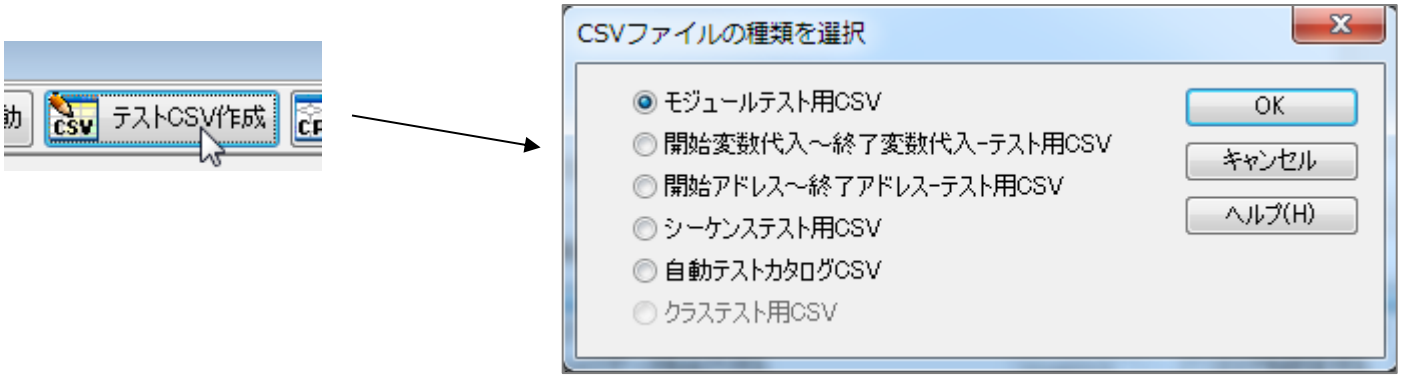
アドレスを直接入力
※アドレス値の代わりに
シンボル名を使用可能

②テストケースを設定

# 実習2-2：func2()テスト用CSV雛形を作成

- テスト対象は 関数func2() です　　※「実習2-1」のソースコードを参照
- func2()の入力データと出力データの変数名を確認します
  - 入力変数【3個】：　char *gb_port1（グローバル　ポインタ変数），
    int mode,　int *data_in（引数）
  - 出力変数【1個】：　int　*gb_data_out;（グローバル　ポインタ変数）

- モジュールテスト用CSV作成機能を使用して 入出力変数名を指定
  1. 上部の「テストCSV作成」ボタンを押します
  2. 「モジュールテスト用CSV」を選択して「OK」を押します



# 実習2-2：func2()テスト用CSV雛形を作成（続）

## ■ モジュールテスト用雛形作成機能で、テスト関数名・変数名などを作成

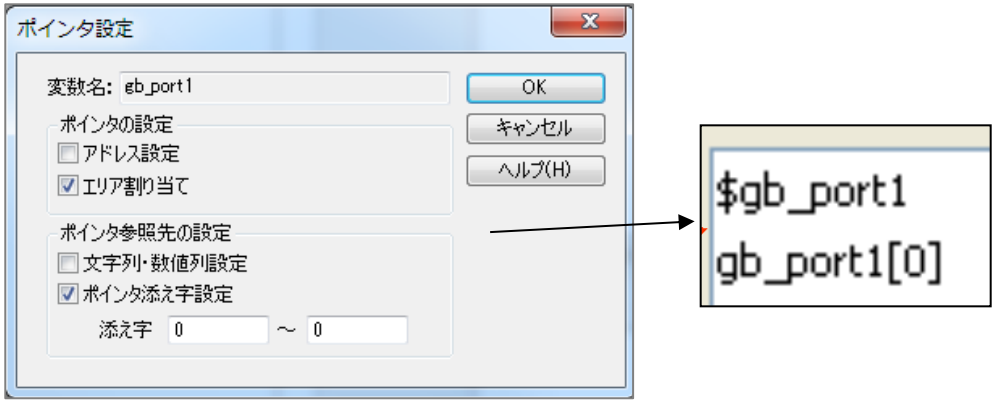
1. ファイル名「func2_data」、関数名「func2」を指定　（テストタイトルは任意）
2. 変数INPUT/OUTPUT ツリービューから 変数を選択して登録
   - ポインタ変数は、「エリア割り当て」を選択して、「実体の個数」を指定する変数を作成
   - ポインタ変数の「実体の値」を指定する変数を作成

**$**

（例）char *gb_port1の場合
この変数は配列ではなく、1個のchar実体を持つポインタなので

【1】エリア割り当てで
「$gb_port1」を作成
（このCSVではこの変数に、実体の個数を指定）

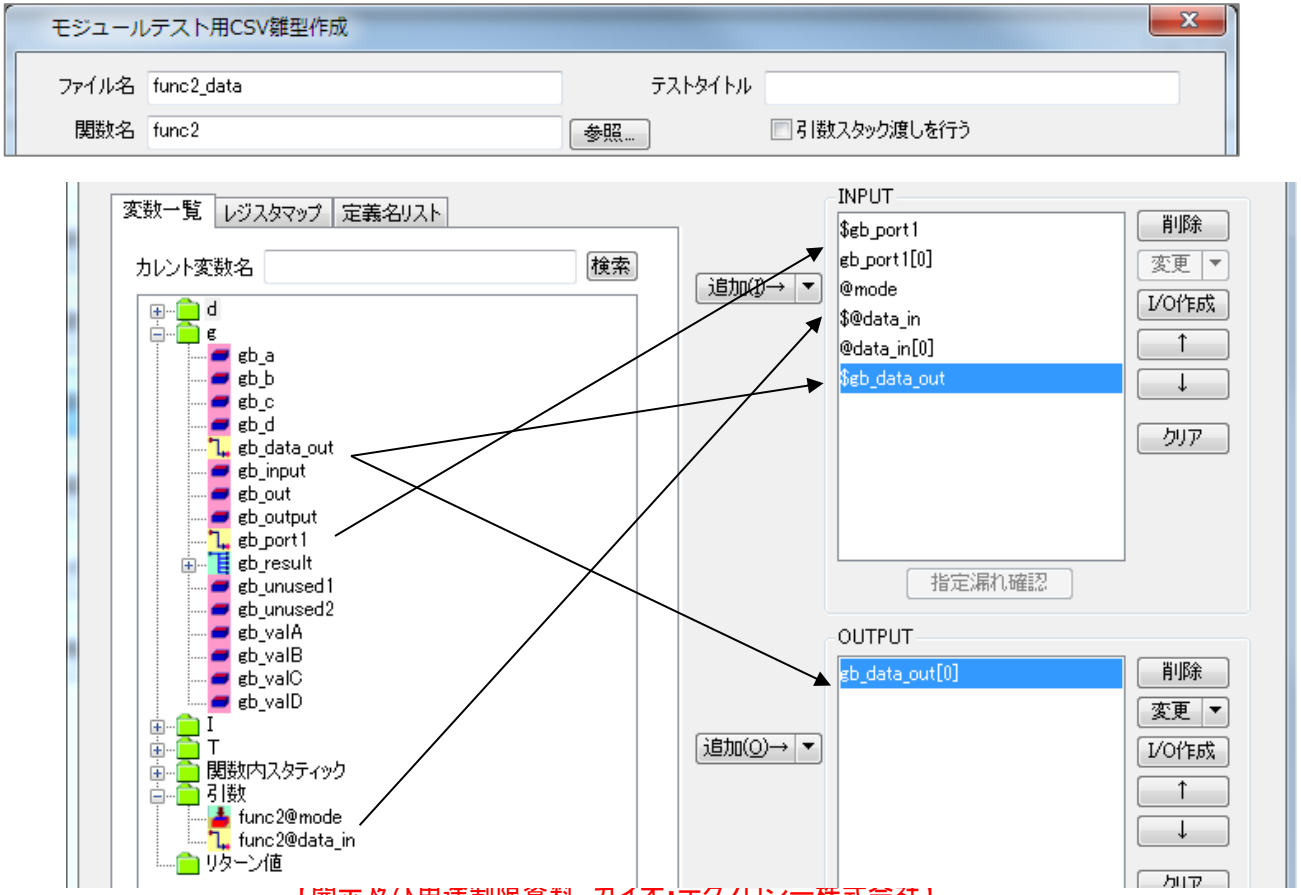
【2】ポインタ添え字設定で
0～0の添え字で、1個の実体を指定する
gb_port1[0]



# 実習2-2：func2()テスト用CSV雛形を作成（続）

■ ポインタ変数は INPUT側に「エリア割り当て」の変数を作成する

– 【注意】出力変数 *gb_data_out　のエリア割り当ては、入力条件なので INPUT側に！



# 実習2-3：func2()雛形にテストデータを追加

## ■ テストケースを追加します

1. $の付いた変数 $gb_port1, $@data_in, $gb_data_out には1を指定します
2. 今回は「期待値」を設定します

期待値

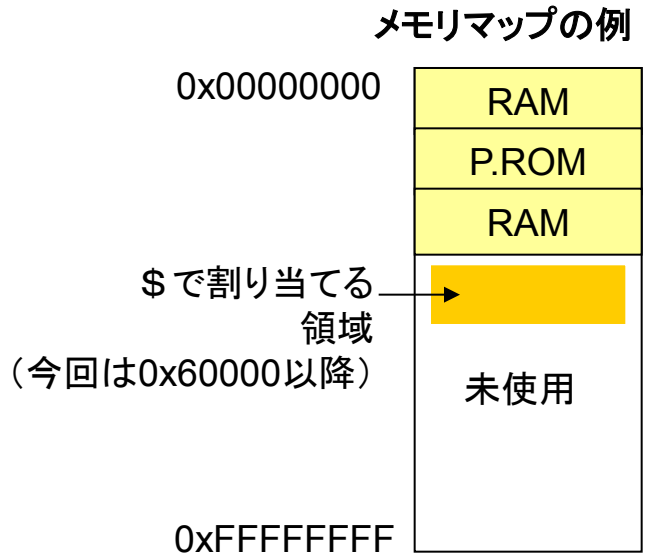
| | A | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|---|
| 1 | mod | func2 | | 6 | 1 | | | | CPP |
| 2 | #COMMENT | $gb_port1 | gb_port1[0] | @mode | $@data_in | @data_in[0] | $gb_data_out | gb_data_out[0] | |
| 3 | | 1 | 0 | 0 | 1 | 10 | 1 | 0 | |
| 4 | | 1 | 0 | 0 | 1 | 10 | 1 | 0 | |
| 5 | | 1 | 1 | 0 | 1 | 10 | 1 | 10 | |
| 6 | | 1 | 1 | 1 | 1 | 10 | 1 | 100 | |
| 7 | | 1 | 1 | 2 | 1 | 10 | 1 | 1000 | |
| 8 | | 1 | 1 | 3 | 1 | 10 | 1 | 1000 | |
| 9 | | 1 | 1 | 3 | 1 | 200 | 1 | 10000 | |
| 10 | | 1 | 1 | 4 | 1 | 10 | 1 | -1 | |

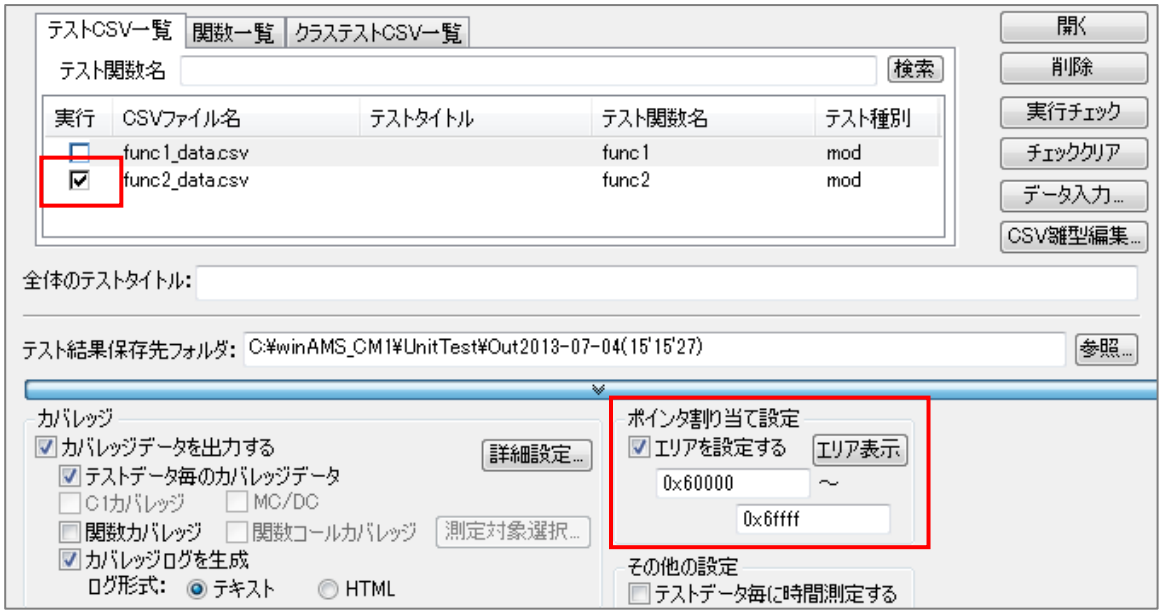
$の付いた変数：　実体の個数

# 実習2-3：func2()テスト実行

## ■ 実行のための設定（ポインタ割当てエリア設定）を追加します

1. 「CSVファイル一覧」で「func2_data.csv」の左のチェックボックスをONにします
2. 「ポインタ割り当てエリア設定」欄の「エリア設定をする」をONにします
3. エリアに、「0x60000 ～ 0x6ffff」を指定します
   - 使用マイコンのメモリマップ上で、未使用の領域をアサインする



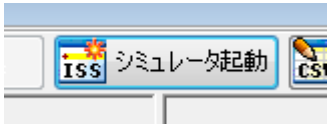



# 実習2-3：func2()テスト実行（続き）

## ■ テスト実行（シミュレータ起動）します

- 自動実行モードで実行され結果が表示されます

SSTManager




## ■ テスト結果（func2_data.csv）の確認を行います

- 結果確認方法は、「実習1-6(続き)」 を参照
- 結果CSV右側（9列目）がすべて「OK」の判定になります
- ポインタのアドレスが解決され、実行できたことが確認できます

| | A | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|---|
| 1 | mod | func2 | | 6 | 1 | | | | CPP |
| 2 | #COMMENT | $gb_port1 | gb_port1[0] | @mode | $@data_in | @data_in[0] | $gb_data_out | gb_data_out[0] | |
| 3 | | 1 | 0 | 0 | 1 | 10 | 1 | 0 | OK |
| 4 | | 1 | 0 | 0 | 1 | 10 | 1 | 0 | OK |
| 5 | | 1 | 1 | 0 | 1 | 10 | 1 | 10 | OK |
| 6 | | 1 | 1 | 1 | 1 | 10 | 1 | 100 | OK |
| 7 | | 1 | 1 | 2 | 1 | 10 | 1 | 1000 | OK |
| 8 | | 1 | 1 | 3 | 1 | 10 | 1 | 1000 | OK |
| 9 | | 1 | 1 | 3 | 1 | 200 | 1 | 10000 | OK |
| 10 | | 1 | 1 | 4 | 1 | 10 | 1 | -1 | OK |

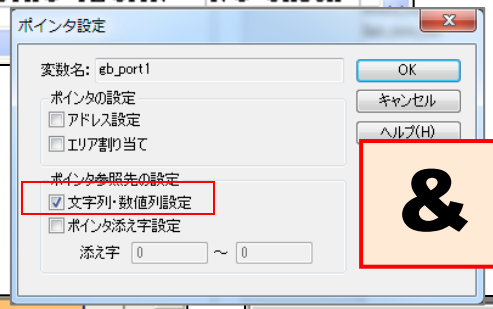
# 【参考】文字列・数字列のCSVファイル指定

## ■ 文字列データ　：　' '（シングルクォーテーション）で囲む

```
/*
	文字列データ
*/
char outStr[64];

void StringTest( int limit )
{
	outStr[limit] = '¥0';
}
```

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | mod | StringTest | | 2 | 1 |
| 2 | #COMMENT | &outStr | @limit | &outStr | |
| 3 | | 'GAIO TECHNOLOGY' | 4 | 'GAIO' | NO Check |
| 4 | | 'GAIO TECHNOLOGY' | 6 | 'GAIO T' | NO Check |
| 5 | | 'GAIO TECHNOLOGY' | 8 | 'GAIO TEC' | NO Check |
| 6 | | 'GAIO TECHNOLOGY' | 10 | 'GAIO TECHN' | NO Check |

StingTest




## ■ 数値列データ　：　｜　で区切る

- 評価対象の場合は、出力したい数値列の個数分－１の「｜」を期待値欄に入れる

```
int ArrayTable[10];

void ArrayTest( int index )
{
	int i;
	if( index>10 ) index=10;
	for( i=0; i<index; i++ )
	{
		ArrayTable[i] = 0;
	}
}
```

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | mod | ArrayTest | | 2 | 1 |
| 2 | #COMMENT | &ArrayTable | @index | &ArrayTable | |
| 3 | | 1\|2\|3\|4\|5\|6\|7\|8\|9 | 2 | \|\|\|\|\|\|\|\|\| | |
| 4 | | 1\|2\|3\|4\|5\|6\|7\|8\|10 | 4 | \|\|\|\|\|\|\|\|\| | |
| 5 | | 1\|2\|3\|4\|5\|6\|7\|8\|11 | 6 | \|\|\|\|\|\|\|\|\| | |
| 6 | | 1\|2\|3\|4\|5\|6\|7\|8\|12 | 8 | \|\|\|\|\|\|\|\|\| | |

ArrayTest

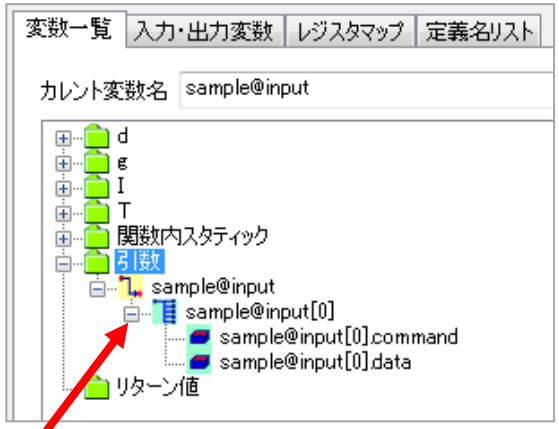
| D |
|---|
| 2 |
| &ArrayTable |
| 0\|0\|3\|4\|5\|6\|7\|8\|9 |
| 0\|0\|0\|0\|5\|6\|7\|8\|10 |
| 0\|0\|0\|0\|0\|0\|7\|8\|11 |
| 0\|0\|0\|0\|0\|0\|0\|0\|12 |

# 【参考】構造体ポインタのメンバー変数設定

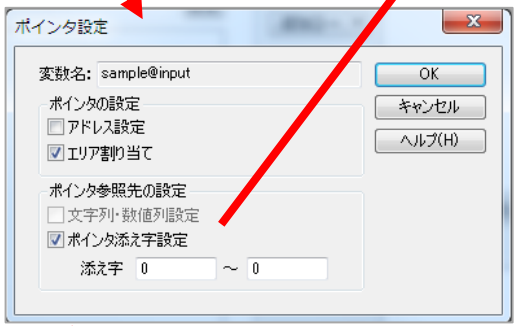
## ■ 構造体ポインタのメンバー変数指定方法

- 最初は、構造体ポインタは、ポインタ変数として表示されている
- 構造体の実体を作成する必要あり
- 添え字[0]を指定して実体を作成すると、その下にメンバーが表示される

```
typedef struct _TAG_PTRIN
{
    unsigned char command;
    unsigned int data;
} PtrIn;

void sample( PtrIn *input )
{
    if( input->command == 0x01 )
    {
        input->data = 0xFF00;
    }
    else
    {
        input->data = 0x00FF;
    }
}
```


メンバーが表示される



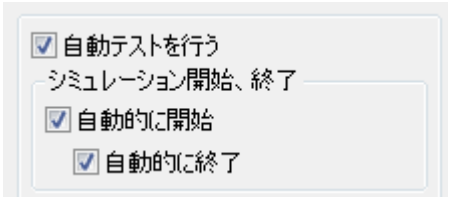

ポインタ添え字設定で、
添え字**[0]**の構造体の
実体を作成

# （参考）：自動実行モードのマイコンシミュレータ

## ■ 自動実行モード時には マイコンシミュレータは表示されない

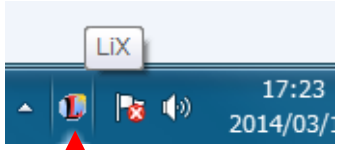
- 自動実行モード時にマイコンシミュレータが停止（ホールト）した際の対処方法
  - Windowsタスクバーから「L」マークのアイコンを右クリック
  - 「表示」メニューを選択
  - 高速モード時のマイコンシミュレータ「Lix」が表示される
  - ファイルメニューから「終了」を選択すると、SSTManagerへ戻ることが可能

【注意】Windows強制終了は使用しない！

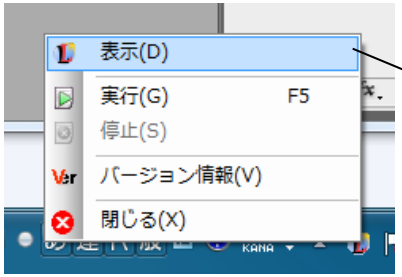


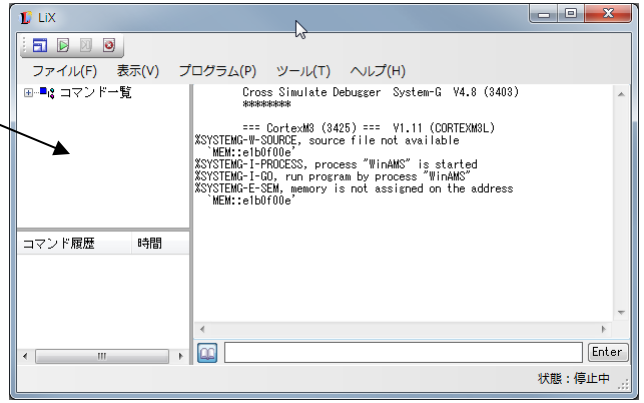

自動実行モード時

Windowsタスクバー




「Lix」



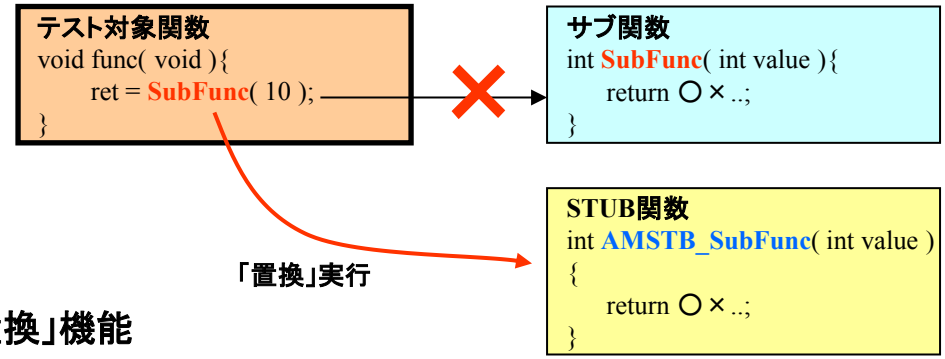

マイコンシミュレータ「Lix」

# 実習3
# スタブ機能を使用して
# サブ関数を置換

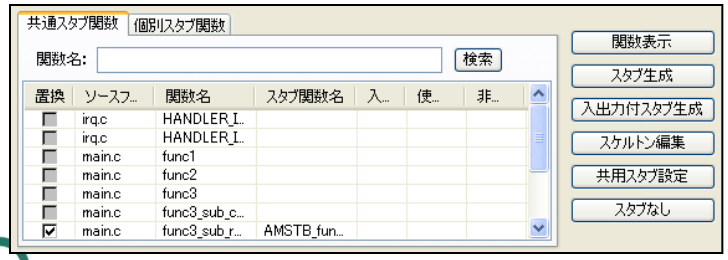
カバレッジマスターのスタブ関数置換機能で
サブ関数コールを持つ関数をテスト

# （参考）サブ関数のSTUB化について

- **単体テストのフェーズでは 関数を独立させて単体で評価する**
  - 実物のサブ関数を使用すると、結合テストとなってしまう
- **テスト対象関数とサブ関数の依存関係を無くすためにSTUBを作成**
  - 下位関数の変更が、上位関数に影響しない様にする
  - テストの効率化、分業化を促進する




**カバレッジマスターのSTUB「置換」機能**



# 実習3-1：スタブ機能でサブ関数を置換

## ■ 実習3は スタブ機能を使用して サブ関数を置換します

– サンプルソース（main.c）のfunc3()を使用します

```
// 戻り値を返す関数（スタブする）
int func3_sub_read_io( int index )
{
    // 戻り値がdata_tableに依存して複雑
    if( data_table[index]>0x7f )
    {
        return data_table[index];
    }
    else
    {
        return -data_table[index];
    }
}
```

スタブ関数を作成

```
int AMSTB_func3_sub_read_io( int index )
{
    static int ret;
    return ret;
}
```

この変数にCSVからデータ（戻り値）を指定できる様にする

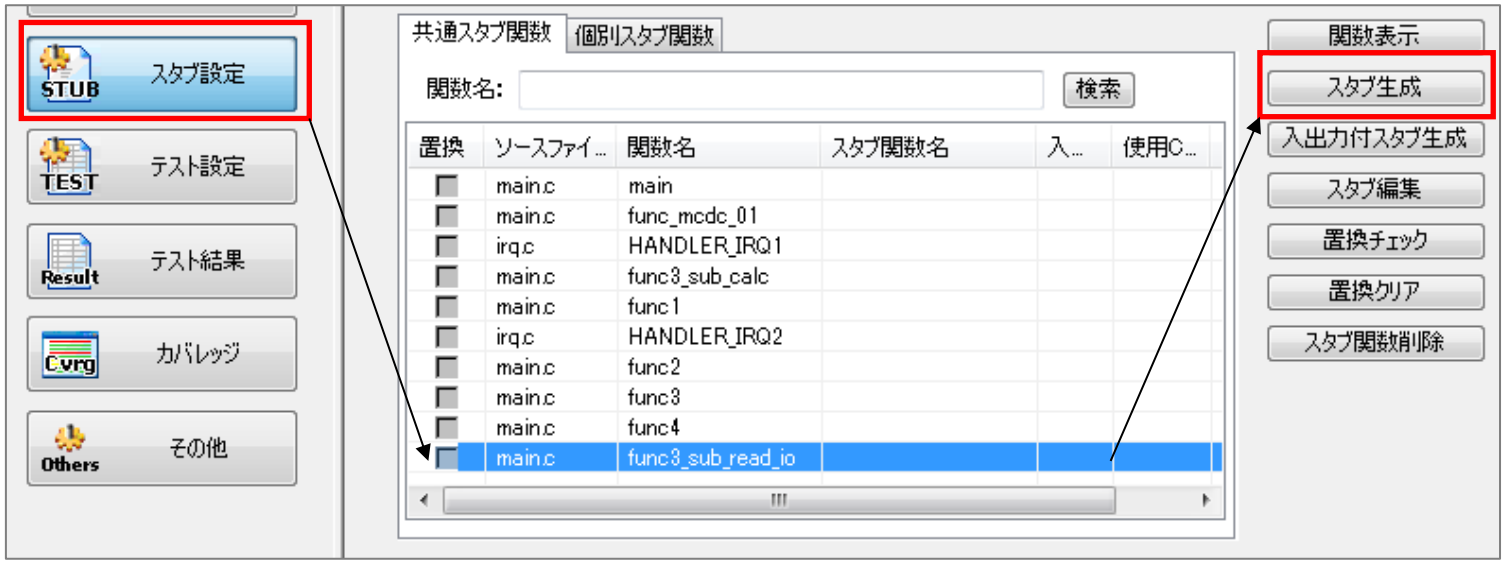
```
void func3( int enable, int mode )
{
    int retval;

    if( enable )
    {
        // スタブを作成して 戻り値を自由に設定する
        retval = func3_sub_read_io( mode );

        // 戻り値を使って分岐
        switch( retval )
        {
            case 0:
                gb_result.data = 0;
                break;
            case 1:
                gb_result.data = 50;
                break;
            case 2:
                gb_result.data = 100;
                break;
            default:
                gb_result.data = -1;
        }
        gb_result.ret_code = TRUE;
    }
    else
    {
        gb_result.data = 0;
        gb_result.ret_code = FALSE;
    }
}
```

