〇〇〇株式会社　御中

検証代行作業ガイドライン

版　数	日　付
初　版	２００８年９月４日

承認	ガイドライン作成担当者
技術 2008/09/04 桑田	技術 2008/09/04 水野

改定履歴

改定番号	改定日	改定部門	改定者	改定頁	改定内容
初　版	2008/9/4	営業部	水野祐樹	全頁	新規作成

＜目　次＞

１．はじめに

本書はガイオ・テクノロジー(株)が供給する「単体テスト代行サービス」のガイドラインを定義するものである。

２．目的

ガイドラインの内容は○○様向けに設定されており、弊社単体テストツールを用いた「単体テスト代行サービス」における、テスト基準及び、生成ドキュメントを明確にすることを目的とする。

３．検証環境

３．１　使用ツール

MPU【型番】	
クロスコンパイラ	
カバレッジマスター WinAMS	最新版
CasePlayer2	最新版
対象プロジェクト	－

３．２　基本設定

カバレッジマスター WinAMS	クロック数
CasePlayer2	

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。
- C1：分岐網羅率(ブランチカバレッジ)コード内の全てのブランチを少なくとも 1 回は実行
  １：input1 = 1、input2 == 1(任意)、２：input1 = 2(1 以外)、input2 == 1(2 以外)のテストデータが 2 件あれば、C1 カバレッジを満たすことになる。
  ※C0 との違いは、else case があろうとなかろうと、真偽の 2 ケースをテストケースとする。
- C2：条件網羅率(コンディションカバレッジ):コード内のすべての条件を少なくと も 1 回は実行
  １：input1 = 1、input2 == 2、２：input1 = 1、input2 == 1(2 以外)、３：input1 = 2(1 以外)、input2 == 2、
  ４：input1 = 2(1 以外)、input 2== 1(2 以外)のテストデータが 4 件あれば、C2 カバレッジを満たすことになる。条件文中の or 条件や and 条件等とは無関係に全ての変数に対しての真偽の総組み合わせで試験データを作成する。１条件文のテストデータ件数は「2^(変数の数量)」で算出。
  ※C1 との違いは、全ての変数に対しての真偽の組み合わせを確認。

- MC/DC：　Modified Condition/Decision
  １：input1 = 1、input2 == 1(2 以外)、２：input1 = 2(1 以外)、input2 == 2、３：input1 = 2(1 以外)、input 2== 1(2 以外)のテストデータが 3 件あれば、MC/DC カバレッジを満たすことになる。
  ※C2 との違いは、着目する変数以外の他変数の真偽を固定して、自分の変数の値を変えることにより、
  式全体の真偽が変わる入力データのみをテストデータとする事によって、テストパターン数が減る事である。

４．２　テストデータの作成

４．２．１　テストデータ作成基準

・テストデータ作成の考え方

データを限定する手法を用いる。

ー境界値テスト

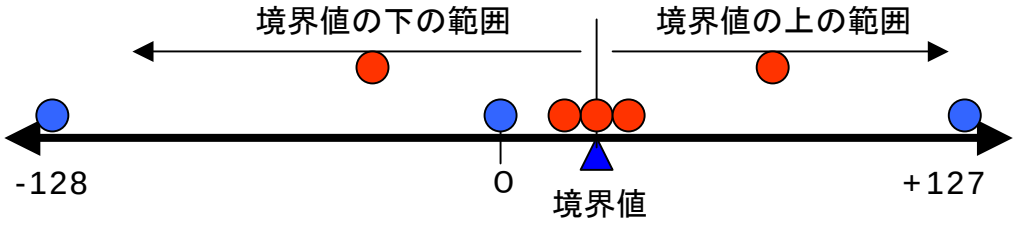
・境界値を境にして有効な範囲を決めて代表値を与える

-境界値そのもの、境界値±１、

-境界値の上/下範囲で任意の１つの値

ー特異点テスト

・変数値の０、－１、最大値、最小値



・テストデータ作成する上での基準を以下に示す。

■基準値で使用する変数は全て signed である前提で作成するので、unsigned の場合は、テストケースが減少する。

■基準は入力値をそのまま使用している条件文及び入力値をローカル変数に代入して使用している条件文（演算処理している場合は除く）、関数の戻り値を使用している条件文（スタブ化した場合）に適用する。但し、試験データが非常に複雑になるケースは、都度お客様へ確認し、真偽２パターンのみの検証とする。