# 実習3-2: スタブの指定と作成

## ■ スタブ設定で func3_sub_read_io()をスタブします

この関数のスタブ関数を作成して、呼び出し時に入れ替えて実行するように設定します

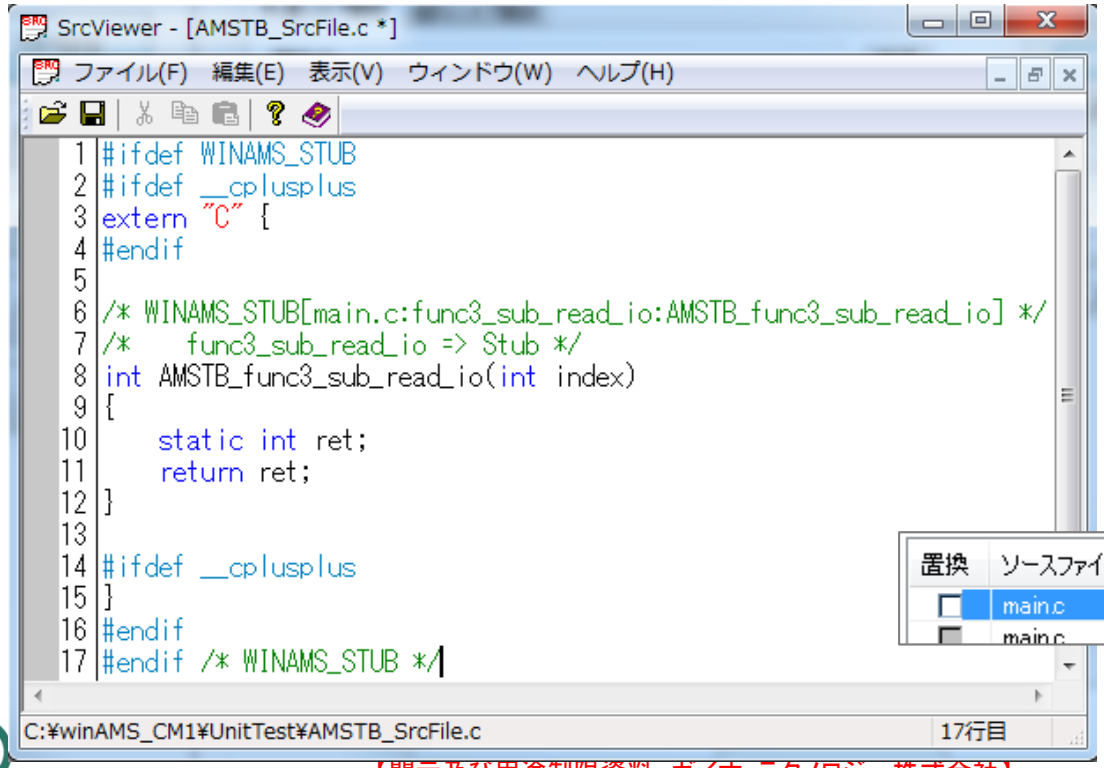
1. 「スタブ設定」のボタンを押します
2. func3_sub_read_io　を選択して「スタブ生成」を押します
   - スケルトンのスタブ関数「AMSTB_func3_sub_read_io」が作成されます
   - この関数は別ファイル（AMSTB_SrcFile.c）に作成されます

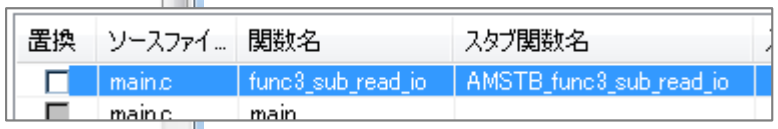


# 実習3-3: スタブの指定と作成（続）

## ■ スタブ関数を作成します

1. 作成された AMSTB_func3_sub_read_io()に、以下のコードを追加します。
   - 関数内スタティック変数「ret」を、そのままリターン
2. ファイル（AMSTB_SrcFile.c）を保存します

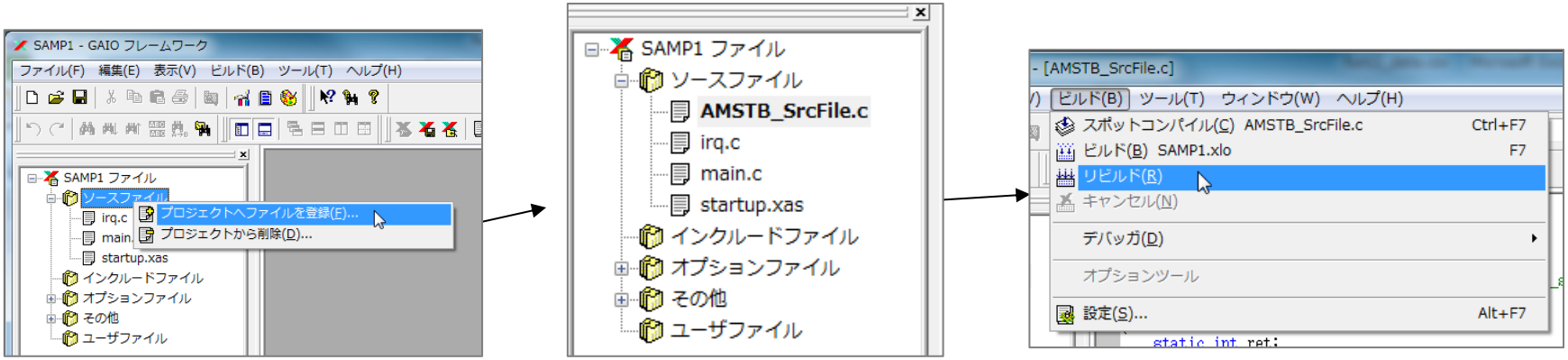


# 実習3-4: スタブの指定と作成（続）

## ■ スタブ関数のソースファイルをコンパイル＆リンクします

1. ガイオ開発環境「フレームワーク」を起動します
   - P.44を参照 「SAMP1.gxp」を開きます
2. ファイル管理ビューの「ソースファイル」を右クリックして「プロジェクトへファイルを登録」を選択します
3. 「AMSTB_SrcFile.c」を登録します
4. 「ビルドメニュー」から「ビルド」を選択してビルドします
   - 評価オブジェクトに 作成したスタブ部分がリンクされます

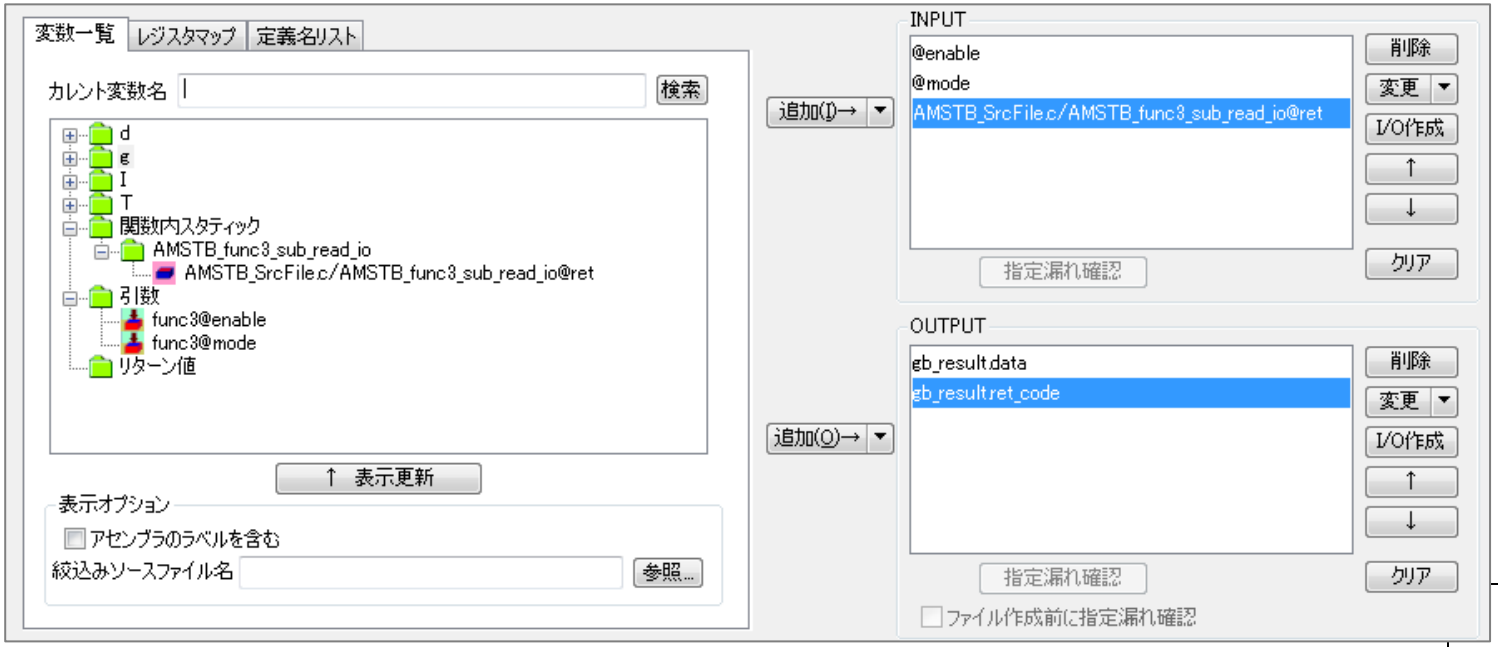



※上は、ガイオ・テクノロジーのコンパイラを使用している場合です。
実行オブジェクトのコンパイル＆リンクの方法は、皆様がお使いの開発環境に依存します。

# 実習3-5: func3テストCSVの雛形を作成

## ■ func3()の入出力条件を確認します

- 入力変数：　引数　int enable, int mode
  - STUBの戻り値を設定する変数を入力変数に追加
    - 関数内スタティック　ret
- 出力（評価）変数
  - グローバル変数　gb_result.data, gb_result.ret_code



# 実習3-6: func3テストデータを入力

## ■ C0を100%にするテストデータを設定します

- @modeはSTUBでは使われないが、適当な値を設定する必要有り
- ret には、戻り値を設定
  - switch文の条件を全て網羅する値を設定

（参考）

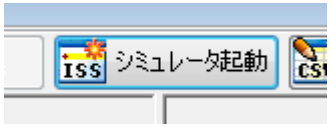
テストで使用されない変数（分岐パスにより処理に係わらない変数）には、
テスト仕様書で予め決めた値を設定（111など） ←データのレビュー工数を削減

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | mod | func3 | | 3 | 2 | |
| 2 | #COMMENT | @enable | @mode | AMSTB_SrcFile.c/AMSTB_func3_sub_read_io@ret | gb_result.data | gb_result.ret_code |
| 3 | | 0 | 111 | 111 | | |
| 4 | | 1 | 111 | 0 | | |
| 5 | | 1 | 111 | 1 | | |
| 6 | | 1 | 111 | 2 | | |
| 7 | | 1 | 111 | 3 | | |

# 実習3-7: スタブの指定、テスト実行

## ■ スタブ関数に置換したい サブ関数のスイッチを設定します

1. func3_sub_read_io の行の置換ボックスをチェックします
   - これで、func3_sub_read_io()が呼ばれると、AMSTB_func3_sub_read_io()に自動的に置換されて実行されます
     **※ テスト対象関数のコードを変更する必要はありません**
2. テスト実行します。 実行方法は「実習1-6」を参照。


# 実習3-8: スタブの指定、テスト実行（続）

## ■ 実行結果を確認します

1. テスト結果の「func3_data.csv」を確認します

| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| 1 | mod | func3 | | 3 | 2 | | |
| 2 | #COMMENT | @enable | @mode | AMSTB_SrcFile.c/AMSTB_func3_sub_read_io@ret | gb_result.data | gb_result.ret_code | |
| 3 | | 0 | 111 | 111 | 0 | 0 | NO Check |
| 4 | | 1 | 111 | 0 | 0 | 1 | NO Check |
| 5 | | 1 | 111 | 1 | 50 | 1 | NO Check |
| 6 | | 1 | 111 | 2 | 100 | 1 | NO Check |
| 7 | | 1 | 111 | 3 | -1 | 1 | NO Check |

2. カバレッジビューで、スタブ関数が実行されていることを確認します。

| テスト関数名 | C0 |
|---|---|
| func3 | 100% |

| その他の関数 | C0 |
|---|---|
| AMSTB_func3_sub_read_io | 100% |

# （参考）スタブ関数自動生成機能について

## ■「入出力付スタブ生成」のボタンでスタブ関数を自動生成

- スタブに自動実装される機能
  - 戻り値がある場合に、戻り値をCSVファイルで指定する機能
  - 渡された引数をCSVファイルのOUTPUTに出力する機能


自動生成される
スタブ関数

```
 9 /* WINAMS_STUB[main.c:func3_sub_read_io:
10 /*    func3_sub_read_io => Stub */
11 int AMSTB_func3_sub_read_io(int index)
12 {
13     static int volatile AMIN_return;
14     static int volatile AMOUT_index;
15     AMOUT_index = index;
16     return AMIN_return;
17 }
```

変数：AMIN_return
→CSVのINPUTに登録

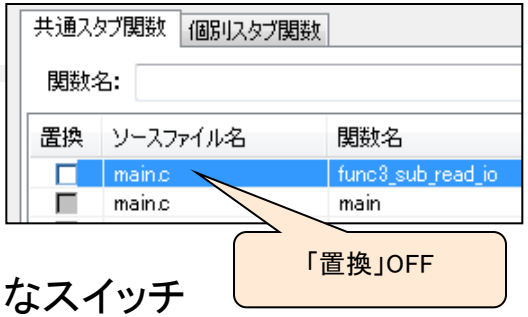
変数：AMOUT_index
→CSVのOUTPUTに登録

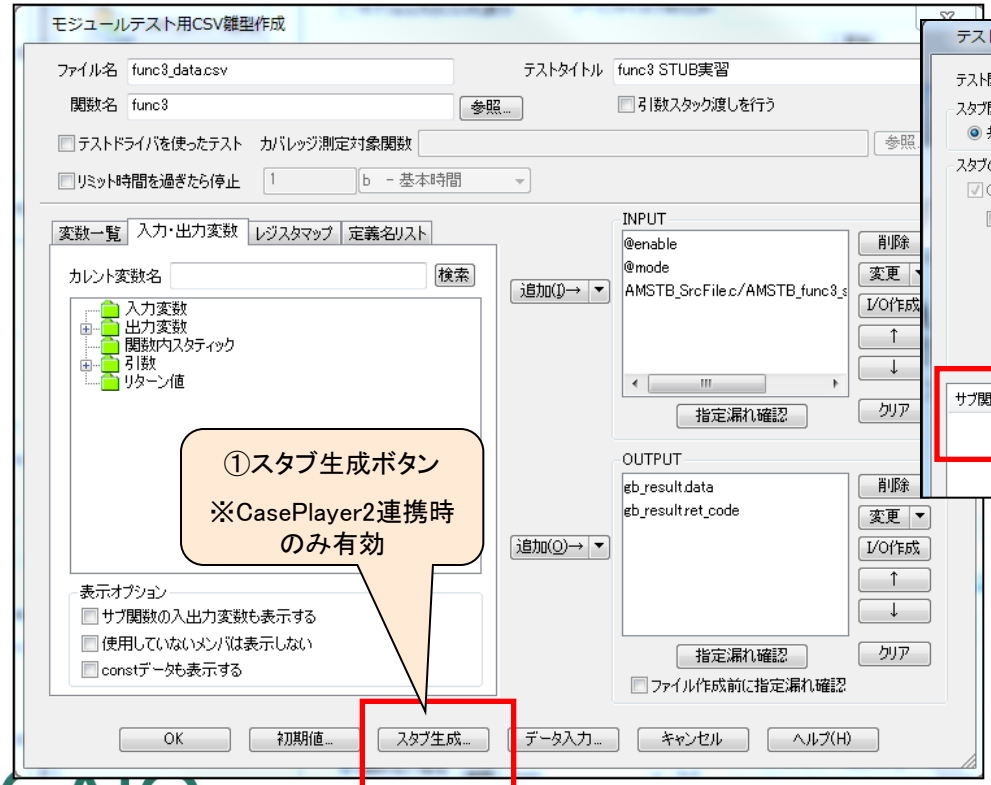
引数を結果CSVに出力

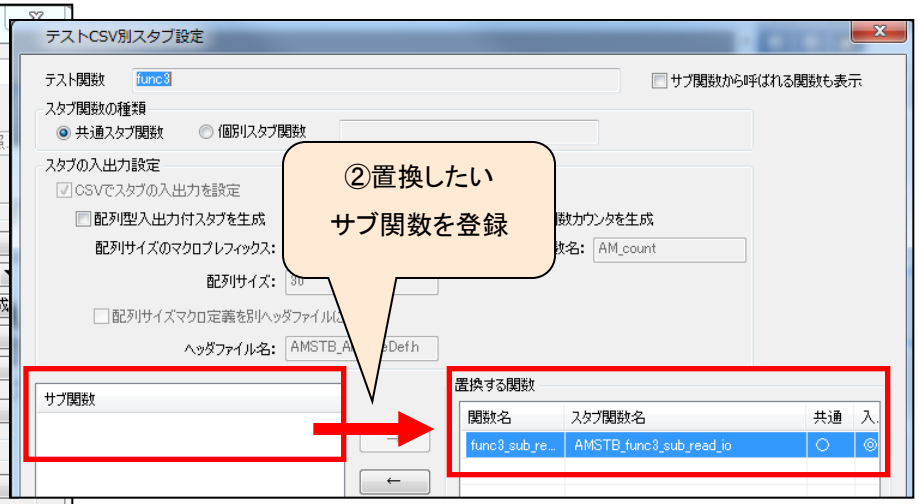
CSVの値を戻り値に返す

# （参考）CSV別スタブ 設定

- **スタブ設定ビューの「置換」スイッチを使用せず、CSV毎にスタブ関数の置換状態を設定する方法**
  - スタブ設定ビューの「置換」スイッチは、プロジェクト全体に有効なスイッチ




| | A | B | C |
|---|---|---|---|
| 1 | mod | func3 | func3 STUB実習 |
| 2 | % | AMSTB_func3_sub_read_io | func3_sub_read_io |
| 3 | #COMMENT | @enable | @mode |
| 4 | | 1 | 0 |
| 5 | | 1 | 0 |
| 6 | | 1 | 0 |

③CSVファイルに
登録される

# プログラム解析ツール CasePlayer2との連携機能 (実習4の前に)

# CasePlayer2との連携による拡張機能

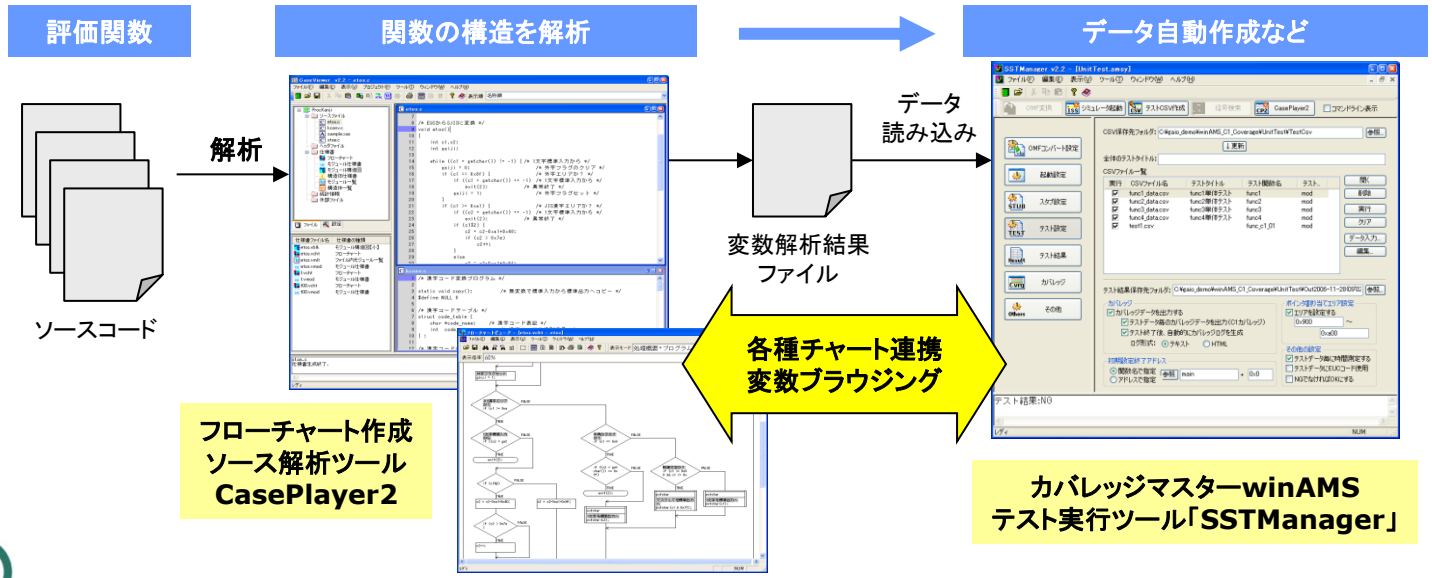
## ■ カバレッジ計測機能の拡張

- C1カバレッジ計測（実コード実行による計測）　※C1計測にはCasePlayer2が必要
- MC/DCカバレッジ計測（埋め込みコードによる計測）

## ■ テストデータ設計機能の効率化

- テスト対象関数の入出力変数を自動抽出
- C1、MC/DC構造カバレッジ計測用 テストケース自動生成

## ■ C++コード単体テストへの対応



# 対象関数の入出力変数の自動抽出機能（標準機能）

## ※ CasePlayer2との連携機能

# 対象関数の入出力変数の自動抽出機能

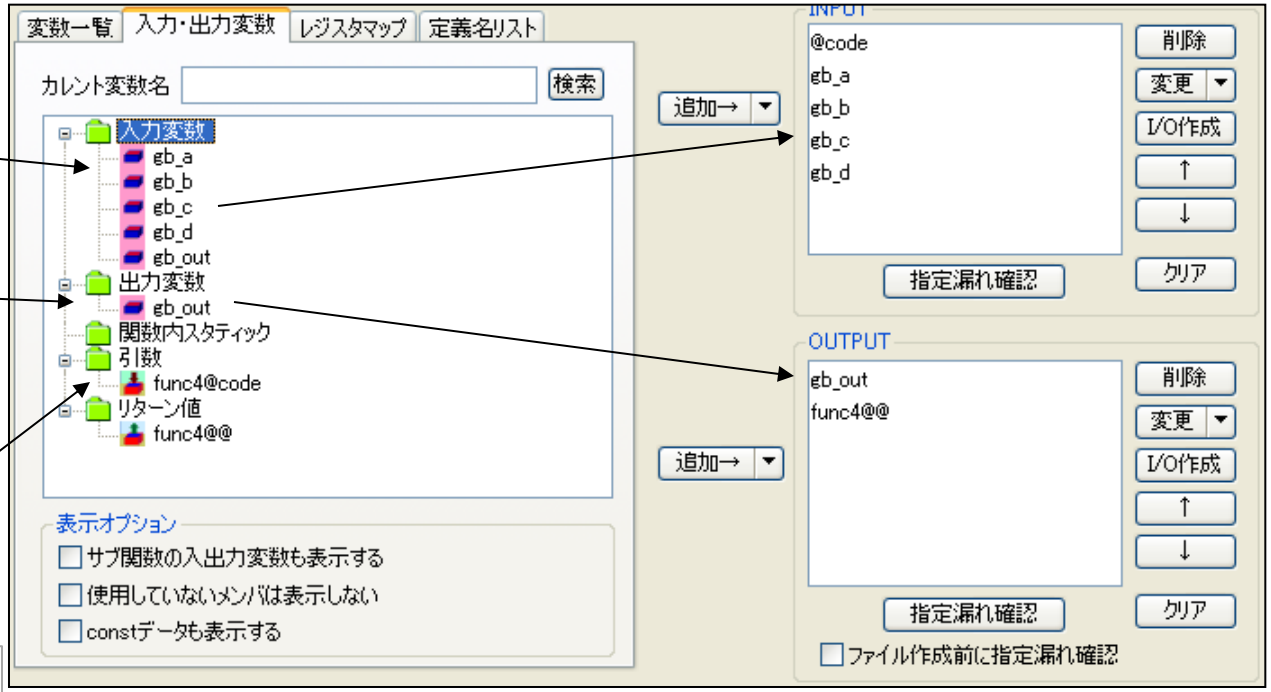
■ **評価対象の関数が使用(参照/代入)している外部変数を自動抽出**

– 入力条件となる変数(引数、外部変数)を自動検索し コード解析工数を省力化

**関数が参照している外部変数**
(単体テストの入力変数)

**関数が書き換えている外部変数**
(単体テストの評価変数)

**関数の引数リスト**
(単体テストの入力変数)




**※注意事項:**
**表示される変数はソースコードを解析した情報 → 仕様と一致しているかは確認の必要がある**

**入出力変数リスト**
評価対象の関数を指定すると自動表示される

リストアップされた変数から評価に必要な変数をユーザーが選択・指定

# C1/MCDC 計測用テストケース自動生成機能（標準機能）

## ※ CasePlayer2との連携機能

# C1カバレッジデータを自動生成

## ■ 静的解析により各条件文の論理を成立させるテストデータを自動生成

– 関数内の条件文を自動表示、 各条件文のTRUE/FALSEケースを自動生成

```
// C1説明用サンプル
int  gb_a, gb_b, gb_c, gb_d, gb_out;
char gb_unused1, gb_unused12;

int func_c1_01( int code )
{
    int return_value=FALSE;

    if( gb_a > 10 )
    {
        if( gb_b > 20 && gb_c > 30 )
        {
            gb_out = 0;
        }
        else
        {
            gb_out = -1;
        }
        return_value = FALSE;
    }
    else
    {
        switch( code )
        {
            case 1:
                gb_out = 1;
                break;
            case 2:
                gb_out = 2;
                break;
            case 3:
                gb_out = 3;
                break;
            default:
                gb_out = -1;
                break;
        }
        return_value = FALSE;
    }
    return return_value;
}
```

**C1カバレッジ モードでの 条件文解析結果**

条件文　if ( gb_a>10 )

| 条件文 | 行番号 | 編集 | テストデータ |
|---|---|---|---|
| if ( gb_a>10 ) | 48 | ◎ | |
| TRUE | | | gb_a=11 |
| FALSE | | | gb_a=10 |
| if ( gb_b>20&&gb_c>30 ) | 50 | ◎ | gb_c=30 |
| TRUE | | | gb_b=21 ; gb_c=31 |
| FALSE | | | gb_b=20 ; gb_c=31 |
| switch ( code ) | 63 | ◎ | |
| case 1: | | | @code=1 |
| case 2: | | | @code=2 |
| case 3: | | | @code=3 |
| default: | | | @code=4 |
| その他 | --- | | |

各々の条件文の TRUE/FALSE の各ケースを満たすために
必要な変数を自動設定する

# C1を満たすデータの組合せを自動生成

## ■ 条件文のネスト構造を解析して 全てのパスを通るテストケースを生成

- 各条件文（if,switchなど）に対して生成したデータの組み合わせを自動生成
- 同じ分岐経路を通るデータ、同じ分岐条件は、重複生成しない
- 必要最小限のデータ組み合わせを生成

**C1のカバレッジを満たす入力データを自動生成**

| | COMMENT | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| COMMENT | | | | | | | |
| NAME | コメント | @code | gb_a | gb_b | gb_c | gb_out | func_c1_01@@ |
| 1 | | 4 | 11 | 20 | 31 | | |
| 2 | | 4 | 11 | 21 | 31 | | |
| 3 | | 4 | 11 | 20 | 30 | | |
| 4 | | 4 | 10 | 20 | 31 | | |
| 5 | | 3 | 10 | 20 | 31 | | |
| 6 | | 2 | 10 | 20 | 31 | | |
| 7 | | 1 | 10 | 20 | 31 | | |

ここは期待値

実行結果の評価は、
**関数の詳細仕様を基に**
必ず行わなければ
ならない
↓
必ず発生する工数

# 生成したデータが評価する条件式をコメント表示

- **CSVファイルに 生成したデータに対応する条件式と論理を表示可能**
  - 生成されたテストベクターが 何を評価しているのかを明示

CSVデータ編集

タイトル @code

| | COMMENT | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| COMMENT | | | | | | | | |
| NAME | コメント | @code | gb_a | gb_b | gb_c | gb_d | gb_out | func4@@ |
| 1 | | ;if ( gb_a>10 ) | | | | | | |
| 2 | | ;TRUE | | | | | | |
| 3 | | 1 | 11 | 21 | 31 | 1 | | |
| 4 | | ;FALSE | | | | | | |
| 5 | | 1 | 10 | 21 | 31 | 1 | | |
| 6 | | ;if ( gb_b>20&&gb_c>30 ) | | | | | | |
| 7 | | ;TRUE | | | | | | |
| 8 | | 1 | 11 | 21 | 31 | 1 | | |
| 9 | | ;FALSE | | | | | | |
| 10 | | 1 | 11 | 20 | 31 | 1 | | |
| 11 | | ;その他の値 | | | | | | |
| 12 | | 1 | 11 | 21 | 30 | 1 | | |

# MC/DCカバレッジデータを自動生成

## ■ MC/DCデータ作成のために 複合条件文に含まれる条件式を解析

- 条件文をリストアップして、各条件文のTRUE/FALSEケースを自動生成

```
// MCDC説明用サンプル
int func_mcdc_01( int x, int y, int z )
{

    //
    if( ( x>10 && y>20 ) || z>30 )
    {       (1)      (2)        (3)
            return TRUE;
    }
    else
    {
            return FALSE;
    }
    return FALSE;
}
```

**MCDCカバレッジ モードでの 複合条件文解析結果**

| 条件文 | 行番号 | 編集 | テストデータ |
|---|---|---|---|
| if ( (x>10&&y>20)\|\|z>30 ) | 27 | ◎ | |
| x>10 | | | |
| TRUE | | | @x=11 |
| FALSE | | | @x=10 |
| y>20 | | | |
| TRUE | | | @y=21 |
| FALSE | | | @y=20 |
| z>30 | | | |
| TRUE | | | @z=31 |
| FALSE | | | @z=30 |
| その他 | --- | | |

各々の条件文の TRUE/FALSE の各ケースを満たすために
必要な変数を自動設定する

# MC/DC指針のデータを自動抽出

■ **複合条件を解析して MC/DCカバレッジを満たす入力データを自動生成**

– MC/DCの指針に従い、全条件組み合わせ（All-Pair）からMC/DCデータを自動抽出

**複合条件例**

```
if( ( x>10 && y>20 ) || z>30 )
      (1)     (2)       (3)
```

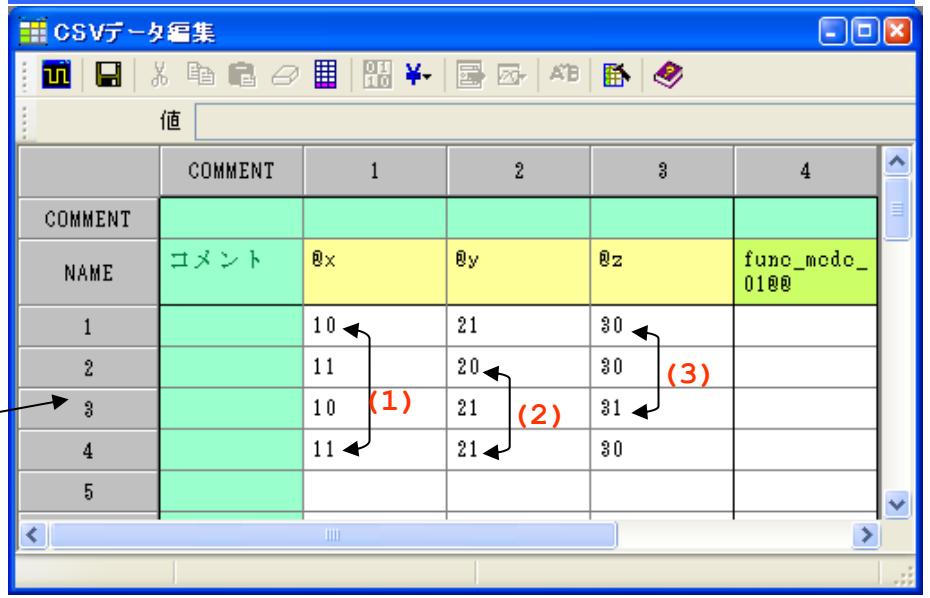
＜MCDCのデータ抽出指針＞
他の条件式の真偽を固定して、1つの条件式の真偽を変化させたとき、全体の条件に変化があるものだけを実行する

**条件式のTRUE/FALSEの全組み合わせから MCDCデータを抽出**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 条件式 (1) | F | F | F | F | T | T | T | T |
| 条件式 (2) | F | F | T | T | F | F | T | T |
| 条件式 (3) | F | T | F | T | F | T | F | T |
| If 全体論理 | F | T | F | T | F | T | T | T |

(3)　(2)　(1)

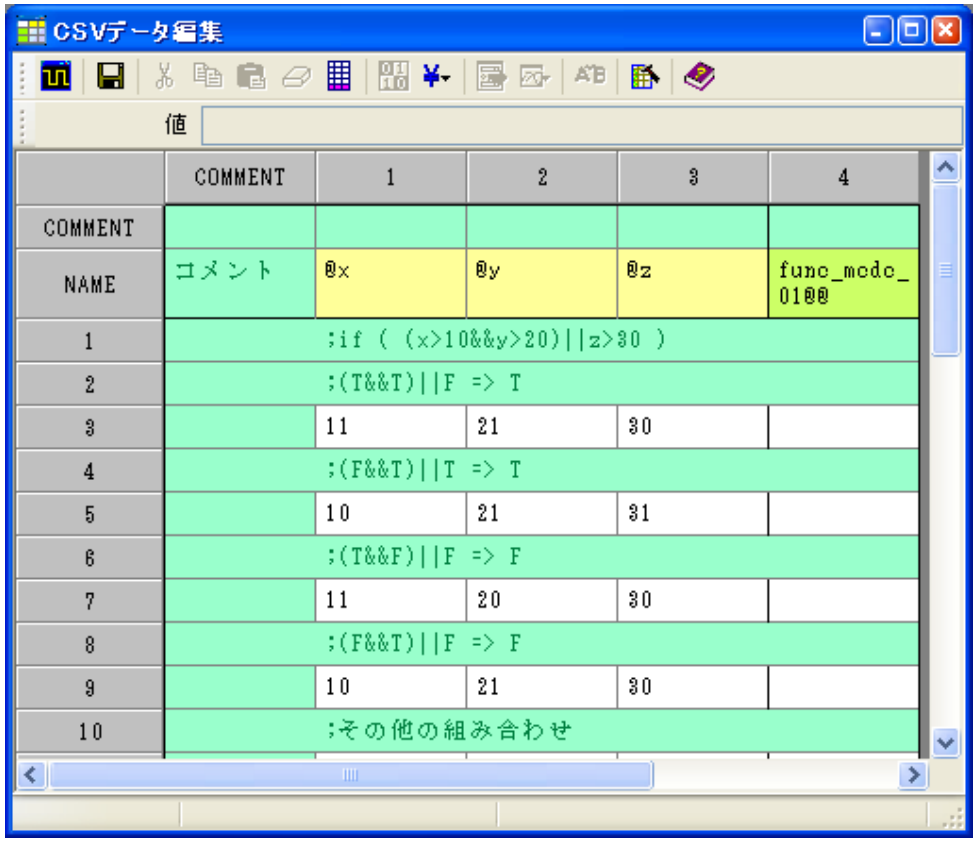
**MCDCのカバレッジを満たす入力データを自動生成**



# MC/DC生成データが評価する論理条件を表示

## ■ CSVファイルに 生成したデータと 対応する論理条件を表示

- 生成されたテストベクターが 何を評価しているのかを明示



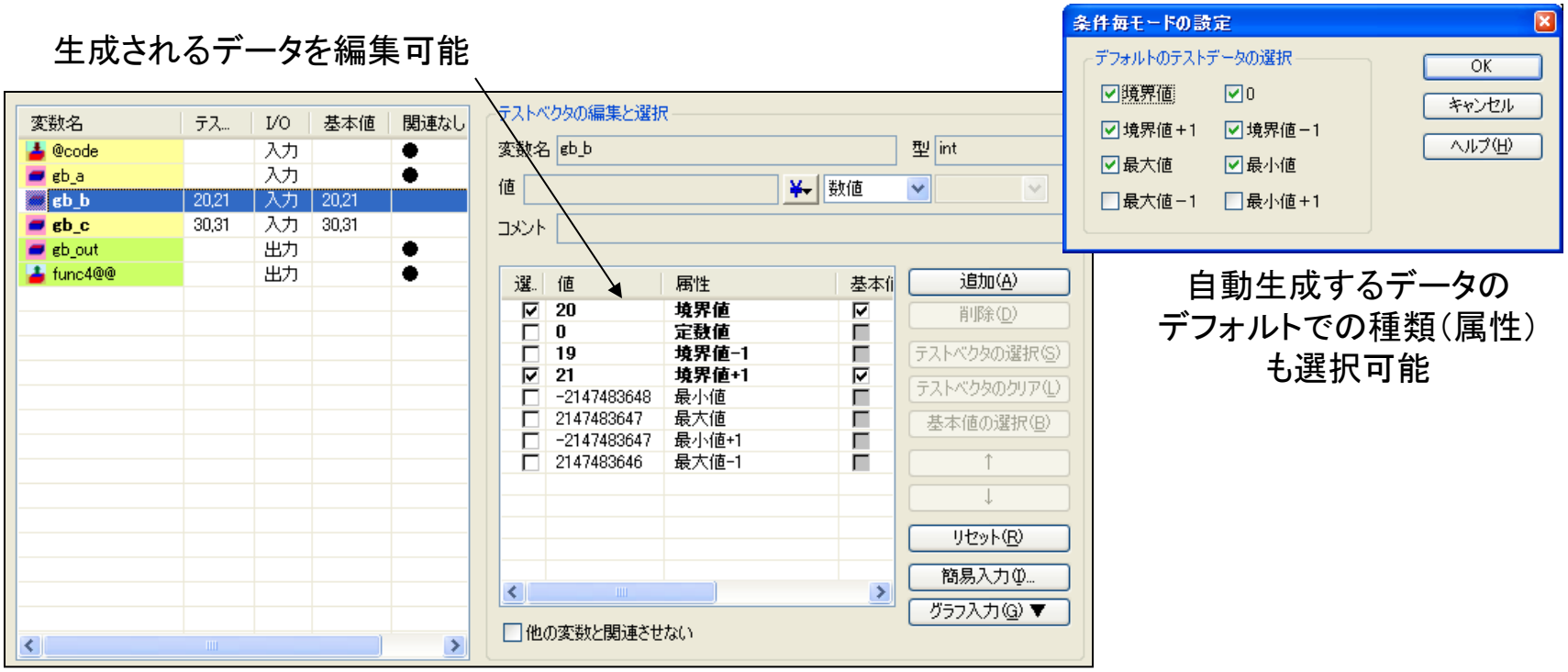

| | COMMENT | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| COMMENT | | | | | |
| NAME | コメント | @x | @y | @z | func_mcdc_01@@ |
| 1 | ;if ( (x>10&&y>20)\|\|z>30 ) | | | | |
| 2 | ;(T&&T)\|\|F => T | | | | |
| 3 | | 11 | 21 | 30 | |
| 4 | ;(F&&T)\|\|T => T | | | | |
| 5 | | 10 | 21 | 31 | |
| 6 | ;(T&&F)\|\|F => F | | | | |
| 7 | | 11 | 20 | 30 | |
| 8 | ;(F&&T)\|\|F => F | | | | |
| 9 | | 10 | 21 | 30 | |
| 10 | ;その他の組み合わせ | | | | |

# 生成データの編集も自由に可能

## ■ 解析情報より自動生成されるデータの追加削除も自由に可能

- テスト指針に従って、境界値近傍のデータ、特異点データなどを追加
- 自動生成するデータの属性（種類）も設定可能

生成されるデータを編集可能

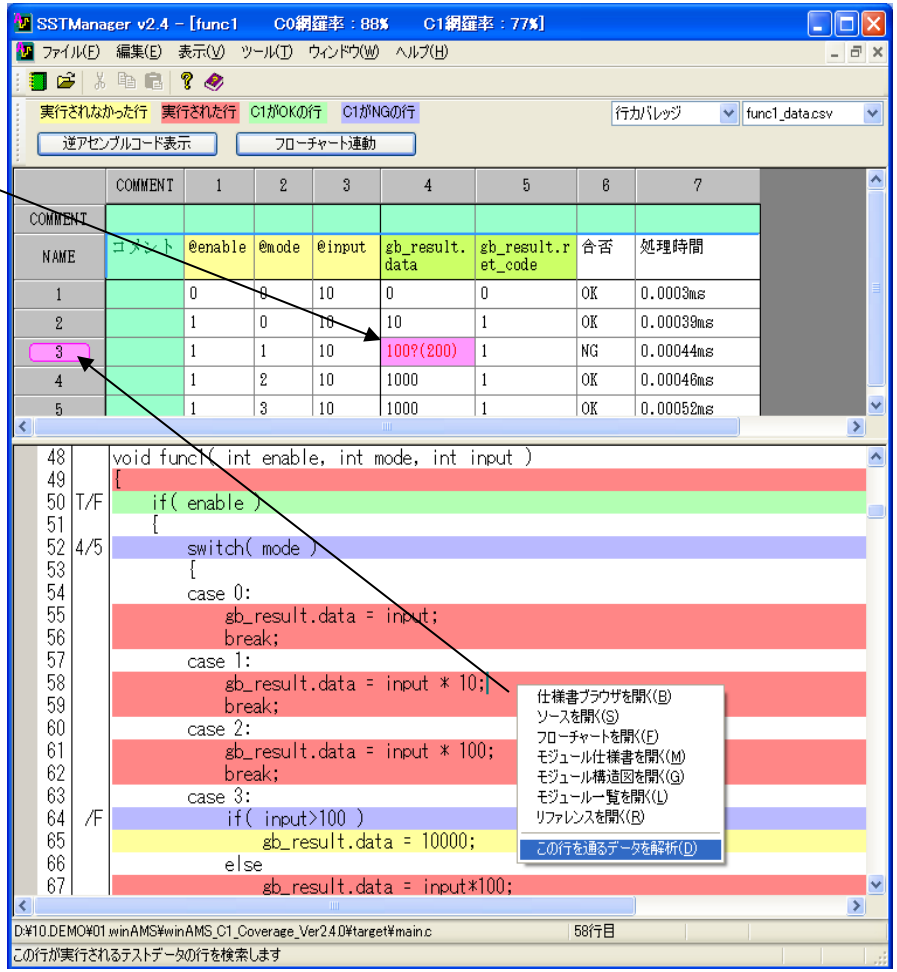



自動生成するデータの
デフォルトでの種類（属性）
も選択可能

# C0/C1カバレッジ、入出力結果表示

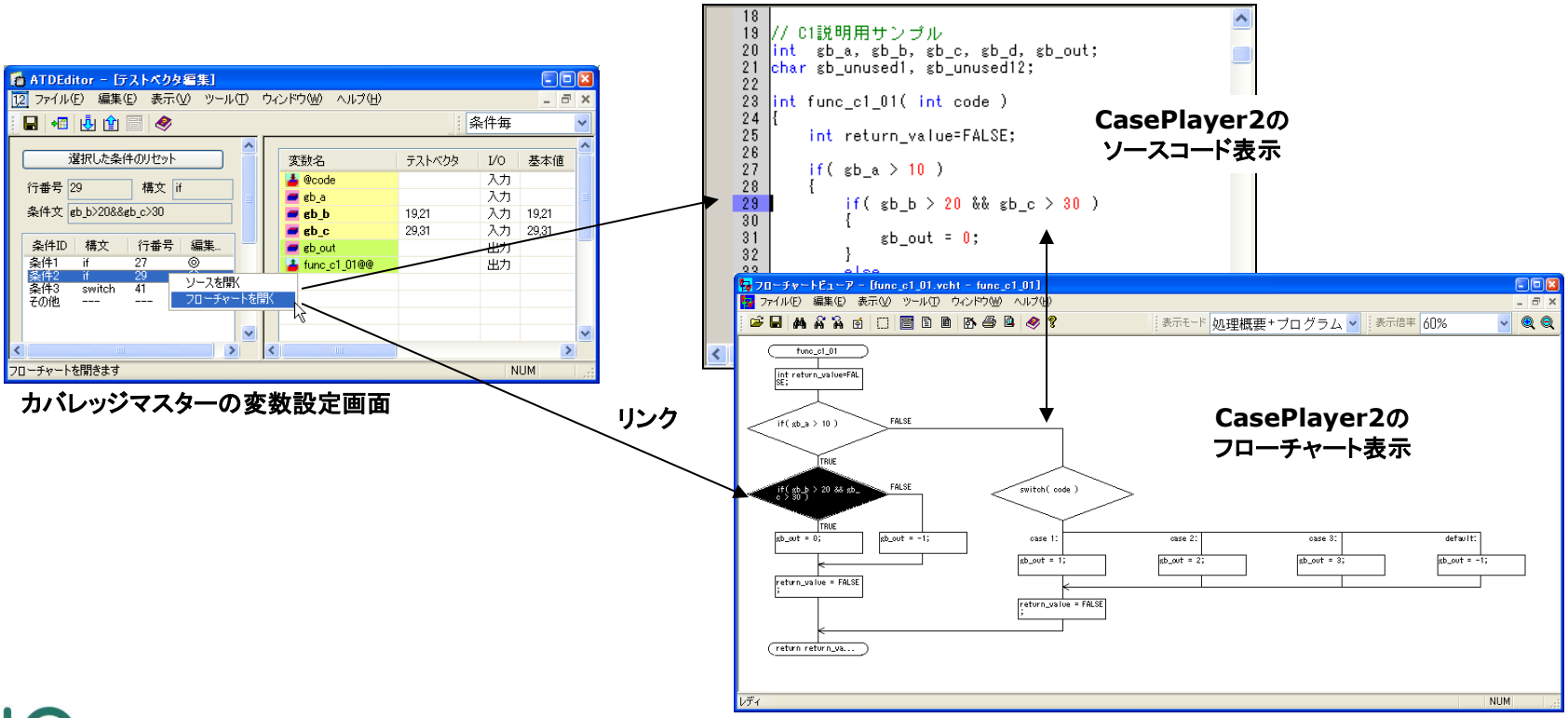
## C0/C1カバレッジビュー

- カバレッジ結果とテストデータを表示
- 期待値と異なるデータセルを表示
  - 100?(200)
    →期待値が200だが結果は100
- C0を満たさない未実行行を表示
- C1を満たさない条件を明示
- ★各行を通過するデータを解析し
  
  で表示
  - 未実行行のデータを追加する際の解析に非常に役立つ機能



# ソース解析に役立つチャートリンク機能

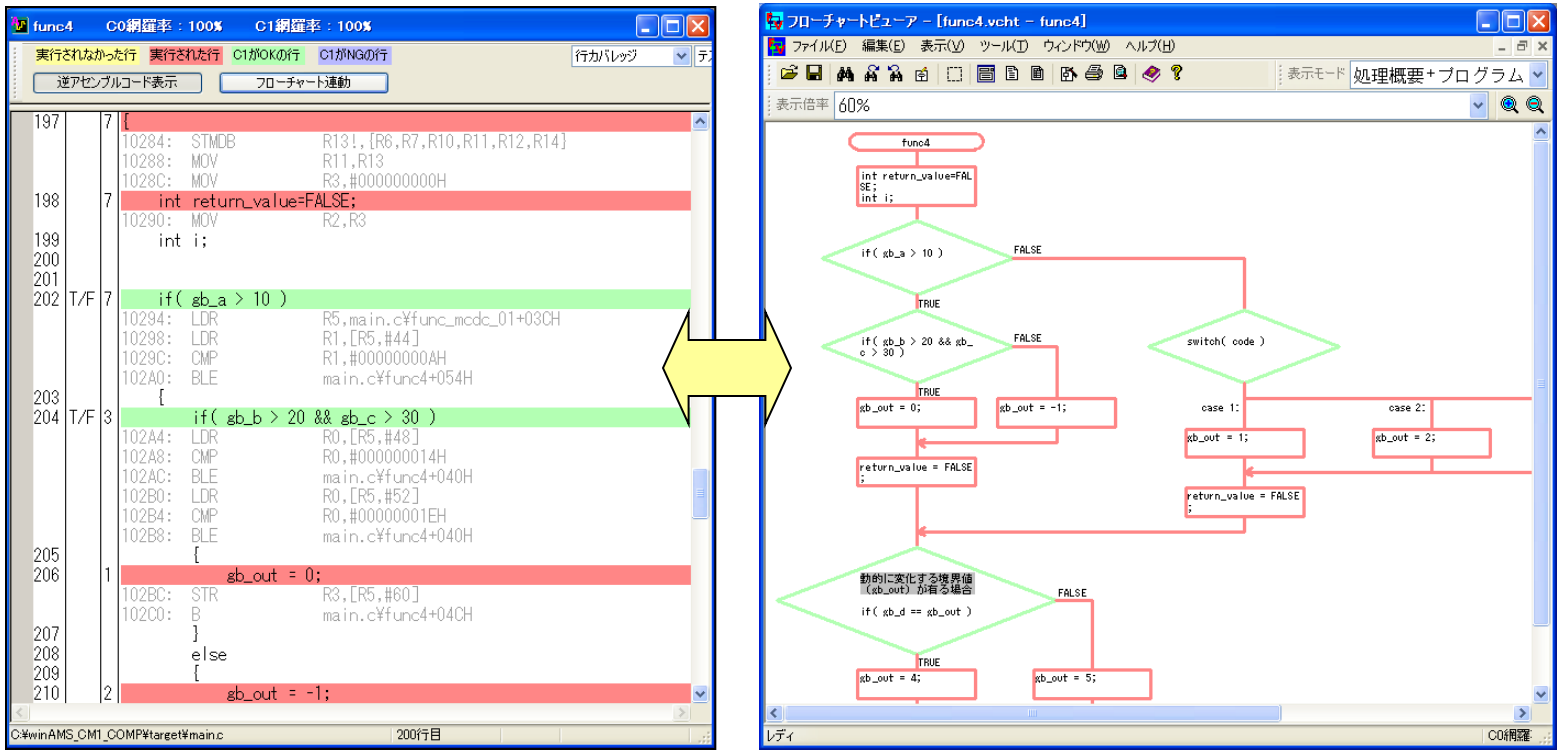
- **CasePlayer2が生成した各種チャート、変数参照リストへリンク**
  - 面倒な設定やソースコードの修正追加なしで 簡単にチャートを生成
  - 入出力データ作成時のロジック解析、パス解析を効率化



# フローチャート連動

## ■ フローチャートC0・C1カバレッジ表示

– テスト実行パスをフローチャート上に表示して、ビジュアルに判断可能



# 実習4
# CasePlayer2と連携した
# C0/C1カバレッジ
# テストデータ作成機能を使う

# 実習4-1：評価対象ソースを確認

- 実習4は CasePlayer2との連携で C0/C1カバレッジを満たすテストデータを自動作成します
  - グローバル変数には未使用の物も含まれる（自動検索機能で使用中の変数を見つけ出す）

```
int gb_input;
int gb_output;
int gb_valA;
int gb_valB;
int gb_valC;
int gb_valD;
int  gb_a, gb_b, gb_c, gb_d, gb_out;
char gb_unused1, gb_unused2;

int func4( int code )
{
      int return_value=FALSE;
      int i;
      if( gb_a > 10 )
      {
            if( gb_b > 20 && gb_c > 30 )
            {
                  gb_out = 0;
            }
            else
            {
                  gb_out = -1;
            }
            return_value = FALSE;
      }
      else
      {
            switch( code )
            {
                  case 1:
                        gb_out = 1;
                        break;
                  case 2:
                        gb_out = 2;
                        break;
                  case 3:
                        gb_out = 3;
                        break;
                  default:
                        gb_out = -1;
                        break;
            }
            return_value = FALSE;
      }

      // 下の条件式は、上の演算結果(gb_out)を使って比較
      // しているので、動的に決まる境界値は静的解析できない

      if( gb_d == gb_out )
      {
          gb_out = 4;
      }
      else
      {
          gb_out = 5;
      }

       return return_value;
}
```

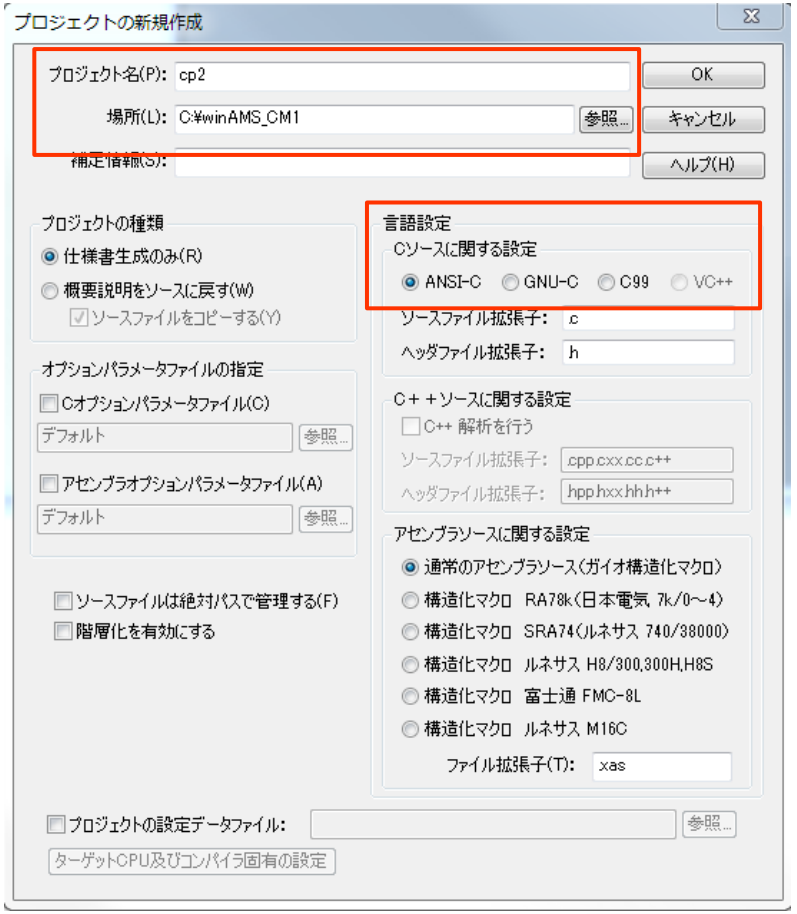
# 実習4-2:プロジェクト作成/言語選択

■ **CasePlayer2を起動し、プロジェクトを新規作成します**
**同時にC言語の種類を選択します**

1. 「スタート」メニュー →「CasePlayer2」
   →「caseviwer.exe」を起動する


2. 「ファイル」メニュー →「新規作成」
   →「プロジェクト」 を選択する
3. 「プロジェクト名」、「場所」を設定します
   1. プロジェクト名:CP2
   2. 場所:c:¥winAMS_CM1
4. 言語設定を選択します
   1. ANSI-C