■関数仕様書に記載されている制限事項を優先させる。

例）関数仕様書に「入力変数 Byte fc_idx(1～80)」と記述されている場合、変数最大値 255→80
最小値 0→1 に入力値を変更する。（制限外の場合、異常終了となるため）

■基本入力値

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。基本は正数(11)とする。

初期値：全ての期待値(出力値)に入力データとして与える値。

> 初期値設定の方針は以下とする。
> ①出力のみの変数(符号なし char 型 fc_xxx)の期待値が 0,1,2,3 の場合、初期値を 10 に設定すれば重ならない。そのため入力設定を除外し初期値設定のみにする。
>
> ②出力のみの変数(符号なし char 型 fc_xxx)の期待値が 9,10,11 の場合、初期値 10 が重なる場合がある。そのケースは重ならない値を付与し変更されたことが確認できる値とする。
>
> ③出力のみの変数(符号なし char 型 fc_xxx)の期待値が 0～255 の場合、初期値を 1 つに固定してしまうと必ずどれかのケースでは期待値と初期値が重なってしまうので、この場合は入力設定し期待値と重ならないような値を初期値とする。

特異値：該当する変数の signed、unsigned を考慮して、-1,0 を特異値とする。

４．２．２　分岐条件が数値、単独、イコール条件の場合

・分岐条件が数値(Define 値)、単独、イコール(Not イコール)条件の場合は、閾値、閾値±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、イコール条件のソース

入力データ		期待値
@a	@out	out
0	10	0
1	10	1
2	10	0

例．分岐条件が数値、単独、イコール条件の場合の入力データ

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

４．２．４　分岐条件が数値、複数、イコールの場合

・分岐条件が数値(Define 値)、複数、イコール(Not イコール)の場合は、AND、OR に関わらず下記の通り検証を行う。

※全ての条件の True、False を考慮し、最低でも条件分の True、False が 1 度は実行されるようなテストデータを作成する。

```
Test(int a, int b, int c){
    if(a == 1 || b==3 && c==2) {
        out = 1;          //期待値　1
    } else {
        out = 0;          //期待値　0
    }
}
```

例．分岐条件が数値、複数、イコールのソース

入力データ				期待値
@a	@b	@c	out	out
1	4	2	10	1
2	3	2	10	1
2	3	3	10	0
2	4	2	10	0
0	2	1	10	0

True, False,True(True から始まる混在)
False,True,True (False から始まる混在)
False,True,False(False から始まる混在)
False, False, True(False から始まる混在)
False, False, False(False のみ)

例．分岐条件が数値、複数、イコールの入力データ

４．２．５　分岐条件が数値、複数、不等号の場合

・分岐条件が(Define 値)、複数、不等号の場合は、AND、OR に関わらず下記の通り検証を行う。

※等号については全ての条件の True、False を考慮し、不等号については「７．１．２　分岐条件が数値、
単独、不等号条件の場合」の値を考慮し、最低でも条件分の True、False が 1 度は実行されるようなテストデータを作成する。(全ての条件が True、False、混在のケース数パターンで作成)

```
Test(int a, int b, int c){
    if(a < 10 || b==3 && c <= 20) {
        out = 1;          //期待値　1
    } else {
        out = 0;          //期待値　0
    }
}
```

例．分岐条件が数値、複数、不等号のソース

入力データ				期待値
@a	@b	@c	out	out
9	3	19	10	1
0	2	0	10	1
11	3	21	10	0
0x80000000	3	0x80000000	10	1
10	4	20	10	0
0xFFFFFFFF	2	0xFFFFFFFF	10	1
0x7FFFFFFF	4	0x7FFFFFFF	10	0

True,True,True(True のみ)
True,False,True(True から始まる混在)
False,True,False(False から始まる混在)
True,True,True(残ったデータで任意)
False,False,True(残ったデータで任意)
True,False,True(残ったデータで任意)
False, False, False(False のみ)

例．分岐条件が数値、複数、不等号の入力データ

４．２．６　分岐条件が変数、単独、イコールの場合

・分岐条件が変数、単独、イコール(Not イコール)の場合は左辺、右辺に上下限値をそれぞれ入力し、TRUE と FALSE の検証を行う。また、一方のデータを固定(任意)して、閾値、閾値±１の検証も行う。