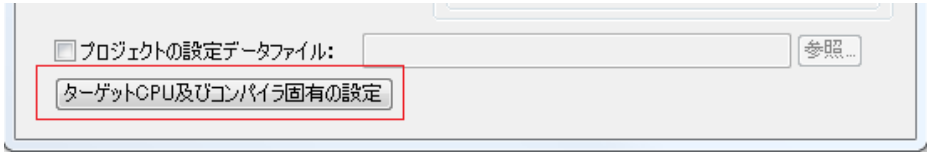
**他のオプションはそのままで構いません**
**(仕様書作成機能のオプションのため、今回は使用しません)**

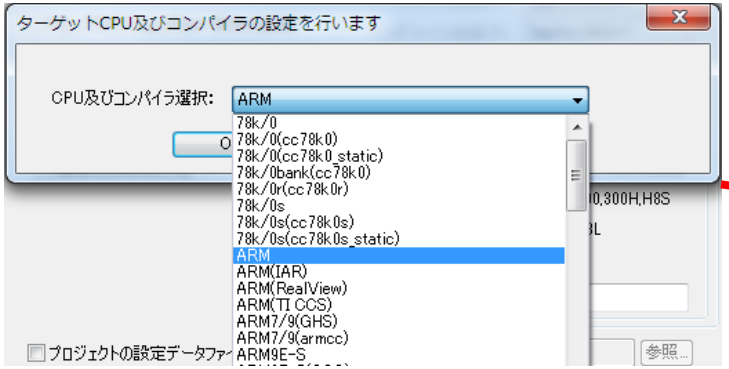


# 実習4-3：ターゲットCPU及びコンパイラ固有の設定

## ■ 解析するソースに含まれる方言に対応するための設定を行う

1. 「ターゲットCPU及びコンパイラ固有の設定」のボタンを押す
2. 使用しているマイコンの種別と、コンパイラを選択する
   - クロスコンパイラ特有の記述（方言）に対応するための機能「Cオプションパラメータ」が自動設定される
   - 「Cオプションパラメータ」は、プロジェクト作成後、[設定]→[Cオプションパラメータ]で確認/変更/追加 可能



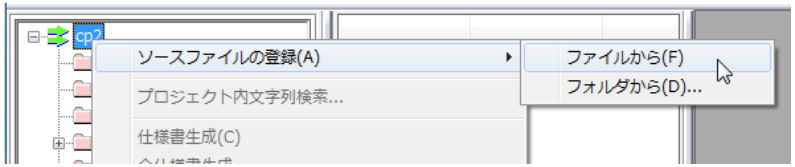

コンパイラの方言に対応するための、後述のCオプションパラメータが自動設定される

# 実習4-4：プロジェクトに解析対象ソースを登録

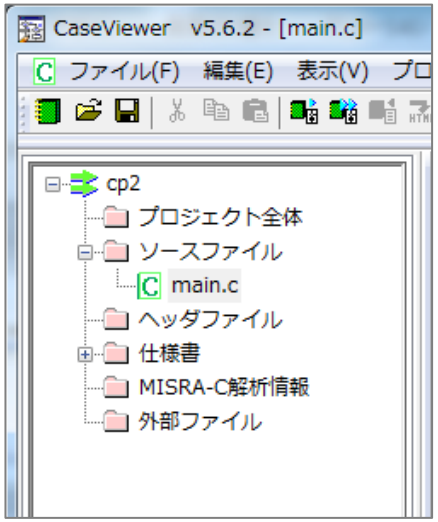
## ■ 解析対象のソースコード（main.c）をプロジェクトに登録します

1. 「OK」を押して、プロジェクトの新規作成ダイアログを閉じます。
   CaseViewerが起動します。

2. CaseViewer のツリーのプロジェクト名「cp2」を右クリックします
3. 「ソースファイルの登録」→「ファイルから」を選択します




4. 「main.c」を選択します
   - C:¥winAMS_CM1¥target¥main.c
5. 「開く」を押して、main.cを登録します
   - CasePlayer2プロジェクトへソースが登録されます

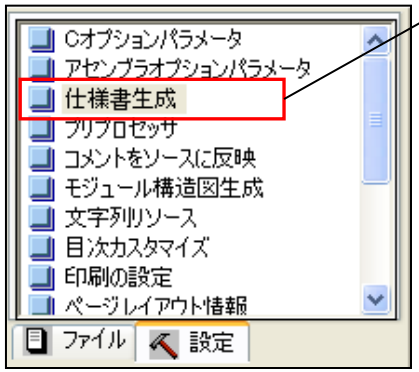


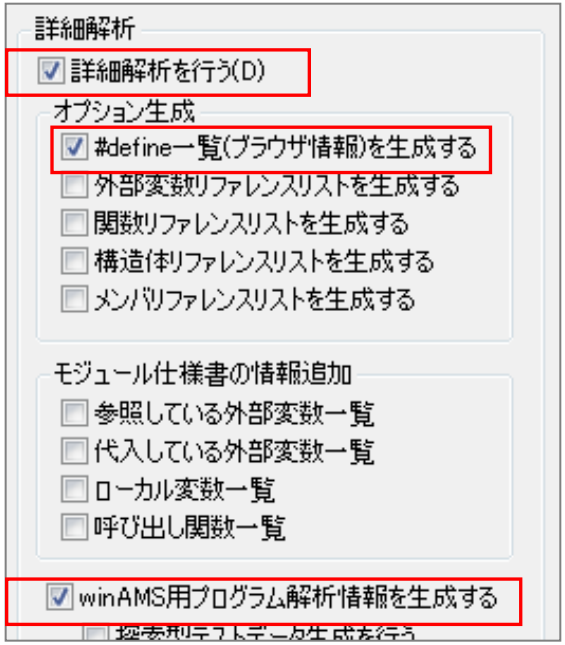
# 実習4-5：カバレッジマスター連携の設定

## ■ カバレッジマスターと連携するための必須オプションをONにします

1. 「設定」タブの「仕様書生成」をダブルクリックします。
2. 以下のオプションをONにします。
   - 詳細解析を行う
     - #define一覧（ブラウザ情報）を生成する
     - winAMS用プログラム解析情報を生成する

   ※他のオプションは仕様書生成のオプションです
3. 「OK」で閉じます。
   - これで、CasePlayer2からカバレッジマスター向けの解析データが出力されるように設定されます



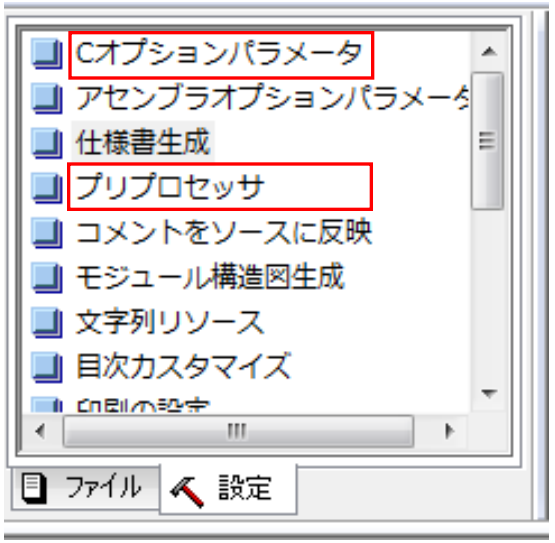
# 実習4-6：プリプロセッサ/Cオプションパラメータ設定

## ■ CasePlayer2でソースを正しく解析するための設定

- **プリプロセッサ**
  - 開発環境、コンパイラに設定しているものと同じにする

  [ヘッダファイルの検索パスの設定]
    - システムインクルードパス
    - ユーザーインクルードパス

  [#ifdefなどの切換指定をコンパイラに与えている場合]
    - #define値
- **Cオプションパラメータ**
  - クロスコンパイラ特有の記述（方言）を正しく解析するための設定
  - プロジェクト作成時に「ターゲットCPU及びコンパイラ固有の設定」で自動設定可能
  - 新規の解析設定（新規方言）を追加可能




（参考）FAQに掲載：

http://www.gaio.co.jp/support/user/faq/winams/faq_D01_01.html

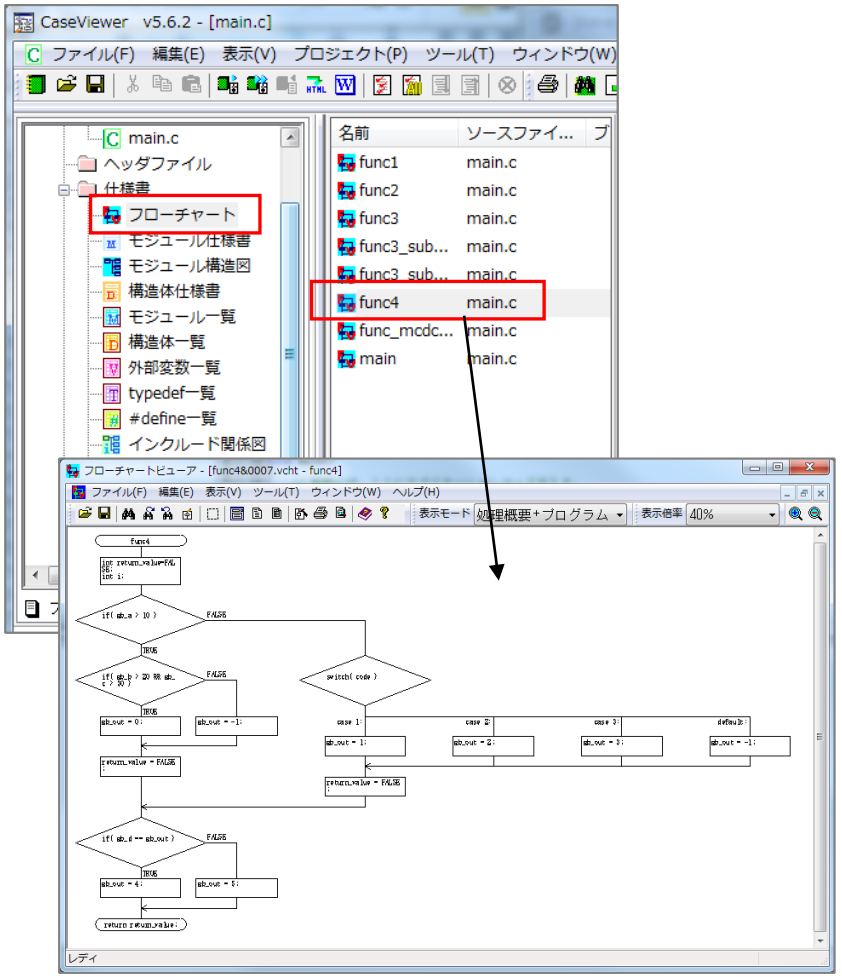
# 実習4-7：解析を実行

■ **仕様書作成＆静的解析を実行します。**

1. 「プロジェクト」メニューの「仕様書を生成」を選択します。
2. 例えば、評価対象の関数func4()のフローチャートを表示してみます。
   - 「仕様書」フォルダの「フローチャート」を選択します。
   - 「仕様書ファイル名」のリストからfunc4.vchtをダブルクリックします。

これにより、静的解析結果が
CP2.vproj ファイルに
生成されます

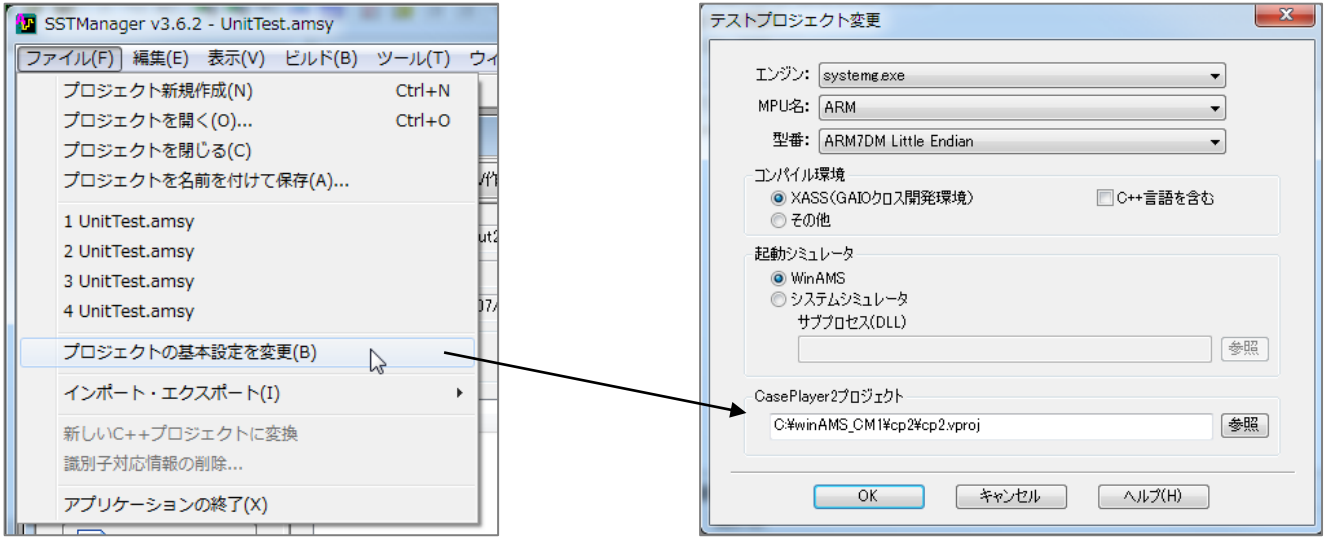
ファイルパス：　C:¥winAMS_CM1¥CP2Project



# 実習4-8：CasePlayer2との連携を設定

## ■ カバレッジマスターにCasePlayer2を連携するための設定をします

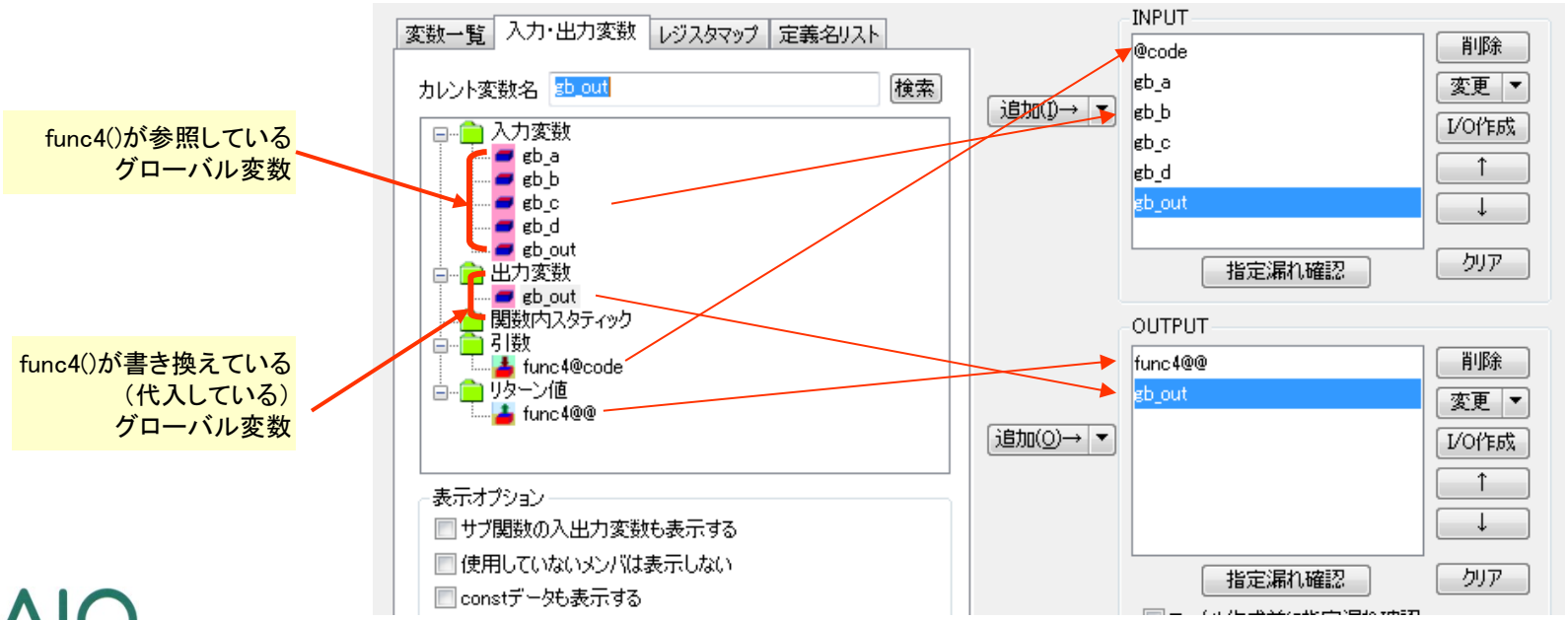
1. SSTManagerの「ファイル」メニューの、「プロジェクトの基本設定を変更」を選択します。
2. 表示されるダイアログで、CasePlayer2のプロジェクトファイルを設定します
   C:¥winAMS_CM1¥CP2¥CP2.vproj
3. 「OK」を押して閉じます。
   - これで、CasePlayer2での静的解析の結果が、カバレッジマスターで利用できる状態になります。



# 実習4-9：テストデータ作成

## ■ 入出力変数自動検索機能を使って 変数を選択する

1. SSTManagerの「テストCSV作成」ボタン押して、「モジュールテスト用CSV」を選択する。
2. テストタイトル、ファイル名（func4_data）、関数名（func4）を設定する。
   - 「入力変数」、「出力変数」のフォルダに、func4()が参照/代入するグローバル変数が自動的にリストアップされる。（←静的解析結果が参照されている）
3. 下図のように、INPUT/OUTPUTの変数を選択する



# 実習4-10:テストデータ作成(続)

## ■ 入力変数データ作成のモード

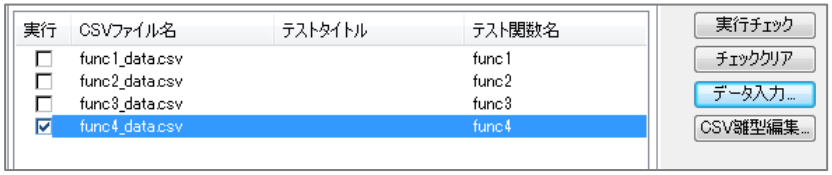
- **「C1データ作成」モード:　C1カバレッジ入力データ作成のためのモード**
  - CasePlayer2解析情報により、C1を満たす最小限のデータを自動生成する
  - 各条件式ごとに、関連する変数の真偽を満たすデータを自動生成
  - 最小限のデータで、C1カバレッジを満たすことができる
- **「MCDCデータ作成」モード:　MCDCカバレッジ入力データ作成のためのモード**
  - **上と同様にして、MCDCに準拠したデータを生成**
- **「マニュアル作成」モード:　データカバレッジのためのデータを生成するモード**
  - CasePlayer2解析情報を使用しない
  - 各変数に与えるデータは手動設定
  - データの組み合わせは「デシジョンテーブル」の手法により
    設定した「基本値」により、組み合わせが作成される
- **「1行データ作成」モード:　CSVの1行文のデータを作成するモード**
  - 各変数には1つの値を設定可能
  - 組み合わせを考えず、単に1つのテストデータを作る機能

※各モードは併用可能　←各モードで作成したデータを1つのCSVファイルに追記可能

# 実習4-11：テストデータ作成エディタを起動

## ■ ATDEditorを起動します

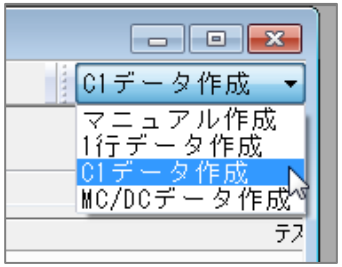
1. テスト設定ビューで「func4_data.csv」を選択します
2. 右にある「データ入力」ボタンを押します
3. ATDEditorが起動します




## ■ 「テストベクタ編集」のウインドウを表示します

※表示されない場合は、「表示」メニューで選択

1. 右上のプルダウンから「C1データ作成」を選択します

# 実習4-12：生成されたテストベクタを確認

■ 「条件毎」モード：　**func4()に含まれる条件文を確認する**

- 「編集」欄に、◎が付いている条件式は、解析によりデータを自動生成できたことを示す
- 各条件式毎に、TRUE/FALSE になるデータを設定する
- 最後の条件式「if(gb_d == gb_out)」は、自動生成ができなかったことを示している

| 条件文 | 行番号 | 編集 | 式種別 | テストデータ |
|---|---|---|---|---|
| if ( gb_a>10 ) | 202 | ◎ | | |
| TRUE | | | | gb_a=11 |
| FALSE | | | | gb_a=10 |
| if ( gb_b>20&&gb_c>30 ) | 204 | ◎ | | gb_c=30 |
| TRUE | | | | gb_b=21 ; gb_c=31 |
| FALSE | | | | gb_b=20 ; gb_c=31 |
| switch ( code ) | 216 | ◎ | -- | |
| case 1: | | | | @code=1 |
| case 2: | | | | @code=2 |
| case 3: | | | | @code=3 |
| default: | | | | @code=4 |
| if ( gb_d==gb_out ) | 236 | | | |
| TRUE | | | | |
| FALSE | | | | |
| その他 | --- | | | |

自動生成ができなかったことを示している

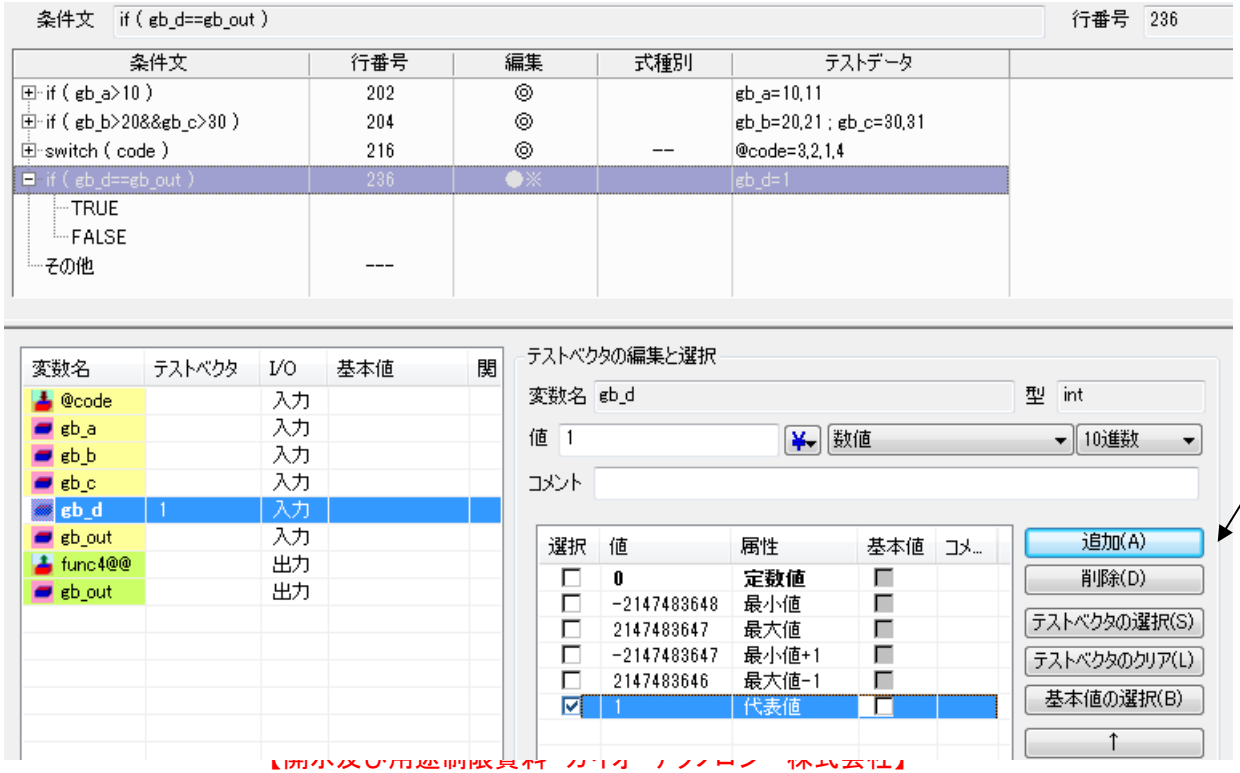
# 実習4-13：解析範囲外の条件を手動設定

## ■ CasePlayer2で解析できない条件文のデータを自分で設定する

1. 条件文（gb_d==gb_out）の部分は「◎」が付かず、解析できないことを示している
2. 関連する変数（gb_d）にテストベクタを設定する
   - gb_dに「1」を設定しておけば、TRUE/FALSEの両方（C1カバレッジ）が満たせる

①条件式を選択

②gb_dを選択




③追加を押す

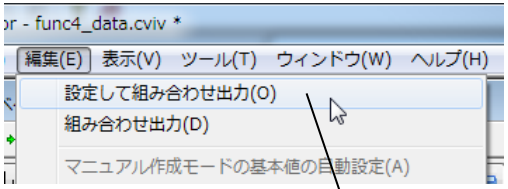
④データに「1」を追加

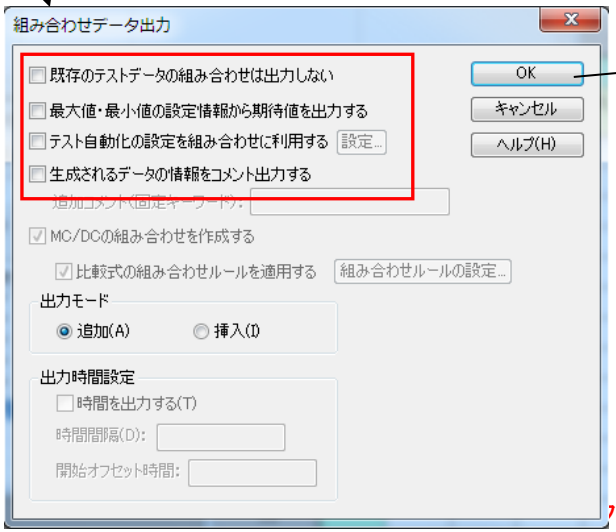
⑤「1」をチェック

# 実習4-14：CSVデータファイルを自動生成

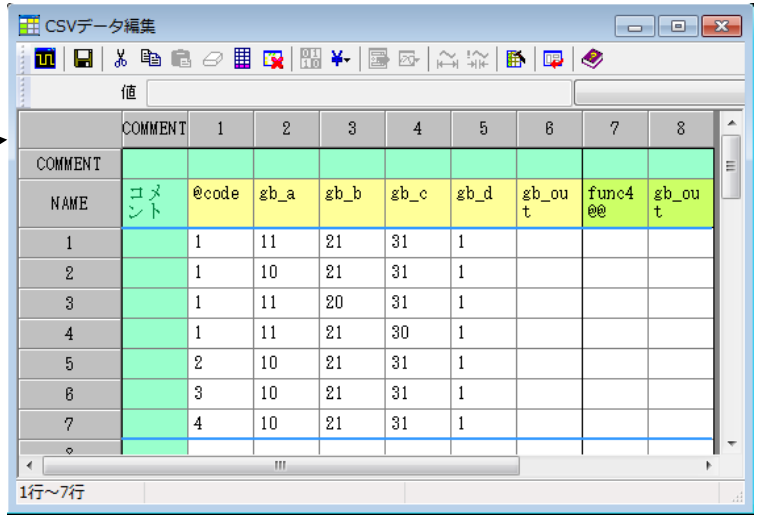
## ■ C0/C1カバレッジを満たす最小限のテストベクタを自動生成

1. 「編集」メニュー　→「設定して組み合わせ出力」を実行
2. 「組み合わせデータ出力」のダイアログで「OK」を押す。
   - 全てのチェックボックスをOFFにする




全てのチェックボックスを**OFF**




| | COMMENT | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| COMMENT | | | | | | | | | |
| NAME | コメント | @code | gb_a | gb_b | gb_c | gb_d | gb_out | func4@@ | gb_out |
| 1 | | 1 | 11 | 21 | 31 | 1 | | | |
| 2 | | 1 | 10 | 21 | 31 | 1 | | | |
| 3 | | 1 | 11 | 20 | 31 | 1 | | | |
| 4 | | 1 | 11 | 21 | 30 | 1 | | | |
| 5 | | 2 | 10 | 21 | 31 | 1 | | | |
| 6 | | 3 | 10 | 21 | 31 | 1 | | | |
| 7 | | 4 | 10 | 21 | 31 | 1 | | | |

# 実習4-15：生成データでテスト実行

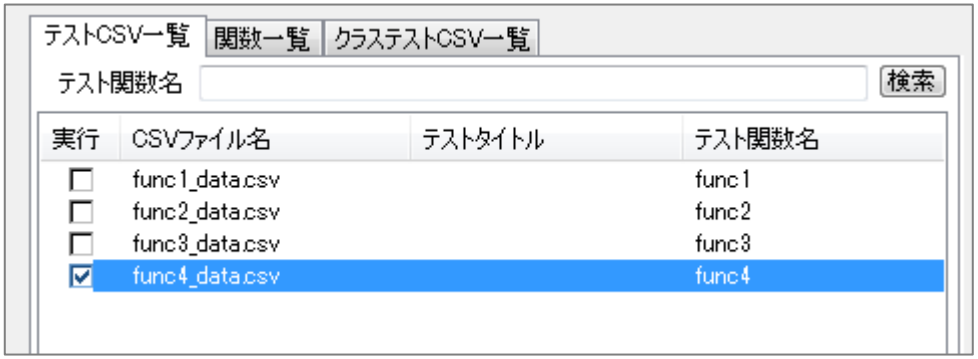
## ■ CSVデータをファイルに保存してテスト実行する

1. 「CSVデータ編集」ウインドウを選択して、「ファイル」メニュー　→「上書き保存」を選択
2. 「ファイル」メニュー　→「アプリケーション終了」を選択
    - 「*.cviv」ファイル（作業状態データ）の保存を求められたら、保存しておく
    - Func4_data.csvが保存される

4.SSTManagerの「テスト設定」で、「func4_data.csv」のチェックボックスをONにする

5.「C1カバレッジ出力」をONに」する

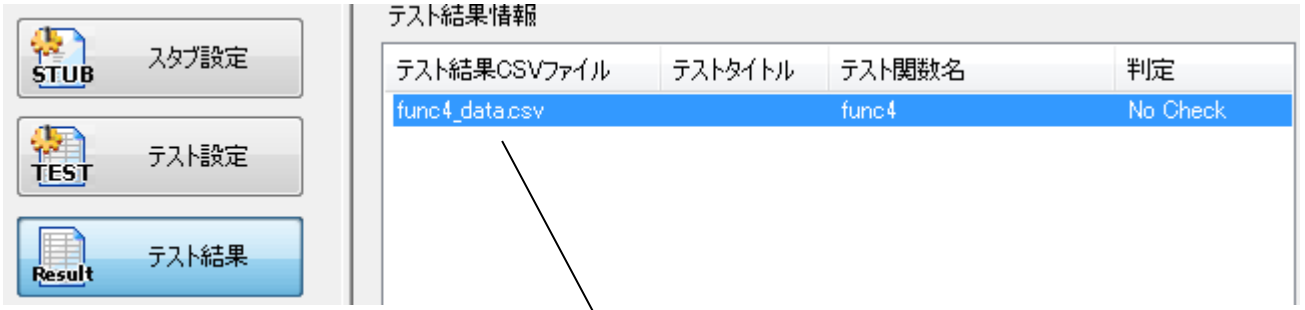
6.「シミュレータ起動」で、テスト実行する。




# 実習4-16：入出力テスト結果を確認する

## ■ 入出力テスト結果を確認

1.「テスト結果」で、「func4_data.csv」をダブルクリックする
　　ー出力結果が確認できる

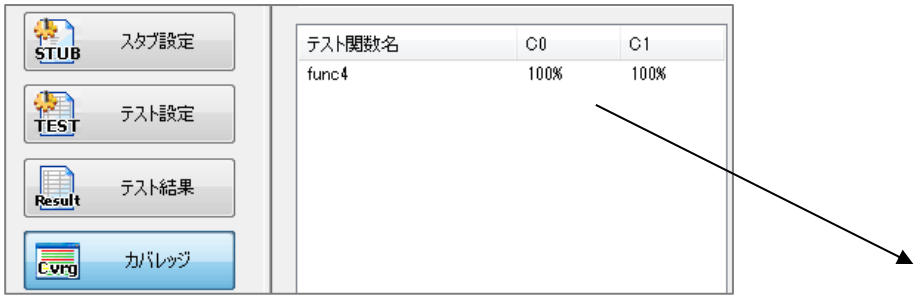


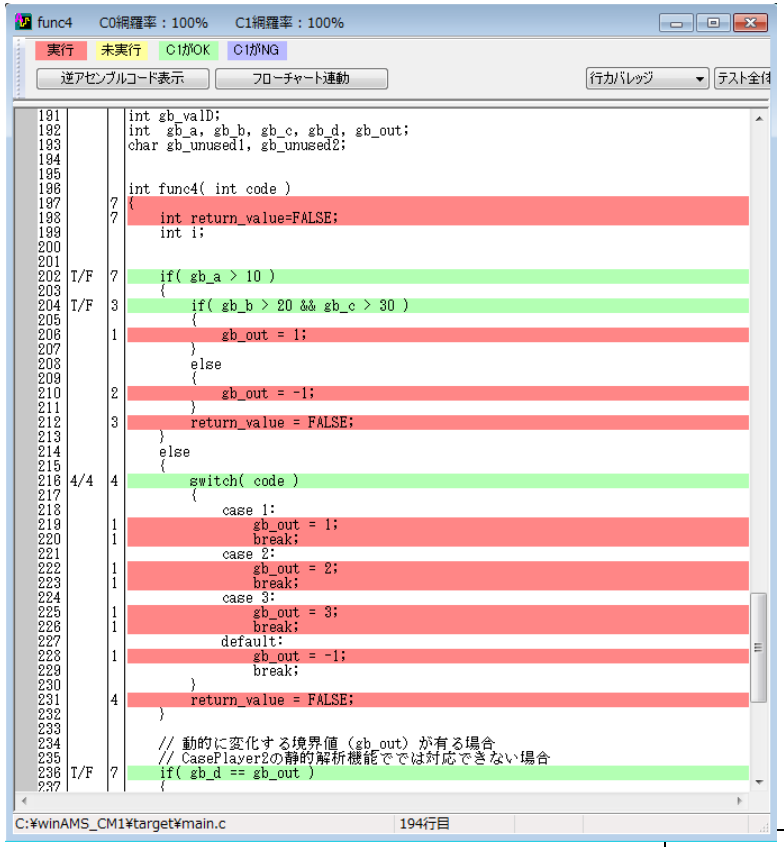

| | A | B | C | D | E | F | G | H | I | J |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | mod | func4 | | 6 | 2 | | | | CPP | |
| 2 | #COMMENT | @code | gb_a | gb_b | gb_c | gb_d | gb_out | func4@@ | gb_out | |
| 3 | | 1 | 11 | 21 | 31 | 1 | | 0 | 5 | NO Check |
| 4 | | 1 | 10 | 21 | 31 | 1 | | 0 | 4 | NO Check |
| 5 | | 1 | 11 | 20 | 31 | 1 | | 0 | 5 | NO Check |
| 6 | | 1 | 11 | 21 | 30 | 1 | | 0 | 5 | NO Check |
| 7 | | 2 | 10 | 21 | 31 | 1 | | 0 | 5 | NO Check |
| 8 | | 3 | 10 | 21 | 31 | 1 | | 0 | 5 | NO Check |
| 9 | | 4 | 10 | 21 | 31 | 1 | | 0 | 5 | NO Check |

# 実習4-17：カバレッジ結果を確認する

## ■ C0/C1カバレッジを確認する

1.「カバレッジ」を押して、C0/C1が100%であることを確認する

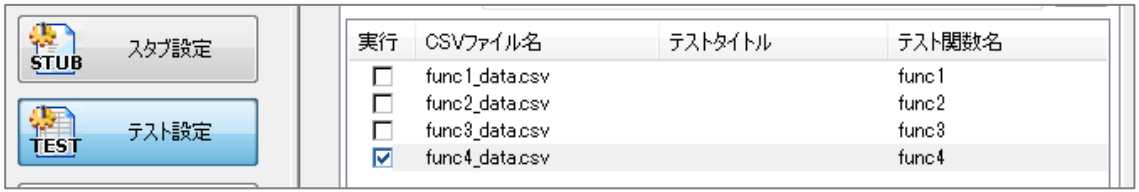


# 実習4-18：100%にならないケースを試す

## ■ カバレッジが100%にならないケースを確認してみる

1. 「テスト設定」で「func4_data.csv」をダブルクリックする
2. エクセルを操作し、1つ目のデータの#COMENT列に ;（セミコロン）を入れてマスクする
3. 「ファイル」メニュー　→「上書き保存」を実行
4. テストを再実行する




| | A | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|---|
| 1 | mod | func4 | | 6 | 2 | | | | CPP |
| 2 | #COMMEN | @code | gb_a | gb_b | gb_c | gb_d | gb_out | func4@@ | gb_out |
| 3 | ; | 1 | 11 | 21 | 31 | 1 | | | |
| 4 | | 1 | 10 | 21 | 31 | 1 | | | |
| 5 | | 1 | 11 | 20 | 31 | 1 | | | |
| 6 | | 1 | 11 | 21 | 30 | 1 | | | |
| 7 | | 2 | 10 | 21 | 31 | 1 | | | |
| 8 | | 3 | 10 | 21 | 31 | 1 | | | |
| 9 | | 4 | 10 | 21 | 31 | 1 | | | |
| 10 | | | | | | | | | |

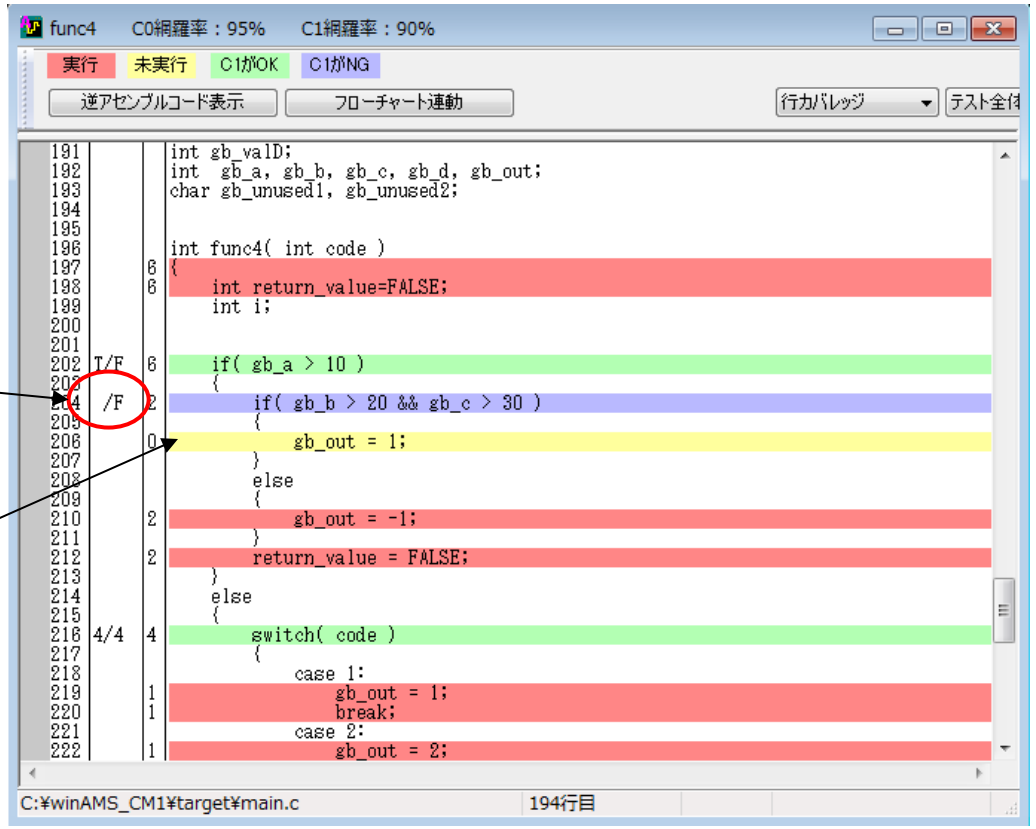
# 実習4-19：C0/C1カバレッジを確認

## ■ Func4_data.csvのカバレッジを確認

| テスト関数名 | C0 | C1 |
|---|---|---|
| func4 | 95% | 90% |

T（True）の論理が実行されていないことを示している

未実行行



# テスト作業をより効率化する
# 「加速オプション」
# （オプション機能）

## ※ CasePlayer2との連携機能

# 加速オプションとは

## ■ 構造カバレッジ（C1、MC/DC）計測を完全自動化するオプション

- 関数を選択するだけで、テスト設計からカバレッジ結果出力までを自動化
- ソースコードの静的解析により テストケースを生成

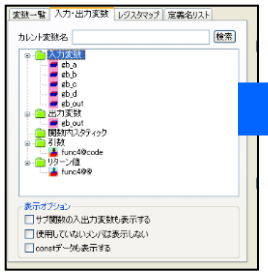
＜自動化される工程＞

- 関数入出力変数の自動設定 （テストCSV雛形自動作成）
- C1、MC/DCテストデータ自動生成
- 単体テストの自動実行 （入出力テスト、カバレッジ計測）

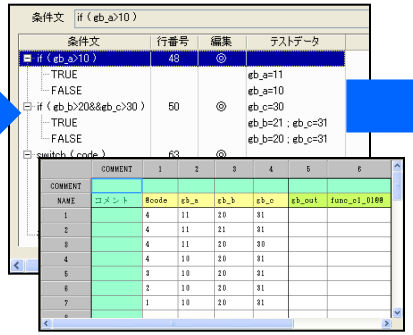
テスト対象
関数を選択


＜自動化される工程＞

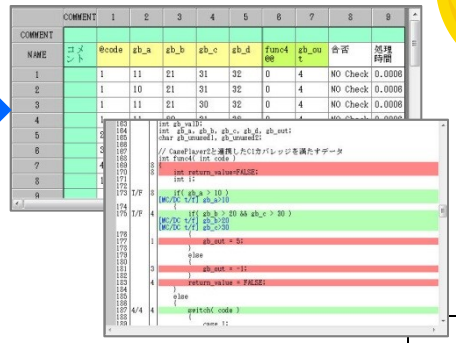
関数入出力変数の自動設定



C1、MC/DCテストデータ自動生成




単体テストの自動実行と結果出力



関数を選ぶだけで
直ぐに構造カバレッジ
結果が見られます！

# 加速オプションの利用手順

## ■ テスト自動化の設定

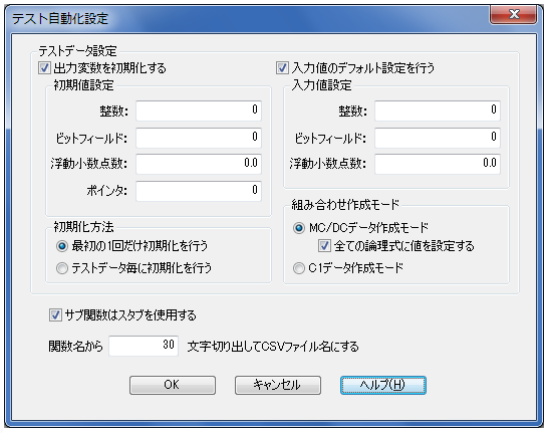
- 自動抽出された入力変数に対する初期値
- 生成するカバレッジデータの選択（C1 or MC/DC）
- スタブ関数の使用切り替え（使用/不使用）

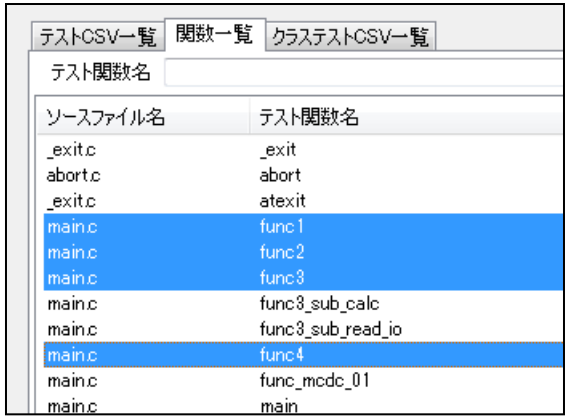
## ■ テスト対象関数の選択

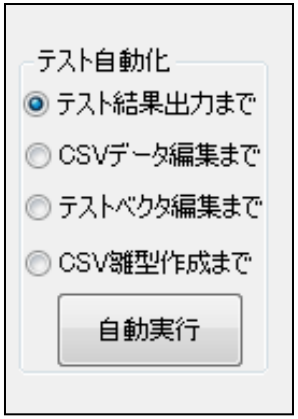
- リストから対象関数を選択（Ctrlキーで複数選択可能）

## ■ テスト自動化オプションの選択

- どの工程までを自動で行うかを選択可能



# （参考）加速オプション利用時の注意点

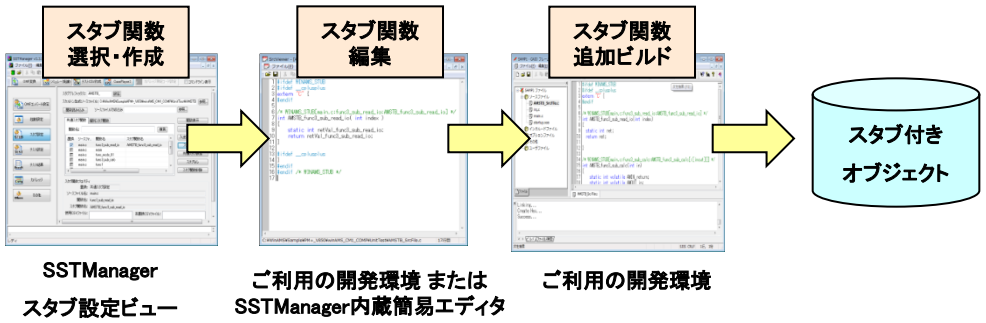
## ■ 必要な準備：　スタブ関数の事前作成

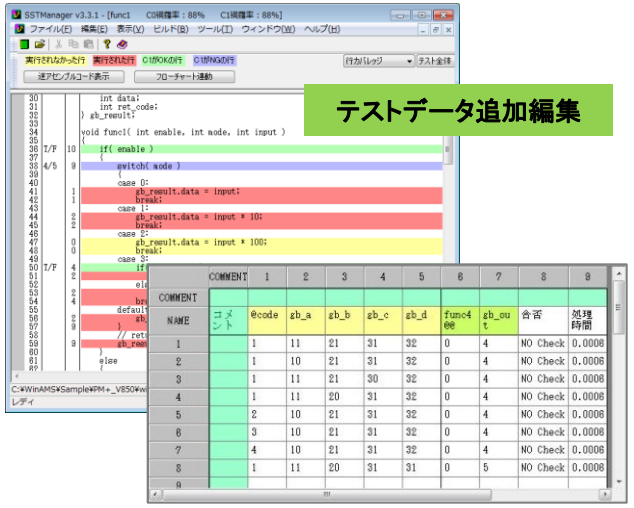
– 関数のテストに必要なスタブ関数は、事前に作成しておく必要があります。

- スタブ関数の雛形作成
- スタブ関数へのコード追加、編集
- スタブ関数の追加ビルド＆リンク

## ■ カバレッジ未達の場合のテストデータ追加編集

– 自動実行結果でカバレッジ未達の場合は、出力されたテストデータを調整し、再テスト（従来の手動操作）を行う必要があります



# MC/DCカバレッジ測定機能
# （オプション機能）

## ※ CasePlayer2との連携機能

# MC/DCは条件式の個別評価を求められる

## ■ MC/DCは 複合条件式に含まれる各条件式の個別評価を要求する

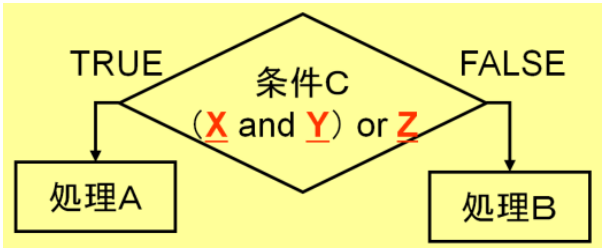
- 複合条件式の中から1つの条件式に着目
- 着目した条件式の論理のみを変化で、分岐の論理が変化するテストを行う

例：（X and Y）or Z　：条件式Xに着目した場合

- Y,Zは変化させず、XのみをTRUE/FALSEに変化させた場合に分岐の論理が変化するテスト行う

- （**X**, Y, Z）→ （**FALSE**, TRUE, FALSE）と （**TRUE**, TRUE, FALSE）

FALSE　　TRUE



# MC/DC計測のためにはコード挿入が必須

- **実コードそのままの状態では各条件式の動作論理を取得できない**
  - 原理的に、複合条件に含まれる各条件式を個別に評価すること（TRUE/FALSEどちらの論理で実行されたかを知ること）はできない
- **MC/DC計測用のフックコードを挿入した「埋め込みコード」で計測**
  - 条件式をフック関数に引数として送り、フック関数側で実行論理をツールに出力する
  - 埋め込みコードは、CasePlayer2により、評価対象の関数から自動生成される

```
if( ( x>10 && y>20 ) || z>30 )
```

←元の複合条件式

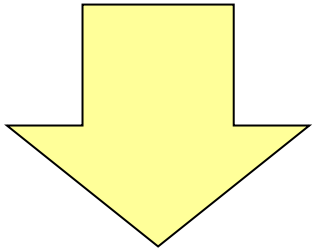
↓ フックコードを挿入

```
if(Hook(Hook(x>10) && Hook(y>20) ) || Hook(z>30) )
```

↑各条件式の論理をHook()関数に引数で送る
Hook()に渡された引数値から実行された論理を知ることができる

# MC/DCカバレッジ測定機能：テスト品質を維持

■ カバレッジマスターwinAMSは、
実コードによるテスト品質を維持したまま MC/DC計測が可能なツール



カバレッジマスターが実コードに忠実な
単体テストが可能な仕組み

■ テスト実行結果（出力）は「実コード」実行結果から取得
- 出力結果は ソースに手を加えない「実コード」から取得（従来の標準機能と同じ）

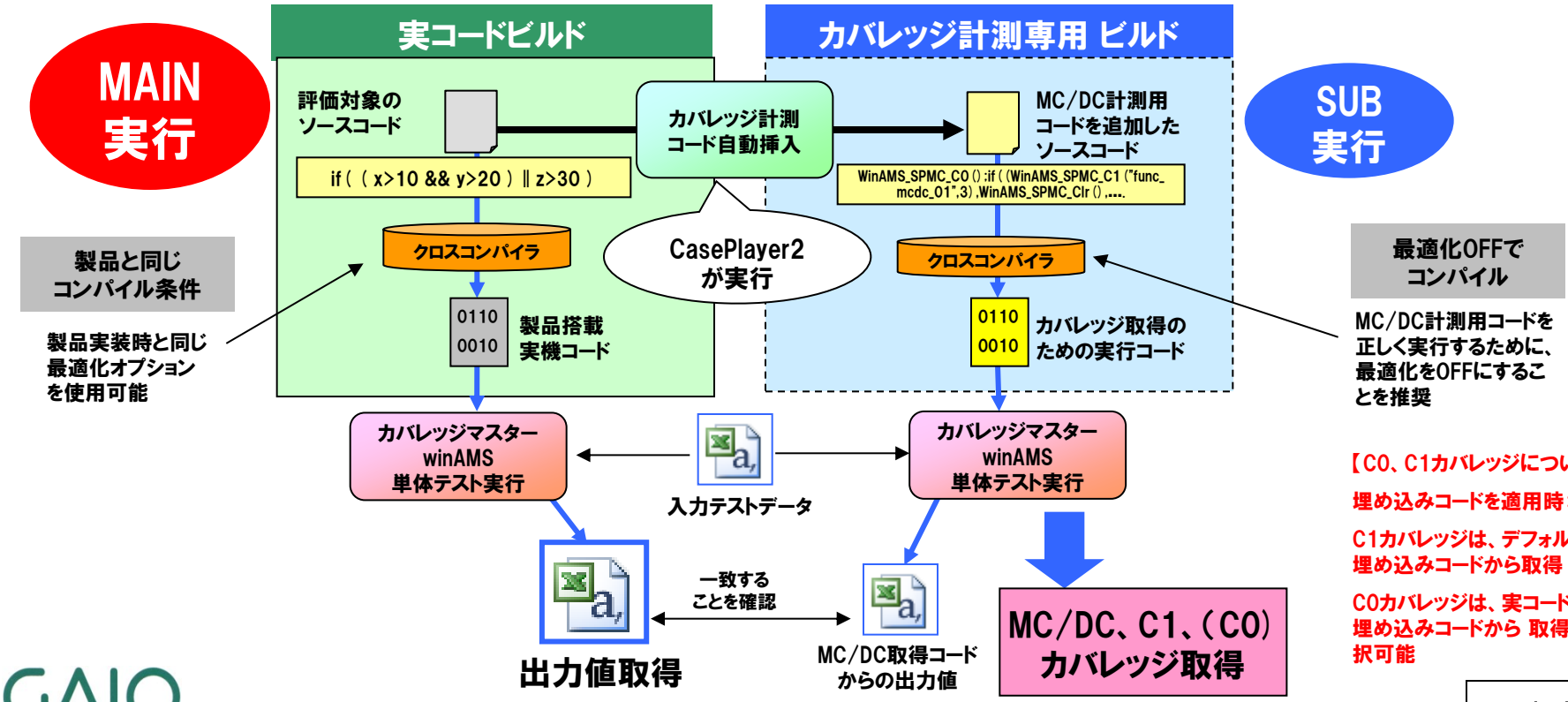
■ カバレッジ結果はフックコードを挿入した「埋め込みコード」から取得
- 論理的なカバレッジのみ、計測用フックコードを使って計測

■ 「実コード」出力値と「埋め込みコード」出力値が一致することを確認
- MC/DC条件解析コードを追加しても、関数出力値に相違が無いことを確認
- 相違が無いことを持って MC/DCカバレッジ結果の確からしさを検証

GAIO TECHNOLOGY

# MC/DCカバレッジ測定機能：テスト品質を維持（続）

■ 「実コード」と「埋め込みコード」の両方のオブジェクトを同時実行
- MCDC計測には、複合条件式を分解する解析コードを自動挿入し実行
- 関数出力値、期待値評価には、製品実装と同じ「実コード」から取得




【C0、C1カバレッジについて】

埋め込みコードを適用時：

C1カバレッジは、デフォルトで埋め込みコードから取得

C0カバレッジは、実コードから/埋め込みコードから 取得 を選択可能

# MC/DCカバレッジ測定機能：解析コード挿入イメージ

## ■ 各条件式を「フック関数」に送るコードに変更　（コード挿入イメージ）

**MC/DCカバレッジ 評価を行う 対象関数**

```
int func_mcdc_01( int x, int y, int z )
{
    if( ( x>10 && y>20 ) || z>30 )
    {
        return TRUE;
    }
    else
    {

    }
    return FALSE;
}
```