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

例．分岐条件が変数、単独、イコールのソース

入力データ			期待値
@a	@b	out	out
0x7FFFFFFF	0x7FFFFFFF	10	1
0x80000000	0x80000000	10	1
0x7FFFFFFF	0x80000000	10	0
10	11	10	0
11	11	10	1
12	11	10	0

例．分岐条件が変数、単独、イコールの入力データ

４．２．７　分岐条件が演算結果、単独、イコールの場合

・分岐条件が演算結果、単独、イコール(Not イコール)の場合は下記の検証を行う。

```
Test(int a){
    if ((a * 3) == 18) {
        out = 1;          //期待値　1
    } else {
        out = 0;          //期待値　0
    }
}
```

例．分岐条件が演算結果、単独、イコールのソース

入力データ		期待値
@a	out	out
3	10	0
18	10	0
6	10	1

例．分岐条件が演算結果、単独、イコールの入力データ

※上記の例だと演算する定数と判定する値自体を入力値とするだけで良いが、それだと TRUE パターンが検証できないので、6 も入力データとする。

※テストケースが 1000 以上になる場合は TRUE,FALSE の 2 パターンのみで試験する。
したがって、このケースでは３を入力データから外す。

４．２．８　分岐条件が演算結果、単独、不等号の場合

・分岐条件が演算結果、単独、不等号の場合は下記の検証を行う。

```
Test(int a){
    if ((a * 3) >18) {
        out = 1;          //期待値　1
    } else {
        out = 0;          //期待値　0
    }
}
```

例．分岐条件が演算結果、単独、不等号のソース

入力データ		期待値
@a	out	out
5	10	0
6	10	0
7	10	1
0x80000000	10	0
0x7FFFFFFF	10	1

例．分岐条件が演算結果、単独、不等号の入力データ

※境界値の 5, 7 と最小・最大値を入力データとして設定する。

４．２．９　分岐条件が変数、単独、不等号の場合

・分岐条件が変数、単独、不等号の場合は左辺、右辺に最大・最小値、-1 をそれぞれ入力し、TRUE と FALSE の検証を行う。また、一方のデータを固定(任意)して、閾値、閾値±１の検証も行う。

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

例．岐条件が変数、単独、不等号のソース

入力データ			期待値
@a	@b	out	out
0x7FFFFFFF	0x7FFFFFFF	10	0
0x80000000	0x80000000	10	0
0x7FFFFFFF	0x80000000	10	1
0x7FFFFFFF	0xFFFFFFFF	10	1
0x80000000	0xFFFFFFFF	10	0
0xFFFFFFFF	0x7FFFFFFF	10	0
0xFFFFFFFF	0x80000000	10	1
10	11	10	0
11	11	10	0
12	11	10	1

例．分岐条件が変数、単独、不等号の入力データ

４．２．１０　分岐条件が変数、複数、イコールの場合

・分岐条件が変数、複数、イコールの場合は、AND、OR に関わらず下記の通り検証を行う。

※全ての変数の True、False を考慮し、最低でも条件分の True、False が 1 度は実行されるようなテストデータを作成する。（全ての条件が True、False、混在のケース 2 パターンで作成）

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

例．分岐条件が変数、複数、イコールのソース

入力データ					期待値	
@a	@d	@b	@e	out	out	
0x7FFFFFFF	0x7FFFFFFF	0x7FFFFFFF	0x7FFFFFFF	10	1	True,True(True のみ)
11	11	12	11	10	1	True,False(True から始まる混在)
10	11	11	11	10	1	False,True(False から始まる混在)
12	11	10	11	10	0	False, False (残ったデータで任意)
0x7FFFFFFF	0x80000000	0x80000000	0x7FFFFFFF	10	0	False, False(False のみ)

例．分岐条件が変数、複数、イコールの入力データ

４．２．１１　分岐条件が変数、複数、不等号の場合

・分岐条件が変数、複数、不等号の場合は、AND、OR に関わらず左辺、右辺に上限値、-1 をそれぞれ入力し下記の通り検証を行う。

※等号については「７．１．５　分岐条件が変数、単独、イコールの場合」の値を考慮し、不等号については「７．１．６　分岐条件が変数、単独、不等号の場合」の値を考慮し、最低でも条件分の True、False が 1 度は実行されるようなテストデータを作成する。

（全ての条件が True、False、混在のケース数パターンで作成）

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

例．分岐条件が変数、複数、不等号のソース

入力データ					期待値
@a	@d	@b	@e	out	out
0x80000000	0xFFFFFFFF	0x7FFFFFFF	0x7FFFFFFF	10	1
0xFFFFFFFF	0x80000000	0x7FFFFFFF	0x80000000	10	1
0x7FFFFFFF	0x80000000	0x80000000	0x80000000	10	1
0x7FFFFFFF	0xFFFFFFFF	0x7FFFFFFF	0x7FFFFFFF	10	1
0x7FFFFFFF	0x7FFFFFFF	0x80000000	0x80000000	10	1
0x80000000	0x7FFFFFFF	0x7FFFFFFF	0x7FFFFFFF	10	1
10	11	10	11	10	1
11	11	11	11	10	1
12	11	12	11	10	0
0xFFFFFFFF	0x80000000	0x7FFFFFFF	0x80000000	10	0

True,True(True のみ)
True,False(True から始まる混在)
False,True(False から始まる混在)
False,True(残ったデータで任意)
False,True (残ったデータで任意)
True,True(残ったデータで任意)
True,False(残ったデータで任意)
False,True (残ったデータで任意)
False,False (残ったデータで任意)
False,False(False のみ)

例．分岐条件が変数、複数、不等号の入力データ

４．２．１２　Switch 分岐の場合

・Switch 分岐は全ての case の条件 default での検証を行う。

※default 判定の為に最小値、最大値での検証を行う。（case に最小値と最大値が含まれる場合は①初期値、②case に含まれない最小値の候補で入力データとする。）

```
int Test(int input1) {
    Switch(input1){
        Case 0:
            out = 0;            //期待値　0
            break;
        Case 1:
            out = 1;            //期待値　2(= 1 + 1)
        Case 2:
            out = out + 1;      //期待値　11(= 10 + 1)
            break;
        Default:
    }
}
```

例．Switch 分岐の場合のソース

入力データ		期待値
@input1	out	out
0	10	0
1	10	1
2	10	11
0x80000000	10	10
0x7FFFFFFF	10	10

例．Switch 分岐の場合の入力データ

４．２．１３　関数の戻り値が分岐条件の場合

・子関数の戻り値はスタブ関数を作成し、入力データとする。
（分岐かどうかに関わらず入力を設定する）
※スタブ関数の戻り値をどのように使用しているかによって、前述の基準に合わせてスタブ関数の入力値を決定する。（例は「７．１．２　分岐条件が数値、単独、不等号条件の場合」）

```
Test(int b) {
    a = func(b);            //関数 func の戻り値が分岐条件
    if(a > 10) {
        out = 0;            //期待値  0
    } else {
        out = 1;            //期待値  1
    }
}
```

例．関数の戻り値が分岐条件のソース

```
int AMSTB_func(int b) {   //スタブ関数          ※スタブで使用する入力値は
    static int AMSTB_func;                        ret_+スタブ関数名とする。
    return(ret_ AMSTB_func);     //戻り値
}
```

例．関数の戻り値が分岐条件の場合のスタブ関数

入力データ		期待値
ret_ AMSTB_func	out	out
0x80000000	10	1
0x7FFFFFFF	10	0
11	10	0
10	10	1
9	10	1
0	10	1
0xFFFFFFFF	10	1

例．関数の戻り値が分岐条件の場合の入力データ

４．２．１４　演算部分の検証について

・演算については最大値、最小値、特異値、一般値にて検証を行う。

※演算には代入は含みません。

```
int out;
int a,b;
void Test( ) {
      if (a == 1) {
            out=1;
      } else {
            out=b+1;
      }
}
```

例．演算部分の検証を行う場合のソース

入力データ			期待値
@a	@b	out	out
1	11	10	1
11	11	10	12
11	0x80000000	10	0x80000001
11	0x7FFFFFFF	10	0x80000000
11	0xFFFFFFFF	10	0x00000000
11	0	10	1

例．演算部分の検証を行う場合のテストデータ

４．２．１５　union データの場合

・union（共用体）データが入出力になっている場合は、ソース内で注目している箇所のみを扱う。

※但し、共用体を bit 情報として扱う場合は、ソース内で使用していなくても union の全ての変数を扱う。

```
typedef union _uniontest
      struct {
            int       x;
            int       y;
            int       z;
      } d1;
      int  d2[3];
} uniontest;
uniontest   u_data;

void Test( ) {
   if (u_data.d1.x == 1) {
      u_data.d2[2] = 99
   }
}
```