コード挿入

**カバレッジ計測のために自動生成された フック関数挿入後の関数**

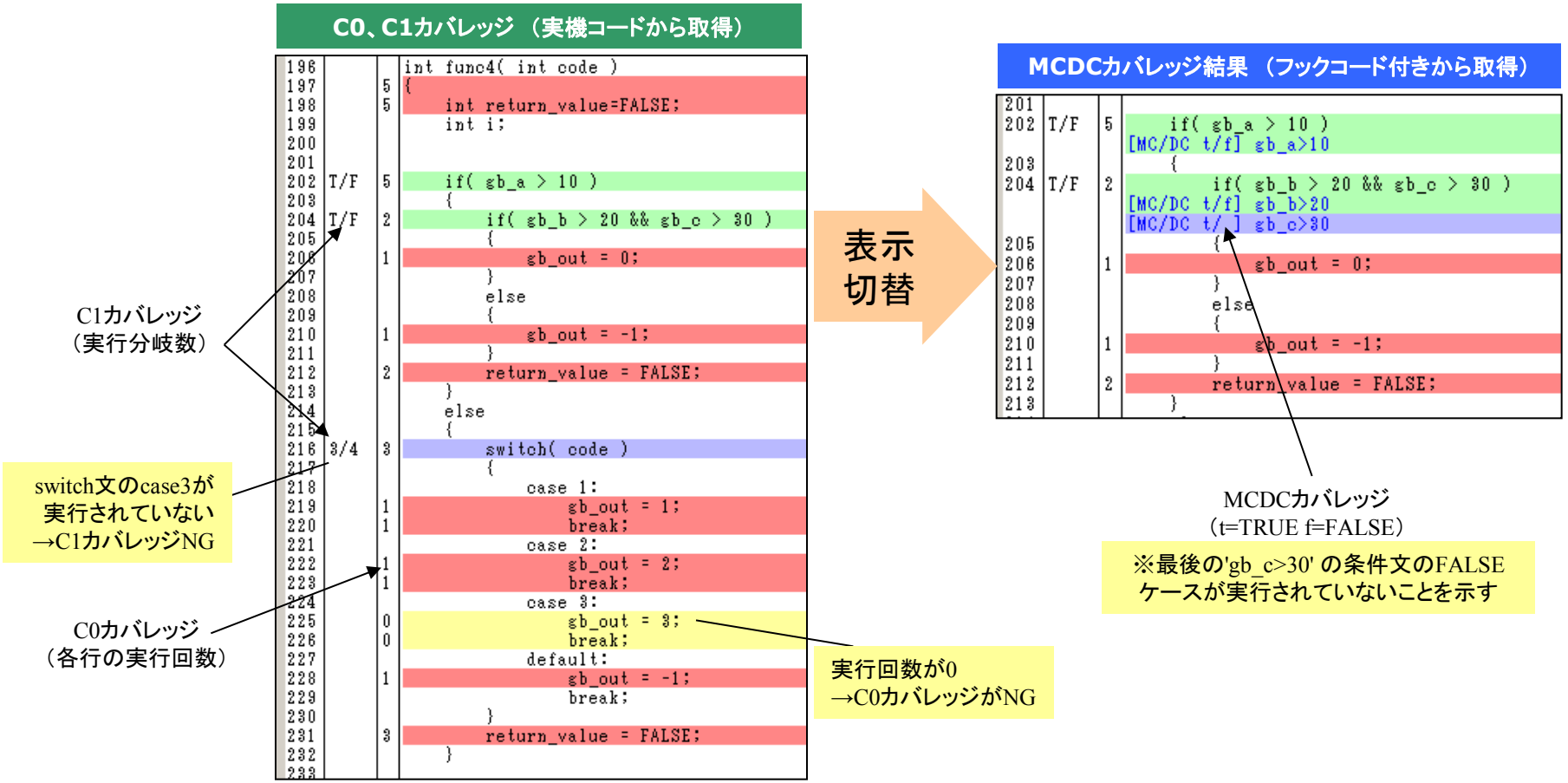
```
#line 252
int func_mcdc_01( int x, int y, int z )
{
if((WinAMS_SPMC_C1("func_mcdc_01",3),WinAMS_SPMC_Clr(3),WinAMS_SPM
C_Res("func_mcdc_01",1,( (WinAMS_SPMC_Exp(0,( x>10))
&&WinAMS_SPMC_Exp(1,( y>20)) )
||WinAMS_SPMC_Exp(2,( z>30))),3,3)) )
{
 WinAMS_SPMC_C1("func_mcdc_01",4);return 1;
}
else
{
#line 261
} WinAMS_SPMC_C1("func_mcdc_01",7);
return 0;
}
```

**フック関数 （この関数が生成する情報をカバレッジマスターが読み取る）**

```
BOOL WinAMS_SPMC_C1(WinAMS_SPMC_TFUNCNAME funcname,U4 blkID)
{
#if defined(WinAMS_SPMC_DI) && defined(WinAMS_SPMC_EI)
    WinAMS_SPMC_DI();
#endif
    if (WinAMS_SPMC_Lock == 0) {
        WinAMS_SPMC_Lock = 1;
#ifdef WinAMS_SPMC_CVT_FUNCNAME
        WinAMS_SPMC_funcname = WinAMS_SPMC_CVT_FUNCNAME(funcname);
#else
        WinAMS_SPMC_funcname = funcname;
            :
            :
```

```
BOOL WinAMS_SPMC_Exp(U2 expID,BOOL exp)
{
#if defined(WinAMS_SPMC_DI) &&
defined(WinAMS_SPMC_EI)
    WinAMS_SPMC_DI();
#endif
    if (WinAMS_SPMC_Lock == 0) {
        U2 nest;
        WinAMS_SPMC_Lock = 1;
        nest = WinAMS_SPMC_nest-1;
        if (expID < WINAMS_SPMC_MCDC_MAXCONDCNT &&
            nest < WINAMS_SPMC_MCDC_MAXCONDNEST) {
            U1 amsbit = exp ? 0x3 : 0x2;
```

# C0、C1、MC/DCカバレッジ計測結果表示



# ※CM1　追加実習
# MC/DCカバレッジ計測機能
# 環境作成

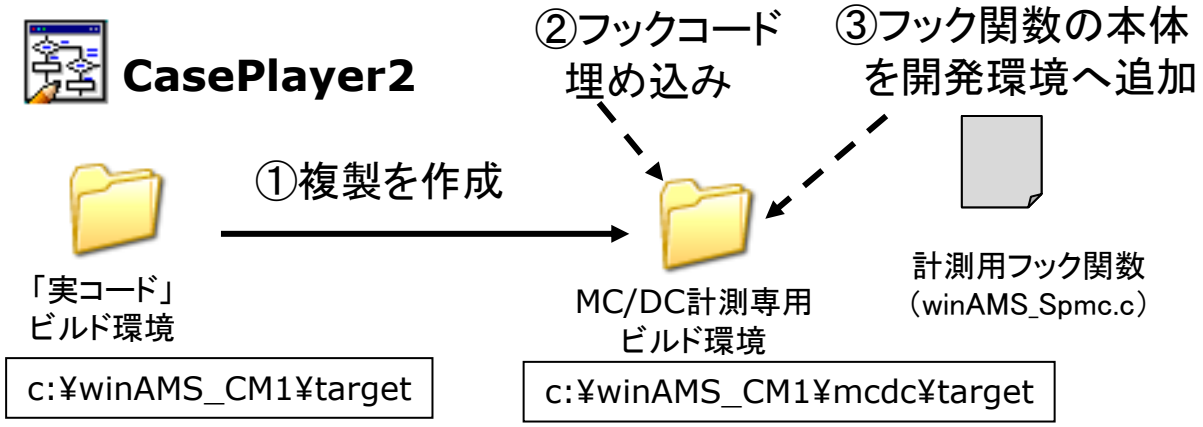
この実習は通常のセミナーには含まれません

# MC/DC計測①: 環境作成　概要

## ■ MC/DC計測のための埋め込みコード実行環境作成の流れ

– 「埋め込みコード」を生成するためのビルド環境を、もう1つ作成する

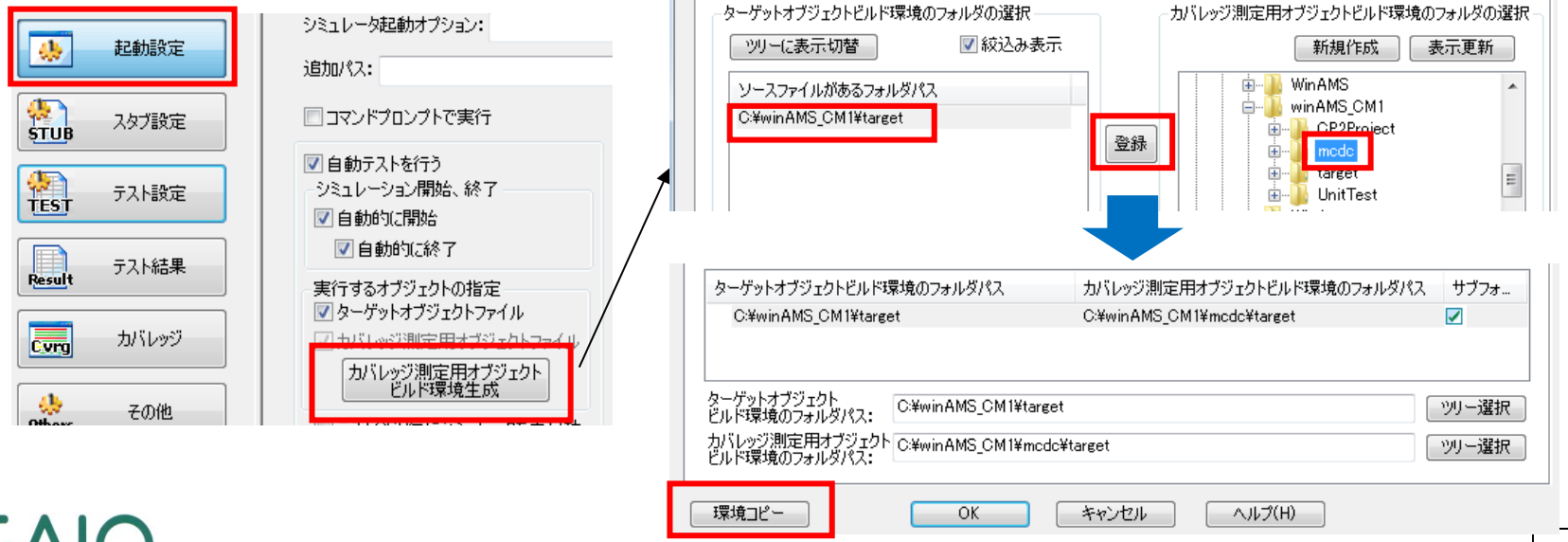
1. CasePlayer2の機能を使用して、「実コード」開発環境をフォルダごと複製する。
2. 複製された開発環境のソースコードに、CasePlayer2がフックコードを埋め込む。
   - CasePlayer2「MC/DC測定用コード生成」オプションがONであることを確認
3. CasePlayer2により生成されるフック関数の本体を含むソースファイルを開発環境に加える
4. 複製した環境でビルドし、実行可能なMC/DC計測用オブジェクトコードを作成する。



# MC/DC計測②: 埋め込みコードのフォルダを指定

## ■ 埋め込みコードを実行するための環境を作成します

1. 「起動設定」ビューの「カバレッジ測定用オブジェクトビルド環境生成」ボタンを押します
2. 「カバレッジ測定用オブジェクトビルド環境フォルダの選択」ツリーで、「C:¥winAMS_CM1」の下に、新規作成ボタンで「mcdc」フォルダを作成します。
3. 「C:¥winAMS_CM1¥target」と、mcdcフォルダをクリックして、「登録」ボタンを押します。
4. 「環境コピー」ボタンを押すと、mcdcフォルダの中に、targetフォルダのコピーが作成されます。



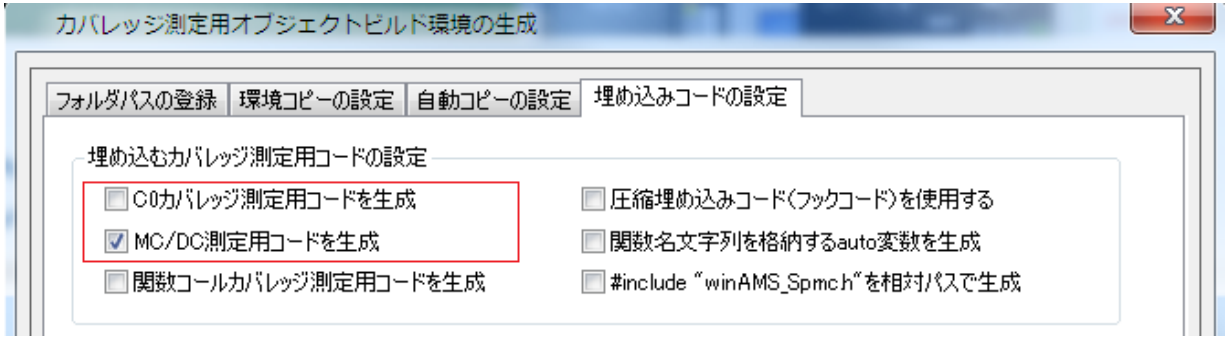
# MC/DC計測③: 埋め込みコードの種類を設定

## ■ 埋め込みを行うカバレッジの種類を確認します

1. CasePlayer2の「プロジェクト」メニューから「カバレッジ測定用オブジェクトビルド環境生成」を選択します
2. ダイアログの「埋め込みコードの設定」タブを選択します。
3. 「MC/DC測定用コードを生成」のオプションがONになっていることを確認します。

## ■ （参考）C0、C1カバレッジも「埋め込みコード」から取得する

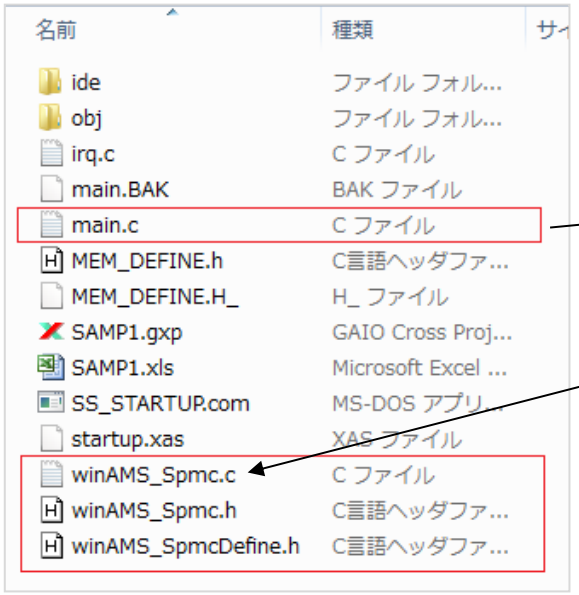
- コンパイラの最適化による影響でC0、C1カバレッジが不完全となる場合に適用
- 埋め込みコード適用時は、C1カバレッジはデフォルトで有効
  C0カバレッジのみ埋め込みコード適用の有無を切り替え可能
- MC/DCオプションがない場合でも使用可能



# MC/DC計測④: 埋め込みを実行

■ **複製したフォルダのソースファイルに フックコードを埋め込みます**

1. CasePlayer2の「プロジェクト」メニューから「全仕様書生成」を選択します。
   - 埋め込みコードが生成され、複製された環境のソースに書き込まれます
   - 計測用フック関数のファイルが、複製されたtargetフォルダに生成されます



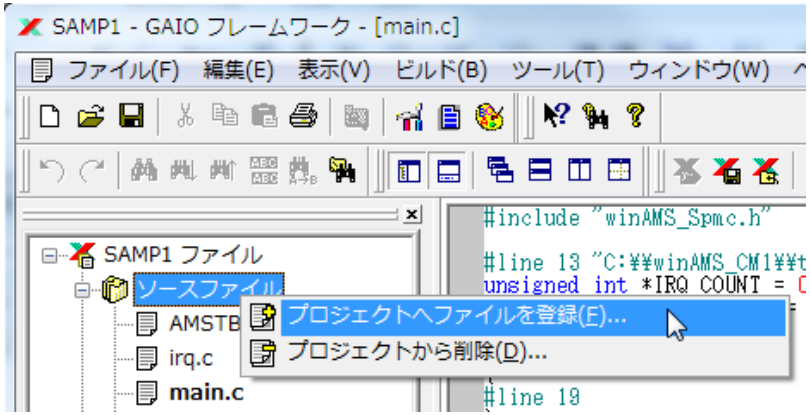

計測用フック
関数のファイル

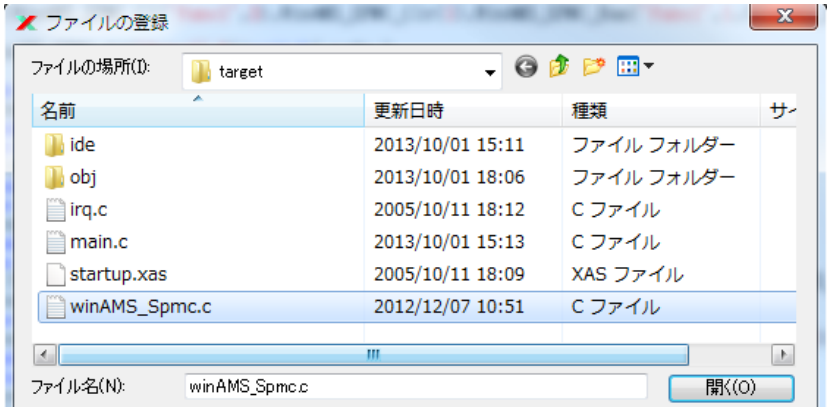

**「全仕様書生成」**
**実行後**
main.cに書き込まれた
埋め込みコード

# MC/DC計測⑤: フック関数ソースを追加ビルド

## ■ 計測用フック関数を開発環境（IDE）に追加してビルドします

1. 複製したtargetフォルダ内の「SAMP1.gxp」をダブルクリックして、フレームワークを起動します
   - C:¥winAMS_CM1¥mcdc¥target¥SAMP1.gxp
   - ※運用時には、ご利用中の開発環境（HEW、Softune、PM＋など）を利用
2. ツリーの「ソースファイル」を右クリックして、「プロジェクトへファイルを登録」を選択し、計測用フック関数のファイル「winAMS_Spmc.c」をプロジェクトへ追加します
3. 「ビルド」メニューから「リビルド」を実行します



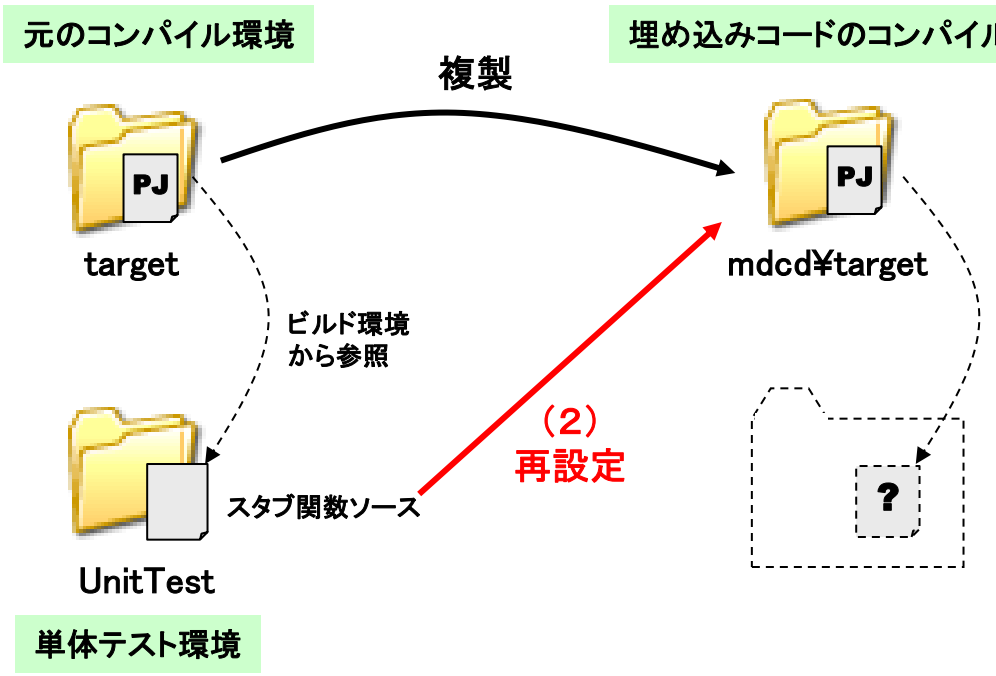



# （参考）スタブ関数ソースファイルの保存場所

## ■ スタブ関数ソースは 通常、開発プロジェクトとは別管理されている

スタブ関数ソースは 単体テストのプロジェクトに管理される場合が多い

- （1）開発環境、ソースコードの複製時にスタブ関数ソースは複製されない
- （2）埋め込みコードのコンパイル環境にスタブ関数ソースを再設定する必要がある

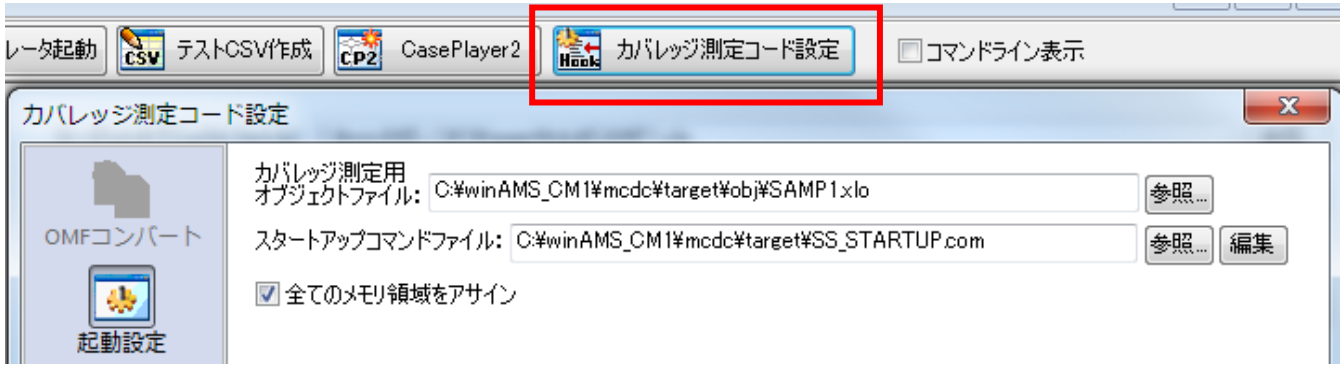



（1）
パスが変わるため
複製されたビルド環境から
スタブ関数ソースが
参照できない

# MC/DC計測⑥: カバレッジ測定コードを指定

## ■ 作成したMC/DC計測用オブジェクトをSSTManagerに登録します

1. SSTManagerの「カバレッジ測定コード設定」ボタンを押します
2. オブジェクトファイルの項目に、複製した埋め込みコードのフォルダに生成されたオブジェクトコードを指定します。 ※元のターゲット環境と間違えないようにして下さい！
   - C:¥winAMS_CM1¥mcdc¥target¥obj¥SAMP1.xlo
3. スタートアップコマンドファイルの項目に、複製した埋め込みコードのフォルダに生成されたスタートアップコマンドを指定します。
   C:¥winAMS_CM1¥mcdc¥target¥SS_STARTUP.com
4. 「すべてのメモリ領域をアサイン」をチェックします
5. 「OK」で完了します

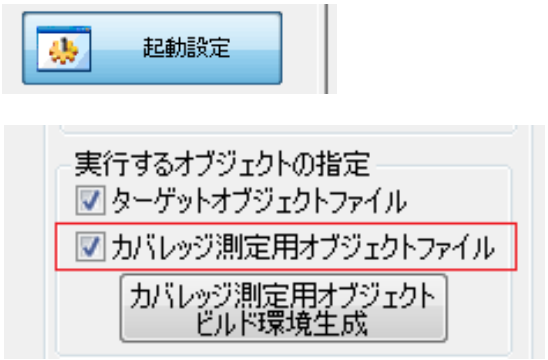


# MC/DC計測⑦: MC/DCカバレッジオプションをON

## ■ SSTManagerにMC/DC測定の設定を追加します

1. 「起動設定」ビューの「カバレッジ測定コード埋め込みオブジェクトファイル」をチェックします
   - これにより、埋め込みコードがシミュレータで実行されます
2. 「テスト設定」ビューの「MC/DCを出力する」をチェックします
   - これにより、MC/DC計測結果が出力されます

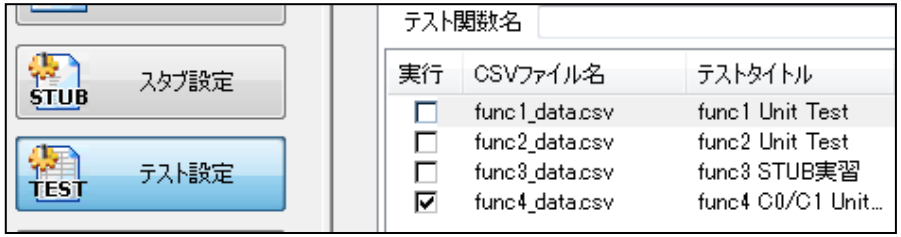
- これでMC/DC計測の環境設定は終了です

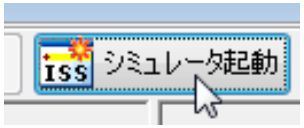



# MC/DC計測⑧: テスト実行

## ■ MC/DC計測を実行して結果を確認します

1. 「テスト設定」のボタンを押します。
2. テストCSV一覧のタブで、「func4_data.csv」の実行ボックスをチェックします。
3. 「シミュレータ起動」ボタンを押します。

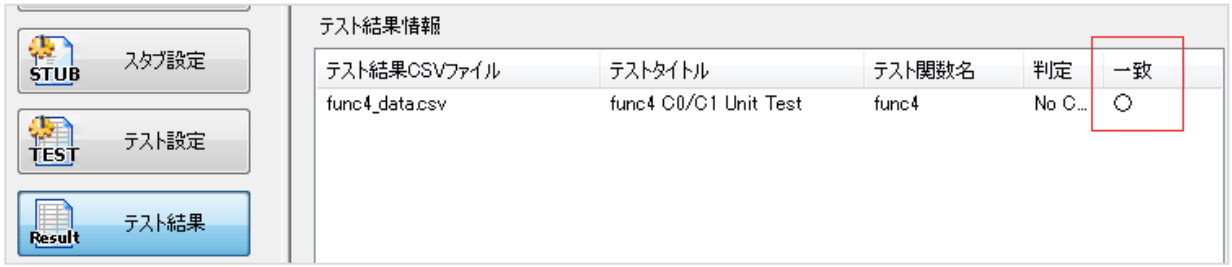


# MC/DC計測⑨: 結果確認

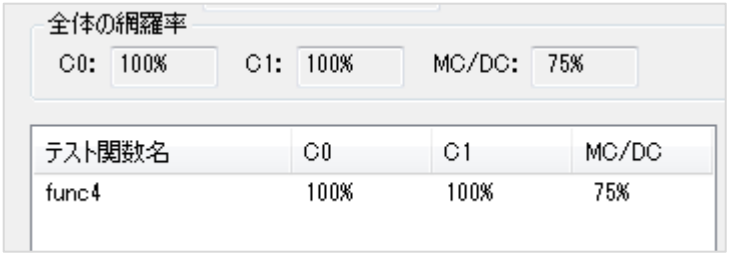
## ■ テスト結果ビューに、func4()のテスト結果が表示されます

- 「一致」の〇印：「実コード」と「カバレッジ測定用コード」の出力変数の値が一致している
  →埋め込みコードにより、関数本来の分岐などの動作に影響を与えていないことを示す




## ■ カバレッジ結果を確認

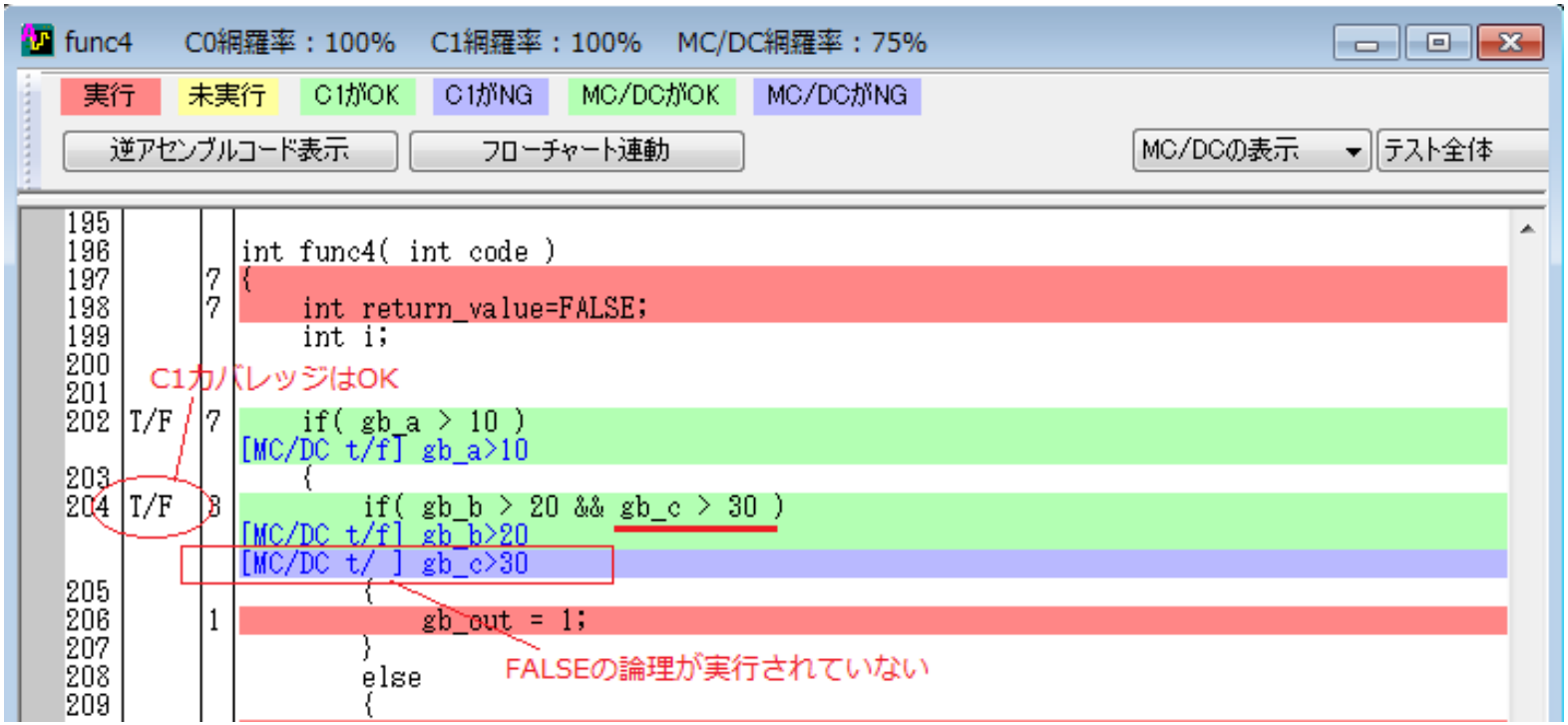
- func4()は C0、C1は100%でOKだが、MC/DCは未達で75%

# MC/DC計測⑩:　カバレッジビュー確認

## ■ MC/DCが未達の箇所を確認

複合条件式 if( gb_b>20 && gb_c>30)のうち
gb_c>30の条件式のfalseの論理がテストされていない

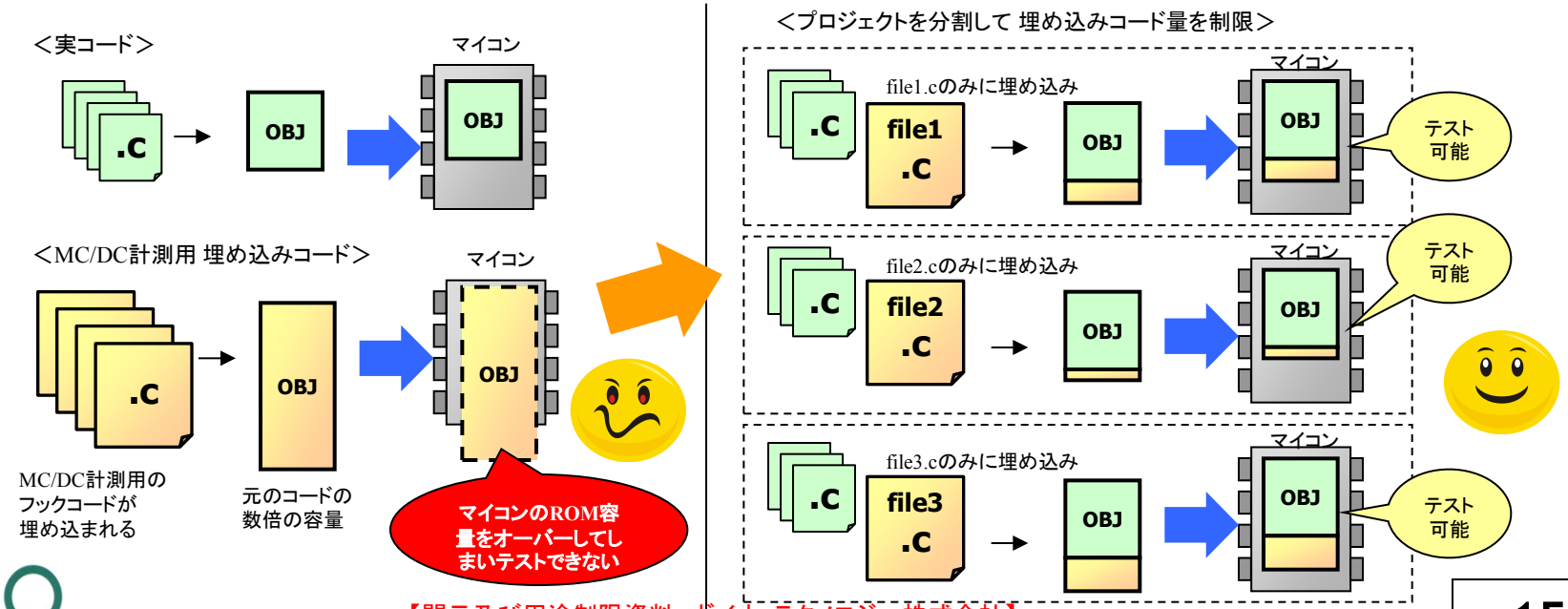


# （参考）埋め込みコードサイズ増加への対応方法

## ■ MC/DC計測のための埋め込みコードは 元のコードの数倍の容量になる

- 計測用の埋め込みコードがマイコンのROM容量をオーバーしてしまう場合がある
- 埋め込みコード生成対象のソースを制限して、プロジェクトを管理する必要がある
- CasePlayer2の「埋め込みコードの設定」機能が利用可能
  - CasePlayer2 → [プロジェクトメニュー] → [カバレッジ測定用オブジェクトビルド環境生成]

**「埋め込みコードの設定」機能を利用した 埋め込みソースの制限による コードサイズの増加抑制**



# MC/DCの意味と
# MC/DC未達の場合の対処法

# （参考）コントロールカバレッジ

## ■ コントロール（実行制御に対する）カバレッジ

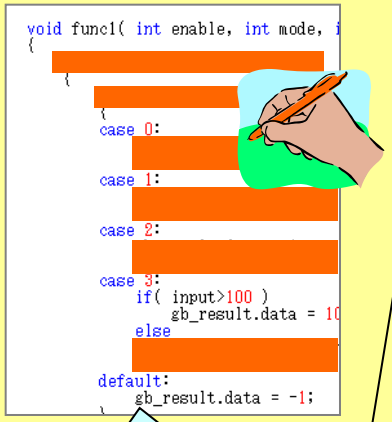
### C0：ステートメント（命令）カバレッジ

【指針】全ての実行文を1度実行すればOK

【保証できる内容】

・コーディングしたが実行されない実行文はない

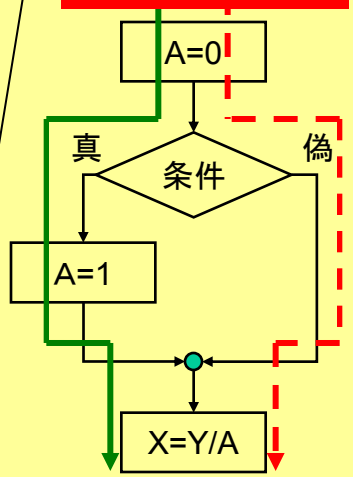
・存在してはいけない実行文はない




C0カバレッジ：

ソース行をマークして、全て塗りつぶす

C0では十分でない例



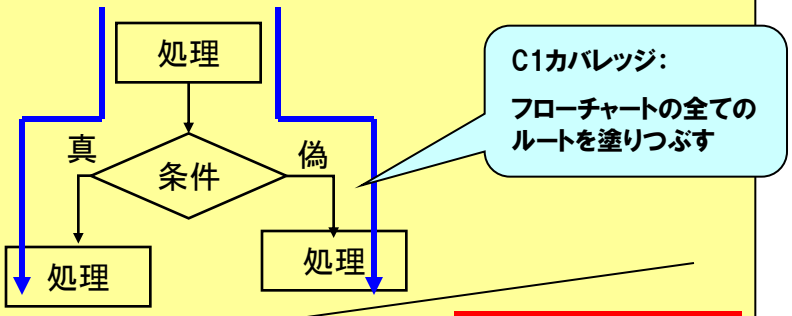

赤のパスには実行文がないので、C0ではテストされない

### C1：ブランチ（分岐）カバレッジ

【指針】全ての条件文の真/偽を実行すればOK

【保証できる内容】

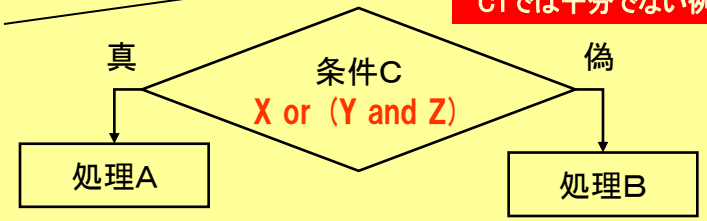
・存在してはいけない実行パスは存在しない




C1カバレッジ：

フローチャートの全てのルートを塗りつぶす

C1では十分でない例




複合条件では1つのTRUE/FALSEがテストされるのみ
全ての組合せがテストされる訳ではない

# （参考）コントロールカバレッジ（続）

## ■ コントロール（実行制御に対する）カバレッジ　（続き）

MC/DC:
全ての条件式のTRUE/FALSEの効果を確認する

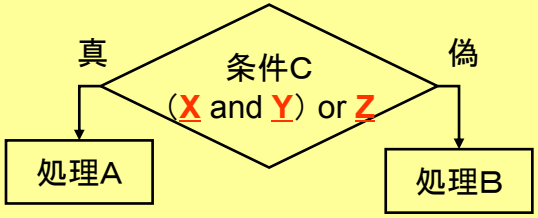
### C2：コンディション（条件）カバレッジ

【指針】全ての条件文の条件式の組み合わせを1度実行すればOK

【保証できる内容】

・存在してはいけない条件判定式の要素がないこと

C2カバレッジ:
複合条件の全数組合せをテストする




全ての条件文の条件式の組み合わせを実行

| 条件式 X | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|
| 条件式 Y | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 条件式 Z | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

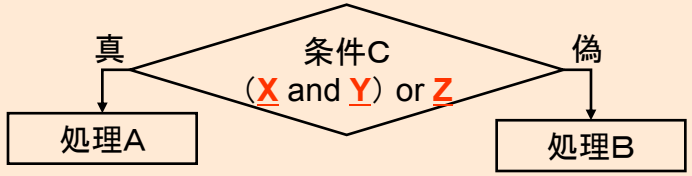
条件要素に対する網羅性は100%だが、
効果のないテストが含まれてしまう

### MC/DC：Modified Condition/Decision カバレッジ

【指針】全ての条件文の条件式の組み合わせのうち、意味のある組み合わせを全て1度実行すればOK

【保証できる内容】

・C2と同等の網羅内容（テストデータはC2より少ない）




他の条件式の真偽を固定して、1つの条件式の真偽を変化させたとき、全体の条件に変化があるものだけを実行する（↓色のついたデータ）

| 条件式 X | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|
| 条件式 Y | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 条件式 Z | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 論理 | F | T | F | T | F | T | T | T |

2つのデータ組み合わせで、各条件式を評価→ Z X Y

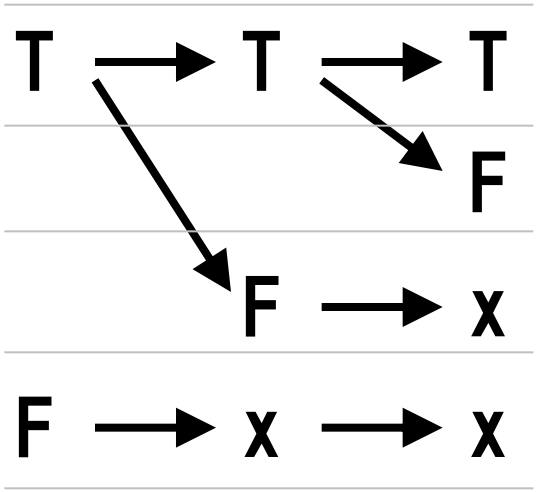
C2の網羅性を維持して、テストパターン数を抑えることができる

# （参考）MC/DCテストケースの設計方法

■ （例1）論理演算の順序に従って、論理解析を行う

## 論理解析

**A && B && C**




※ x は評価の必要が無い

## テストケースの組合せ

| A | B | C | 論理 |
|---|---|---|---|
| T | T | T | T |
| T | T | F | F |
| T | F | T/F | F |
| F | T/F | T/F | F |

Aの論理式をテストする
MC/DCテストケースの
見つけ方

B、Cを固定したとき
（B→True、C→True）
Aの論理を変化させた
とき、全体の論理が変
化するテストケースを選
択する

---

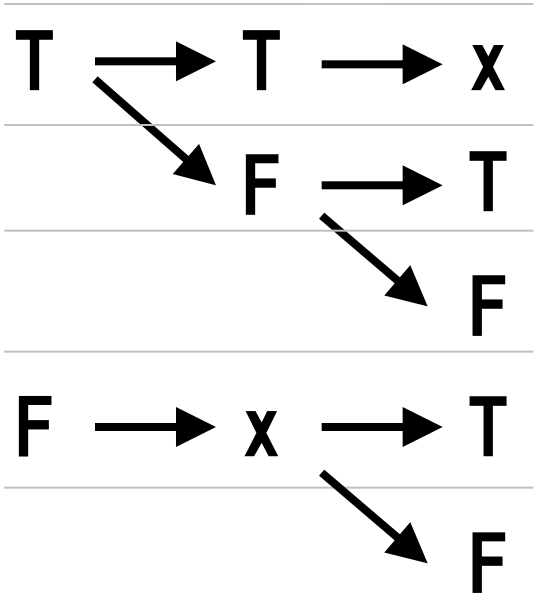
B、Cの論理式も同様

# （参考）MC/DCテストケースの設計方法

■ （例2）論理演算の順序に従って、論理解析を行う

論理解析

(A && B) || C




※ x は評価の必要が無い

テストケースの組合せ

| A | B | C | 論理 |
|---|---|---|---|
| T | T | T/F | T |
| T | F | T | T |
| T | F | F | F |
| F | T/F | T | T |
| F | T/F | F | F |

Aの論理式をテストするMC/DCテストケースの見つけ方

B、Cを固定したとき（B→True、C→False）Aの論理を変化させたとき、全体の論理が変化するテストケースを選択する

---

B、Cの論理式も同様

# MC/DCが未達の場合のデータレビュー方法1

## ■ ATDEditorの各論理に対する生成データを確認する

各条件式に同じ変数が使われている場合、論理矛盾を起こす場合がある

```
int func4( int code )
{
    int return_value=FALSE;
    int i;

    if( gb_a > 10 )
    {
        if( gb_b > 20 && gb_c > 30 )
        {
            // 処理
        }
        else
        {
            gb_out = -1;
        }
        return_value = FALSE;
    }
    else
    {
```

この条件式の
MC/DCが未達の場合

```
T/F  7    if( gb_a > 10 )
          [MC/DC t/f] gb_a>10
          {
T/F  2        if( gb_b > 20 && gb_c > 30 )
          [MC/DC t/f] gb_b>20
          [MC/DC t/ ] gb_c>30
              {
1             gb_out = 5;
```

| 論理式 | 行番号 | 編集 | 式種別 | テストデータ |
|---|---|---|---|---|
| if ( gb_a>10 ) | 190 | ◎ | x<>C | |
| TRUE | | | | gb_a=11 |
| FALSE | | | | gb_a=10 |
| if ( gb_b>20&&gb_c>30 ) | 192 | ◎※ | | |
| gb_b>20 | | | x<>C | |
| TRUE | | | | gb_b=21 |
| FALSE | | | | gb_b=20 |
| gb_c>30 | | | x<>C | |
| TRUE | | | | gb_c=31 |
| FALSE | | | | gb_c=32 |
| switch ( code ) | 204 | ◎ | -- | |
| case 1: | | | | @code=1 |
| case 2: | | | | @code=2 |
| case 3: | | | | @code=3 |
| default: | | | | @code=4 |
| if ( gb_d>gb_c ) | 222 | ◎ | x<>y | |
| TRUE | | | | gb_c=30 ; gb_d=31 |
| FALSE | | | | gb_c=31 ; gb_d=31 |
| その他 | --- | | | |

!

# MC/DCが未達の場合のデータレビュー方法2

## ■ EXCELで未達の条件式を評価するテストケースを見つける

```
int func4( int code )
{
    int return_value=FALSE;
    int i;

    if( gb_a > 10 )
    {
        if( gb_b > 20 && gb_c > 30 )
        {
            // 処理
        }
        else
        {
            gb_out = -1;
        }
        return_value = FALSE;
    }
    else
    {
```

この条件式の
MC/DCが未達の場合

```
T/F  7    if( gb_a > 10 )
          [MC/DC t/f] gb_a>10
          {
T/F  2        if( gb_b > 20 && gb_c > 30
          [MC/DC t/f] gb_b>20
          [MC/DC t/ ] gb_c>30
              {
     1            gb_out = 5;
```

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | mod | func4 | | 5 | 2 | |
| 2 | #COMMENT | @code | gb_a | gb_b | gb_c | gb_d |
| 3 | ;if ( gb_a>10 ) | | | | | |
| 4 | ;TRUE | | | | | |
| 5 | | 1 | 11 | 21 | 31 | 31 |
| 6 | ;FALSE | | | | | |
| 7 | | 1 | 10 | 21 | 31 | 31 |
| 8 | ;if ( gb_b>20&&gb_c>30 ) | | | | | |
| 9 | ;T&&T => T | | | | | |
| 10 | | 1 | 11 | 21 | 31 | 31 |
| 11 | ;T&&F => F | | | | | |
| 12 | | 1 | 11 | 21 | 32 | 31 |
| 13 | ;F&&T => F | | | | | |
| 14 | | 1 | 11 | 20 | 31 | 31 |
| 15 | ;switch ( code ) | | | | | |
| 16 | ;case 1: | | | | | |
| 17 | | 1 | 10 | 21 | 31 | 31 |
| 18 | ;case 2: | | | | | |
| 19 | | 2 | 10 | 21 | 31 | 31 |
| 20 | ;case 3: | | | | | |
| 21 | | 3 | 10 | 21 | 31 | 31 |
| 22 | ;default: | | | | | |
| 23 | | 4 | 10 | 21 | 31 | 31 |
| 24 | ;if ( gb_d>gb_c ) | | | | | |
| 25 | ;TRUE | | | | | |
| 26 | | 4 | 10 | 21 | 30 | 31 |
| 27 | ;FALSE | | | | | |
| 28 | | 4 | 10 | 21 | 31 | 31 |
| 29 | ;その他の組み合わせ | | | | | |

【レビューのフロー】

MC/DCに基づいて、
条件式「gb_c>30」を評価するテストデータは、

↓

「gb_b > 20 」の論理を同じ(T)にして、「 gb_c>30 」の論理を変化させた時に、if文全体の論理が変化するデータを選択

↓

この場合:
　9行目:　T&&T => T　と
　11行目:　T&&F => F となる

↓

このデータをレビューして修正する

↓

この場合、「 gb_c>30」の「f」の論理がテストされていないため、12行目のデータ[32]が間違っている

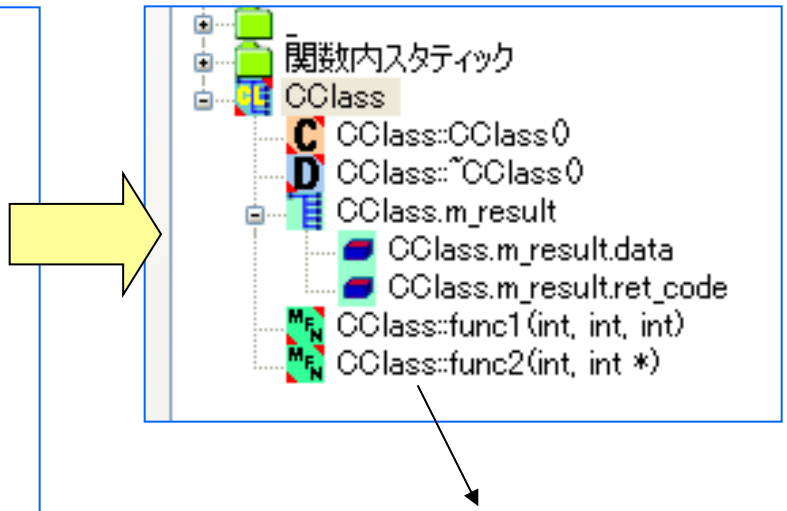
# C++コード 単体テスト機能
# （オプション機能）

## ※ CasePlayer2との連携機能

# C++に対応：　概要

- **実コードを使用したC++単体テストに対応**
  - ※ C++コード対応は、オプション機能
- **C++クラス内に定義されたメソッド（関数）の単体テストが可能**
  - テスト対象クラスの構造をツリーに表示

```
class CClass
{
public:
        CClass(){}   // コンストラクタ
       ~CClass(){}   // デストラクタ

        structST_PARAM     m_result;

        void func1( int enable, int mode, int input )
        {
          /* 処理分岐 */
               if( enable )
               {
                      switch( mode ) /* 4case */
                      {
                      case 0:
                             // input代入
                             m_result.data = input;
                             break;
                      case 1:
                             m_result.data = input * 10; // input * 10 代入
                             break;

                                          ⋮
                                          ⋮

                      }
                      // return code
                      m_result.ret_code = TRUE;
               }
        }
        void func2( int mode, int *data_in );

};
```




テスト対象メソッドや
初期化に必要なコンストラクタ
をツリーから選択

# C++に対応：　クラスオブジェクト生成

- **テスト時に クラスの定義からクラスオブジェクトを自動生成可能**
  - テスト時にクラスオブジェクトを自動生成可能
  - 静的オブジェクト内の関数（メソッド）を指定してテスト可能
- **初期化に必要なコンストラクタの実行を選択可能**
  - テスト時にクラスオブジェクトを生成する場合は、コンストラクタの実行選択が可能
    - ※静的オブジェクトの場合は、コンストラクタの実行はコンパイラのコード展開に依存

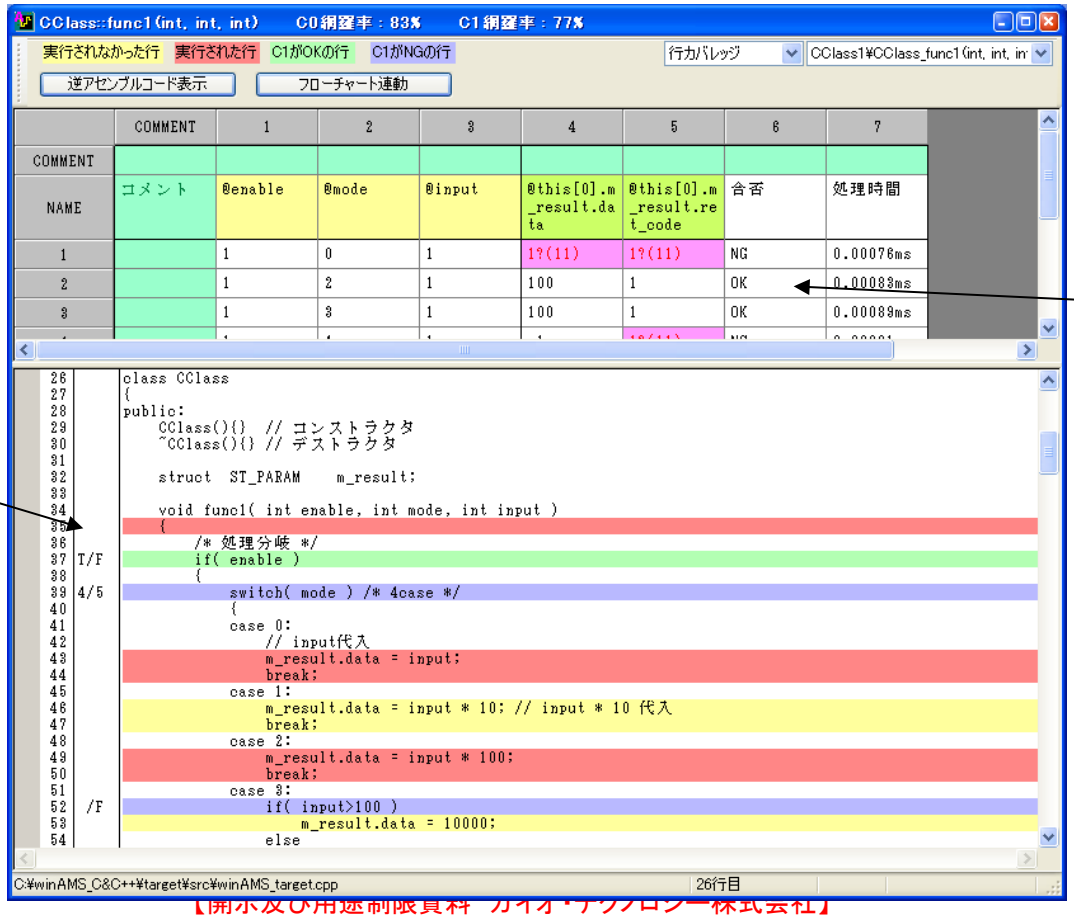


クラスオブジェクトを自動生成

コンストラクタの実行を選択

# C++に対応： C0/C1カバレッジ計測

- メソッドに対する入出力テスト、カバレッジ計測はCコードと同様に可能
  - ※MC/DCカバレッジ計測（オプション機能）はC++に未対応




入出力テスト結果

C0/C1カバレッジ計測

# ※参考資料
# カバレッジマスターの優位性と導入効果

# 単体テストツールの選定ポイント

■ 単体テストツールの比較を行う際には、
多くのお客様が以下の様な評価項目を設定しています

| 項目 | 必要条件 |
|---|---|
| 準備工数 | 製品開発ビルド環境をそのまま使えるか？　ソース変更は発生しないか？　テストドライバ作成の必要があるか？　スタブは自動生成可能か |
| カバレッジ機能 | C0、C1、MC/DC、関数、コール カバレッジは計測可能か？<br>カバレッジ計測結果を容易にレポート出力できるか？ |
| 精度/信頼性 | 実コードに忠実なテストが可能か？　エンディアン/ビットフィールド/アラインメント/ポインタサイズ/シフト演算 等 |
| データ資産化 | CSVファイルのみでテストデータ管理が可能か？回帰テストが自動化できるか？ |
| データ入力支援 | 構造カバレッジ計測のテストデータは自動生成可能か？　生成されるテストデータの冗長性は低いか？ |
| サポート体制 | 開発メーカー直接のサポートが受けられるか？レスポンスは良いか？<br>導入時のレクチャ／セミナーは充実しているか？ |