例．union データの検証を行う場合のソース

入力データ		期待値
u_data.d1.x	u_data.d2[2]	u_data.d2[2]
1	10	99
11	10	10

例．union データの検証を行う場合のテストデータ

※出力値として u_data.d2[2] があるので、テストデータとして扱うが、該当箇所が u_data.d1.z に代入するソースならば、u_data.d2 はテストデータとして扱わない。

```
typedef union _uniontest {
    struct{
        unsigned char   bit0    :1;
        unsigned char   bit1    :1;
        unsigned char   bit2    :1;
        unsigned char   bit3    :1;
        unsigned char   bit4    :1;
        unsigned char   bit5    :1;
        unsigned char   bit6    :1;
        unsigned char   bit7    :1;
    }bit;
    unsigned char byte;
} uniontest;
uniontest   u_data;

void Test( ) {
  if (u_data.bit.bit0 == 1) {
    u_data.bit1 = 1;
  }
}
```

例．union データ(bit 情報)の検証を行う場合のソース

入力データ		期待値	
u_data.byte	u_data.bit.bit0	u_data.byte	u_data.bit.bit1
0	0	0	0
2	1	3	1

例．union データ(bit 情報)の検証を行う場合のテストデータ

※入出力値として u_data.byte はないが、他の bit 状況によって、u_data.byte の値が変わる為、
u_data.byte をテストデータとして扱う。

４．２．１６　無限ループの場合

・無限ループの条件が定数の場合は、スタートアップコマンドにて該当処理を飛ばして試験を実施する。
※カバレッジが 100%にならないので、その旨をテスト結果報告書に明記する。

```
void Test( ) {
    out = 0;
    while (1) {
    }
    out = 99;
}
```

例．無限ループの場合のソース
※サンプルソース赤字部分をマクロによって処理させないようにして試験を実施する。

・無限ループの条件が変数の場合は、テストデータは無限ループしないテストデータのみ使用する。
※カバレッジが 100%にならないので、その旨をテスト結果報告書に明記する。

```
void Test( ) {
    out = 0;
    while (glabal == 1) {
        out = 99;
    }
}
```

例．無限ループの場合のソース
※サンプルソース赤字部分の global の入力値は 0(1 以外)のみを入力値とする。

４．２．１７　「単体テスト仕様書」における入力値の色分け及び特記事項

- 色分け対象は入力値(条件文)のみとする。
  ※初期化をする為に入力としている出力値は除く。
  ※スタブ関数の入力値として使用している入力値は除く。
  ※変数が入出力の両方に該当する場合は、色分け対象とする。
- 最小値はセルを黄色で表示する。
- 最大値はセルを赤色で表示する。
- 閾値、閾値＋１、閾値－１はセルを緑で表示する。
- 閾値、閾値＋１、閾値－１が最小、最大と同一値の場合は、最小値または最大値の色を使用する。

入力データ		期待値
@a	out	out
0x80000000	10	1
0x7FFFFFFF	10	0
11	10	0
10	10	1
9	10	1
0	10	1
0xFFFFFFFF	10	1

例．テストデータの色分け

４．３　スタブ作成基準

スタブ要否判断

・原則、呼び出されるサブルーチン・システムコールはすべてスタブ化する。

※スタブ関数の仕様を**テスト仕様書に記載する**

(スタブソースの保存場所及びスタブ化した関数のソース名、関数名)

・例外等は、特記事項欄へ記載する。

・C 言語のスタブについて

C 言語の場合、以下のサンプルに沿ってスタブを作成する。

スタブサンプル例

```
#define D_CALLMAX_Exsample 8 //8 は呼び出し回数
UI08 AMSTB_Exsample(UI08 ac_hoge_idx, UI08 ac_hoge_sta  )
{
            static UI08 AMSTB_ac_hoge_idx[D_CALLMAX_Exsample];
            static UI08 AMSTB_ac_hoge_sta [D_CALLMAX_Exsample];
            static UI08 ret[D_CALLMAX_Exsample];
            static int cnt;

            AMSTB_ac_hoge_idx[cnt] = ac_hoge_idx;
            AMSTB_ac_hoge_sta [cnt] = ac_hoge_sta  ;

            cnt++;

            return ret[cnt - 1];
}
```

４．４　出力値の期待値判定

期待値と出力値の合否判定を実施

※期待値の判定を実施する場合は、どのような試験を行うか、下記にその特記事項を記載する。

特記事項について明確にする為、該当する文字（有/無）を赤色とする。

特記事項	有	特記内容	仕様書から求めらる期待値を正としてテストを実施する。

５．納品物

検証代行作業で使用した全ての環境（ソース、CasePlayer2 環境、winAMS 環境）のデータを示す。

納品物	備考
ツール環境一式	
ガイドライン(Word 形式)	
テスト仕様書(Word 形式)	
テストデータファイル(Excel 形式)	
テスト結果報告書(Word 形式)	
単体テスト結果ファイル(Excel 形式)	
カバレッジデータファイル(テキスト形式)	
作業レポート(Word 形式)	

５．１　納品物のフォルダ構成

納品テスト環境フォルダの構成は以下とする。

フォルダ階層	説明
１．¥home¥(プロジェクト名)¥source¥	お客様より頂いたソースファイル一式
２．¥home¥(プロジェクト名)¥source¥obj¥	上記をコンパイルしたオブジェクト
３．¥home¥(プロジェクト名)¥winAMS¥ 対象プロジェクト名¥obj¥	関数単位で単体試験を行う為、該当試験で使用するオブジェクトを保存する。 ※OMF コンバートしたファイルも保存する。 ※スタブ関数を使用しない限りは、上記２．のオブジェクトをコピーする。 ※スタブ関数を使用する場合は、スタブファイルも保存する。
４．¥home¥(プロジェクト名)¥winAMS¥	テストプロジェクト環境
５．¥home¥(プロジェクト名)¥CasePlayer2¥	ケースプレーヤー環境

５．２　テストデータ入力ファイルの作成

入力ファイルのスケルトンは winAMS で作成する。生成したスケルトンに対してデータ作成基準に沿ったデータと（期待値）を入力する。

６．テスト結果について

以下の条件を満たした場合はNGと判断し、障害レポートへ詳細を記載する。

- ■ 期待値判定がある場合にNG判定が存在する。
- ■ C0若しくは、C1カバレッジが100%未満となっている。
  - ※ただし、条件文の設定によって常に真もしくは偽となり実行されないロジックが存在する場合は、この限りではない。（結果報告書の特記事項として記載し、且つ障害レポートを作成する。）
  - ※コンパイルの展開や最適化によるwinAMSの仕様によって、１００％にならない場合は、アセンブラコードを確認して、問題なければこの限りではない。（結果報告書の特記事項として記載し、障害レポートは作成しない。）
- ■ お客様からの借用した仕様書とソースとの矛盾が存在する。
- ■ 結果判定はOKだが、修正したほうがよいと思われる個所が存在する。

カバレッジの確認

- ■ 全ての実行行が「○」となっていること
- ■ if文の実行真偽欄が「T/F」となってること
- ■ Switch文のケース実行欄が条件数「n/n」となっていること

※本検証は対象となるソフトウェアの動作を保証するものではない。
関数の単体テストに特化したものであり、テスト結果を元にお客様側で正誤の判定を行うものとする。
また、検証対象となるソフトウェアを用いて、直接的または間接的に生じたいかなる損害に関しても、弊社はその責任を負わないものとする。

７．ツールの仕様について

winAMSのカバレッジはオブジェクトレベルで実行した結果を、Cソースコードベースに置き換えて表示している為、以下の現象が発生する事がある。

- ・ 関数の入り口と出口で実行回数に相違が発生する
- ・ カバレッジログファイルで、関数全てが表示されない
- ・ テストデータ的に100%網羅できているが、表示カバレッジが100%にならない

実際はカバレッジを100%網羅しているが、結果に手を加えることは自動テストの意味をなさなくなる為、そのままの状態で報告する。
このようなケースが発生した場合は、結果報告書に正しくテストが実施されているという事を記載し証明する。

※お客様の要望によりソースの修正や修正後の再試験をする場合は、下記にその特記事項を記載する。
特記事項について明確にする為、該当する文字（有/無）を赤色とする。

特記事項	有 / 無	特記内容	