# 他社ツールとの比較とアドバンテージ

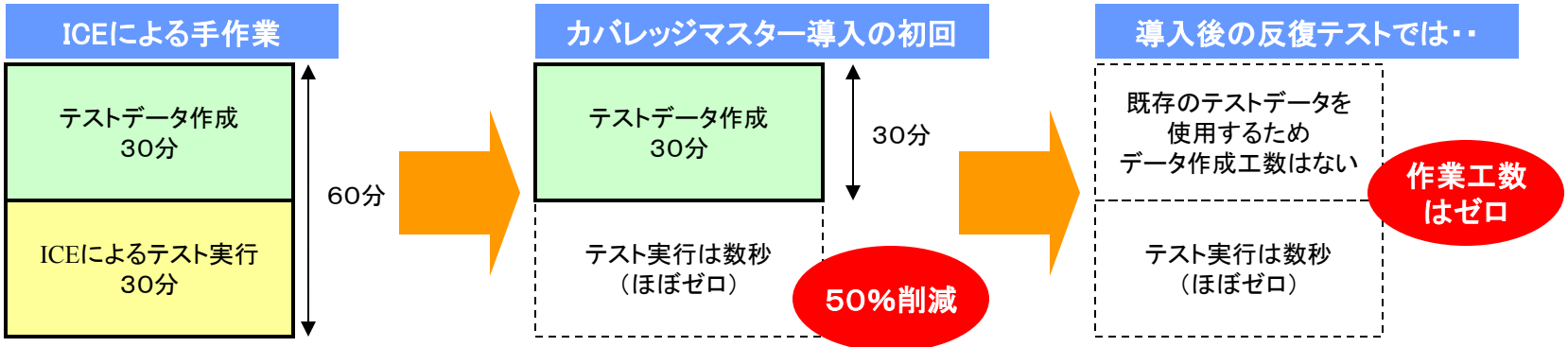
## ■ ツール構成/特長の比較

※各社ツールのWEBページ情報を基に作成 （作成日：2013年6月）

| 比較項目 | カバレッジマスターwinAMS | 他社ツール |
|---|---|---|
| **計測方法** | マイコンシミュレータ(ISS)で実コードをそのまま実行して単体テスト、カバレッジ測定 | PCに接続したターゲットボード上で、関数を駆動するテストドライバを実行し、測定結果をPCへ転送 |
| **ツール構成** | 単体テストアプリ、マイコンシミュレータ<br>※全てのモジュールがソフトウエアのみ<br>※常に安定動作する | 単体テストアプリ、ターゲットボード<br>PCとの接続ケーブル、電源装置<br>※ソフトツール、専用ハードウエアが必要<br>※ノイズなど物理的な影響を考慮する必要あり |
| | 全てソフトウエアであるため保守性が高い | 専用ハードウエアの保守が必要 |
| **環境セットアップ（テスト準備作業）** | 製品開発のビルドをそのまま使用可能<br>作業は発生しない<br>※デバッグ情報を付加してビルドするだけ | ターゲットボードに計測装置I/Fファームウエアの組み込みが必要、ターゲット毎にカスタマイズ(コンフィグレーション)を行う必要有り<br>※計測装置I/F ファームウエア← 計測した情報をボードからPCへ転送する仕組み |
| | テスト開始のリードタイムなし | 専用HWコンフィギュレーションが必要 |
| **テスト対象コード** | 製品に搭載するコードと同じオブジェクト | テスト対象関数にフックコード(プローブ)を挿入したテスト専用ビルドを別途準備する必要あり |
| | 製品へ実装するコードと同一<br>最もターゲットに忠実な状態でテスト | 製品へ実装するコードとは異なる<br>コードに挿入したフックコード(プローブ)による影響が避けられない |
| **ISO26262認証** | 認証取得済み | 認証取得済み |
| **カバレッジ機能** | C0, C1, MC/DC, 関数、コール | C0, C1, MC/DC, 関数、コール<br>（関数、コールをサポートしないツールもある） |

# ICEによる手作業が どのように改善されるか

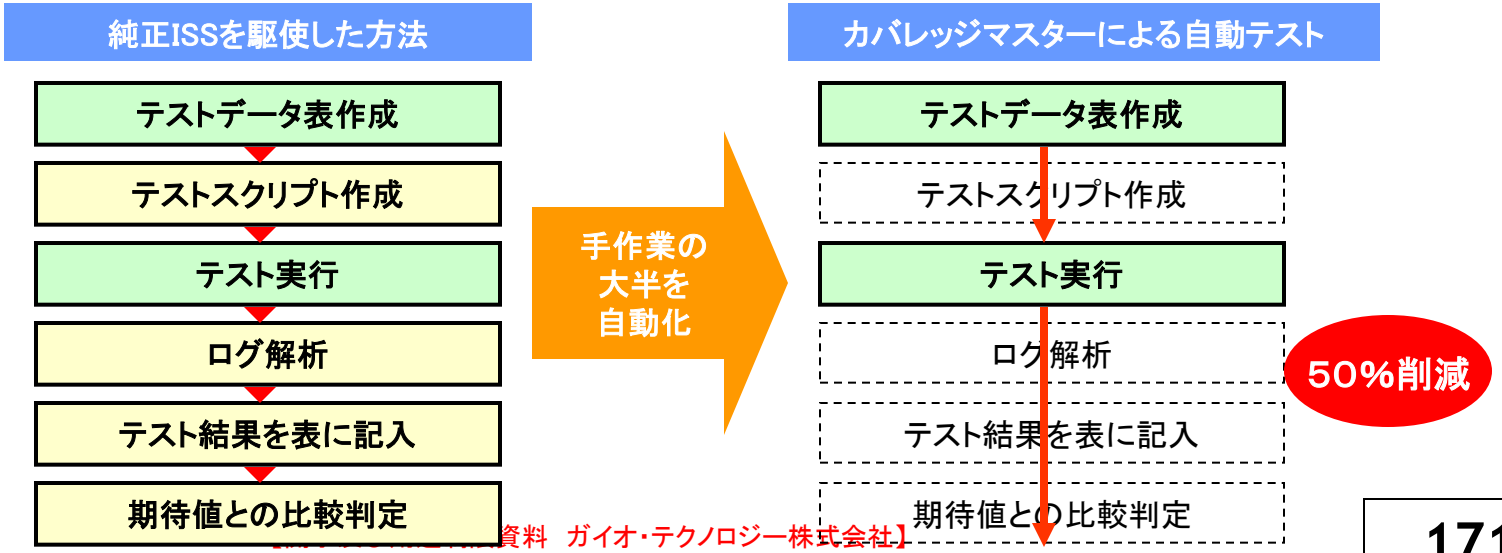
- **ICEを使用して、1関数60分で単体テストを行っている場合を例にとると・・・**
  作業時間の内訳は・・
  - ソース解析＆テストデータ作成：　30分
  - ICEによる手作業でのテスト実行：　30分
- **カバレッジマスターの導入により、初回のテストにおいては・・**
  - ソース解析＆テストデータ作成：　30分　←同じ時間がかかる
  - カバレッジマスターで自動実行：　数秒　←テストは自動化され作業工数はゼロ
- **2回目以降の反復テストでは**
  - ソース解析＆テストデータ作成、テスト実行の作業工数もゼロになる



# 他社ISSを使用した場合との比較

## ■ 他社（半導体メーカー純正）ISSを使用した単体テストとは

- ISS（デバッガ）がサポートするスクリプト言語を駆使して単体テストを行う方法
  - 関数の呼び出し指定、変数に対するデータ設定を行うスクリプトを作成
  - テスト実行後に実行ログファイルを解析して、評価対象の変数の値を確認
  - テスト結果を表に記入（手作業）
  - 期待値との比較判定（手作業）
- カバレッジをとることは困難、または膨大な手作業が必要
  - ステップ実行毎に、プリントしたソースコードをマーカーで塗りつぶす手作業となる




【資料　ガイオ・テクノロジー株式会社】

# ※参考資料
# テスト代行サービスについて

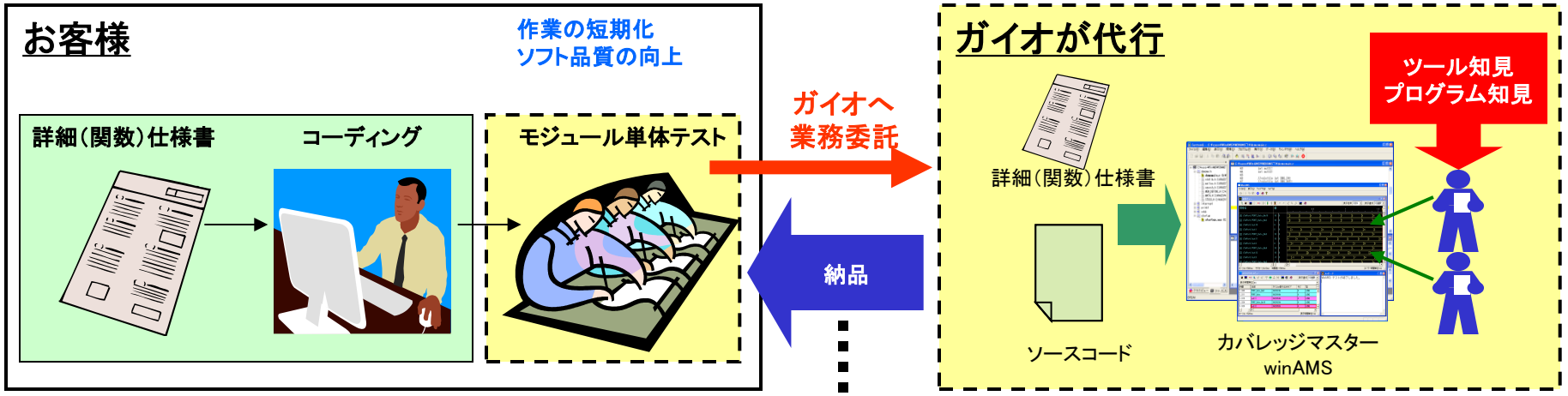
# 単体テスト代行サービスの概要

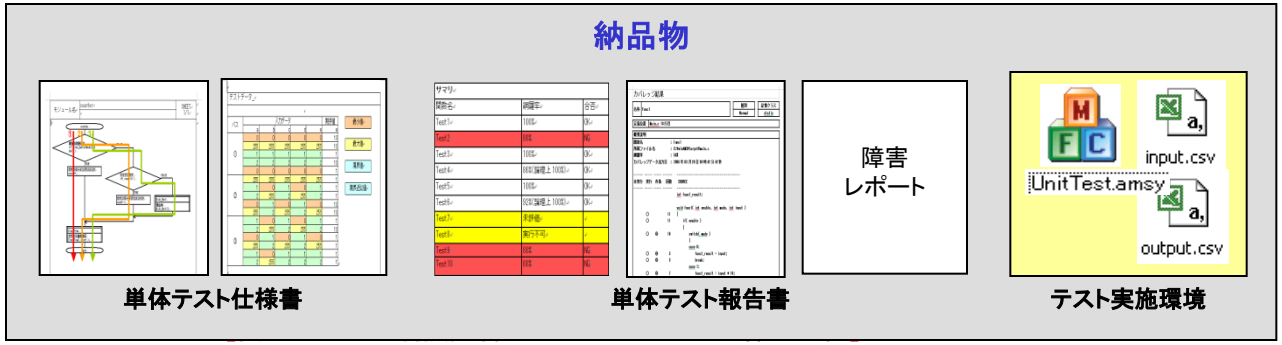
- **ガイオの優位性を活かして単体テストを実施**
  - 単体テストツールメーカとしてのツール知識の活用
  - コンパイラメーカとしてのプログラム知見の活用
- **お客様に代わってテスト業務を完全代行**
  - テスト作業のアウトソーシング
  - 第三者評価機関としての立場で品質評価
- **カバレッジマスター導入の評価に利用可能**
  - 御社のソース（数関数）で 単体テストを代行
  - ツール導入後の実工数、成果物の検討などに利用


# 単体テスト代行作業の全体イメージ

## ■ モジュール単体テスト工程の全てをガイオへ業務委託

- テスト作業の短期化と 品質向上を実現
- カバレッジマスターwinAMSの テスト実施環境・データー式、テストレポートを納品

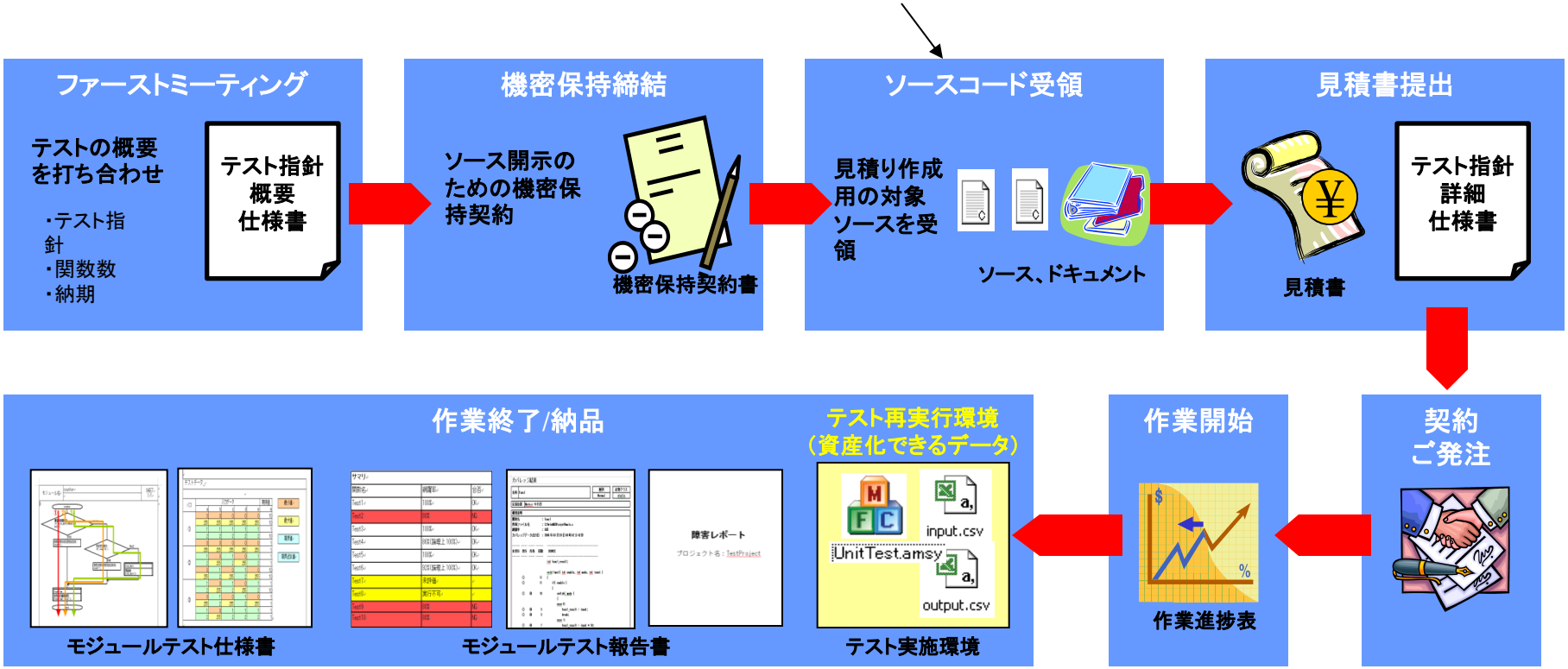


# 単体テスト代行の流れ

## ■ 最終納品までの流れと 各フェーズでの受け渡し書類

– お見積もりはご提供頂くソースコード、ドキュメントにより決定



# 納品物:テスト指針書

## ■ ご依頼元(お客様)との打ち合わせにより テストのゴールを決定

- 仕様書から読み取れる仕様の動作確認の方法
- ホワイトボックス的観点から カバレッジの指針、テストデータの与え方など
  - コーディングの人的ミスを発見するためのデータ作成指針

4．作業方針

■対象関数に対して定められたカバレッジ取得を行う。
■対象関数に対して定められたテストのデータを作成する。
■呼び出されるサブルーチン・システムコールはスタブ化する。
■出力値の期待値判定を行う。

4．1　カバレッジ

・対象カバレッジを明確にする為、下記の該当する文字（C0/C1/C2/MC/DC）を赤色とする。

| カバレッジ | C0 / C1 / C2 / MC/DC |
|---|---|

```
int func1(int input1, int input2){
  if (input1 == 1 || input2 == 2){
    処理 A;
  }
}
```

- C0：命令網羅率(ステートメントカバレッジ):コード内の全てのステートメントを 少なくとも 1 回は実行
  1 ：input1 = 1、input2 == 1(任意)のテストデータが 1 件あれば、C0 カバレッジを満たすことになる。
  ※但し、上記、if 文に else case があり、且つ else case 中で処理がある場合(空処理除く)は if が偽になるようなテストデータ(任意)1 件も必要となる。

4．2．3　分岐条件が数値、単独、不等号条件の場合

・分岐条件が数値(Define 値)、単独、不等号条件の場合は、最大、最小、閾値、閾値±1、特異値の検証を行う。

```
Test(int a, int b, int c){
    if (a < 8){
        out = 1;          //期待値　1
    } else {
        out = 0;          //期待値　0
    }
}
```

例．分岐条件が数値、単独、不等号条件のソース

| 入力データ | | 期待値 |
|---|---|---|
| @a | @out | out |
| 0x7FFFFFFF | 10 | 0 |
| 0x80000000 | 10 | 1 |
| 9 | 10 | 0 |
| 8 | 10 | 0 |
| 7 | 10 | 1 |
| 0xFFFFFFFF | 10 | 1 |
| 0 | 10 | 1 |

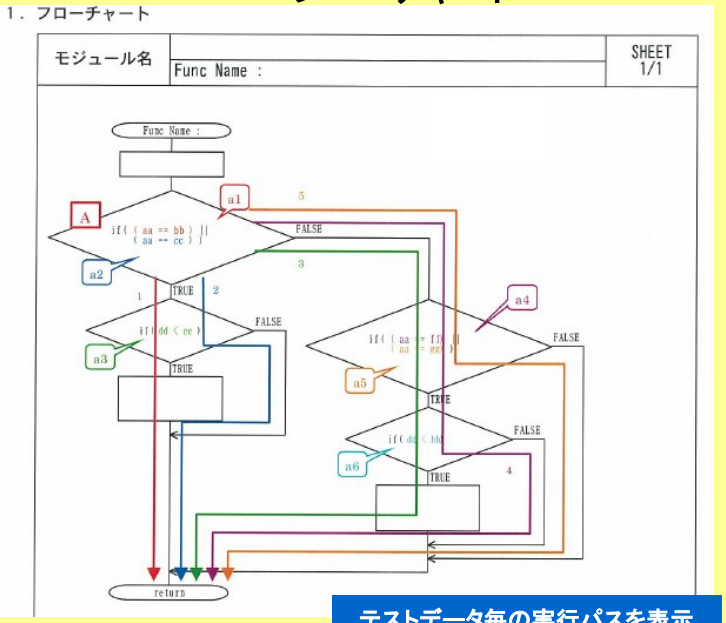
例．分岐条件が数値、単独、不等号条件の場合の入力データ

# 納品物：テスト仕様書とテストデータ

## ■ テスト指針とテストデータの意図を明確化

- カバレッジ指針（C0、C1、C2など）に従って、分岐に対するテストデータを作成
- フローチャート上で、どの分岐パスを通過するデータかを明確化
- 納品後のお客様のレビューを容易にするためのドキュメントを作成

**フローチャート**




テストデータ毎の実行パスを表示

最大値
最小値
閾値（±1）

| | 条件 | | | | | | | 経路 | 入力値 | | 期待値 | 備考 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | a1 | a2 | a3 | a4 | a5 | a6 | A | | @test1 | @test2 | test3@@ | |
| 1 | TRUE | FALSE | TRUE | | | | | 1 | 11 | 0x0E | 28 | MCDC観点 |
| 2 | TRUE | FALSE | FALSE | | | | | 2 | 70 | 0x0E | 70 | MCDC観点 |
| 3 | FALSE | TRUE | TRUE | | | | | 1 | 11 | 0x23 | 28 | MCDC観点 |
| 4 | FALSE | TRUE | FALSE | | | | | 2 | 70 | 0x23 | 70 | MCDC観点 |
| 5 | FALSE | FALSE | | TRUE | FALSE | TRUE | | 3 | 70 | 0x1F | 29 | MCDC観点 |
| 6 | FALSE | FALSE | | TRUE | FALSE | FALSE | | 4 | 200 | 0x1F | 200 | MCDC観点 |
| 7 | FALSE | FALSE | | FALSE | TRUE | TRUE | | 3 | 70 | 0x2C | 29 | MCDC観点 |
| 8 | FALSE | FALSE | | FALSE | TRUE | FALSE | | 4 | 200 | 0x2C | 200 | MCDC観点 |
| 9 | FALSE | FALSE | | FALSE | FALSE | | | 5 | 11 | 0x0D | 11 | MCDC観点 閾値-1 |
| 10 | TRUE | FALSE | TRUE | | | | | 1 | 39 | 0x0E | 0 | 閾値-1 |
| 11 | TRUE | FALSE | FALSE | | | | | 2 | 40 | 0x0E | 40 | 閾値 |
| 12 | TRUE | FALSE | FALSE | | | | | 2 | 41 | 0x0E | 41 | 閾値+1 |
| 13 | TRUE | FALSE | TRUE | | | | | 1 | 0 | 0x0E | 39 | 最小値 |
| 14 | TRUE | FALSE | TRUE | | | | | 1 | 1 | 0x0E | 38 | 最小値+1 |
| 15 | TRUE | FALSE | FALSE | | | | | 2 | 0xFE | 0x0E | 0xFE | 最大値-1 |

どの実行パスを通過するデータか表示

カバレッジ指針に従ったテストデータの意味を色で識別

# 納品物:テスト結果報告書

## ■ 単体テストの実行結果についてのドキュメント

### 単体テスト結果

| 項番 | 入力値 | | | | 出力値 | | 合否 | 処理時間 |
|---|---|---|---|---|---|---|---|---|
| | out | @a | @b | @c | out | test@@ | | |
| 1 | 10 | 0 | 0 | -2147483648 | 10 | 10 | OK | 0.078ms |
| 2 | 10 | 0 | 0 | 2147483647 | 10 | 10 | OK | 0.078ms |
| 3 | 10 | 0 | 0 | -1 | 10 | 10 | OK | 0.078ms |
| 4 | 10 | 0 | 1 | -2147483648 | 10 | 10 | OK | 0.079ms |
| 5 | 10 | 0 | 1 | 2147483647 | 10 | 10 | OK | 0.079ms |
| 6 | 10 | 0 | 1 | -1 | 10 | 10 | OK | 0.079ms |
| 7 | 10 | 0 | -2147483648 | -2147483648 | 10 | 10 | OK | 0.073ms |
| 8 | 10 | 0 | 2147483647 | -2147483648 | 10 | 10 | OK | 0.073ms |
| 9 | 10 | 0 | -1 | -2147483648 | 10 | 10 | OK | 0.073ms |
| 10 | 10 | 1 | -2147483648 | -2147483648 | 10 | 10 | OK | 0.074ms |
| 11 | 10 | 1 | 2147483647 | -2147483648 | 10 | 10 | OK | 0.074ms |
| 12 | 10 | 1 | -1 | -2147483648 | 10 | 10 | OK | 0.074ms |
| 13 | 10 | 0 | 1 | 1 | 4 | 4 | OK | 0.088ms |
| 14 | 10 | 0 | 1 | 0 | 3 | 3 | OK | 0.087ms |
| 15 | 10 | 0 | 1 | 2 | 10 | 10 | OK | 0.079ms |
| 16 | 10 | 0 | 0 | 1 | 2 | 2 | OK | 0.087ms |
| 17 | 10 | 0 | 0 | 0 | 1 | 1 | OK | 0.086ms |
| 18 | 10 | 0 | 0 | 2 | 10 | 10 | OK | 0.078ms |
| 19 | 10 | 0 | 2 | 1 | 10 | 10 | OK | 0.073ms |
| 20 | 10 | 0 | 2 | 0 | 10 | 10 | OK | 0.073ms |
| 21 | 10 | 0 | 2 | 2 | 10 | 10 | OK | 0.073ms |
| 22 | 10 | 1 | 1 | 1 | 8 | 8 | OK | 0.089ms |
| 23 | 10 | 1 | 1 | 0 | 7 | 7 | OK | 0.088ms |

実行結果の出力　&　期待値との合否判定

### カバレッジデータ

```
未実行  実行   C1      回数    SOURCE
                                 *  9通りのパスがあるプログラムです。
                                 * ++++++++++++++++++++++++++++++++++++ */
                                int test(int a , int b , int c)    /*(引き数 : a, b, c)*/
          ○             66      {

                                    int return_int;

          ○    T/F      66          if( a==0 ){
          ○    T/F      18              if( b==0 ){
          ○    T/F       6                  if( c==0 ){
          ○              1                      out=1;       /*1通り*/
                                                }
          ○    T/F       5                  else if( c==1 ){
          ○              1                      out=2;       /*2通り*/
                                                }
          ○              6              }
          ○    T/F      12              else if( b==1 ){
          ○    T/F       6                  if( c==0 ){
          ○              1                      out=3;       /*3通り*/
                                                }
          ○    T/F       5                  else if( c==1 ){
          ○              1                      out=4;       /*4通り*/
                                                }
                                        }
          ○             18          }
          ○    T/F      48          else if( a==1 ){
          ○    T/F      12              if( b==0 ){
          ○    T/F       3                  if( c==0 ){
          ○              1                      out=5;       /*5通り*/
                                                }
```

C0 / C1 カバレッジ結果のテキストレポート

# 納品物:障害報告書

## ■ 単体テストで障害が発生した関数のドキュメント

### 障害発生関数サマリ

障害発生関数リスト

| 関数名 | 網羅率 | 合否 |
|---|---|---|
| Test2 | 88% | NG |
| Test6 | 75% | NG |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

各関数毎の単体テスト結果を
一覧表でレポート

### 障害発生テスト結果報告書

テスト結果

| 入力データ | | 期待値 | 合否 |
|---|---|---|---|
| @a | @b | out2 | |
| 7 | 1 | 30?(10) | NG |
| 7 | 5 | 20 | OK |
| 7 | 0x7fffffff | 30 | OK |
| 7 | 0x80000000 | 30 | OK |
| 0x7fffffff | 0 | 40 | OK |

網羅率：88%

各関数毎の入出力結果(OK or NG)
期待値との照合結果レポート

**障害レポート例:**

2.　確認して頂きたい関数

2.1. GVPMain.c

2.1.1. GVPRectmr

1．静的変数 ga_rec_tmr がオーバーフローする可能性あり

該当テスト項目番号：　No. 34

関数 GVPRectmr にて、静的変数 ga_rec_tmr に格納されている値をチェックせず ga_rec_tmr をインクリメントしているため、オーバーフローする可能性があります。

オーバーフローする可能性があるのは、静的変数 ga_rec_tmr の領域が異常値に化けた場合になります。

# Q&A

# END

## ■ 最新情報はWEBサイトから　www.gaio.co.jp


ガイオ・テクノロジー株式会社

※会社名・商品名は各社の商標または登録商標です。
※本テキストの内容は、予告無く変更される場合があります。
※本書記載の誤りにより生ずる問題や損失に対して弊社は責任を負いません。
※本資料の無断転載、複写はお断りします。

ガイオ・テクノロジー株式会社
営業部
〒140-0002　東京都品川区東品川2-2-4
天王洲ファーストタワー25F
TEL.(03)4455-4767
Email　info@gaio.co.jp　・・ご質問はこちらまで