比較関数の罠

この記事は、インテル® デベロッパー・ゾーンに公開されている「The Evil within the Comparison Functions」の日本語参考訳です。

この記事の PDF 版はこちらからご利用になれます。

私は以前、プログラマーはテキストブロックの最後の行でミスをすることが多いのに気付き、「最後の行問題」という記事で紹介しました。この記事では、私が新しく発見した傾向をお知らせします。その傾向とは、プログラマーは 2 つのオブジェクトを比較する関数でミスをするというものです。俄かには信じられないかもしれませんが、読者の皆さんが驚きを感じると思われる多くのエラーの例を紹介します。これは新しい試みであり、非常に興味深く、そして恐るべきものです。

問題点

見解: プログラマーは 2 つのオブジェクトを比較する単純な関数でミスをすることが多い。

この見解は、C、C++ および C# で記述された多くのオープンソース・プロジェクトを調査したチームの経験に基づいたものです。

ここで示す例は、*IsEqual*、*Equals*、*Compare*、*AreEqual* などの関数や、*==*、*!=* のようなオーバーロードされた演算子です。

私は、「多様なエラーの記事」(英語) を書いているときに、比較関数に関連するエラーが多いことに気付きました。そこで、これを調べることを決め、これまで発見したエラーの「データベース」(英語) を細部まで確認しました。具体的には、*Cmp*、*Equal*、*Compare* などの単語を含む関数の検索を行いました。

結果は印象的かつ衝撃的なもので、「最後の行問題」を書いたときに感じたことに似ていました。私は同様の例外に気付き、それらをさらに詳しく調べることにしました。残念ながら、以前の記事とは異なり、この記事に統計や図を追加する方法が明確でないため、統計を利用したソリューションは、別の機会に紹介することにします。現時点では、直感に基づいて得られた考えを述べるに止めます。比較関数には多くのミスが含まれています。多くの印象的な例を見た後、皆さんも同じ思いを共有されると確信しています。

心理状態

ここで、「最後の行問題」を振り返ってみましょう。この記事をまだお読みでない方は、この機会に目を通してみてください。この問題の詳細は、「The last line effect explained」(英語) で述べています。

一般に、最後の行におけるエラーの原因は、開発者はコードを書き終える前に新しい行やタスクのことを考えている、という事実に関連しています。そのため、同様のテキストのブロックを記述するときに、プログラマーは最後の行でミスをする確率が高いことが分かりました。

私は比較関数を記述する場合も同じだと考えています。一般に、開発者はこれらの作業を大した作業ではないと考えがちです。つまり、深く考えることなく、無意識にコードを記述してしまうのです。そうでなければ、プログラマーがこのような過ちを犯す説明がつきません。

次のエラーは **RunAsAdmin Explorer Shim** プロジェクトのコード (C++) で見つかりました。

```
bool IsLuidsEqual(LUID luid1, LUID luid2){
  return (luid1.LowPart == luid2.LowPart) &&
         (luid2.HighPart == luid2.HighPart);
}
```

PVS-Studio 警告: V501 (英語) There are identical sub-expressions to the left and to the right of the '==' operator: luid2.HighPart == luid2.HighPart RAACommon raacommonfuncs.cpp 1511 (V501 '==' 演算子の左辺と右辺に同じ部分式があります。luid2.HighPart == luid2.HighPart RAACommon raacommonfuncs.cpp 1511)

これは単純な入力ミスです。2 行目は *luid1.HighPart == luid2.HighPart* が正解です。

コードは非常に単純です。どうやら、コードの単純さがすべての原因かと思われます。プログラマーは、典型的で退屈な関数を定義するタスクを考えます。この関数を定義する方法を考えて、コードを実装します。これがプログラマーの日常ですが、必然的に、さらに重要で、複雑で、興味深いコードを書くというタスクもあます。この新しいタスクに集中しすぎると、過ちを犯すことになります。

また、プログラマーがこれらの関数の単体テストについて深く考えることはまずありません。ここでも、関数の単純さが邪魔をしています。これらの関数は単純で、単に繰り返し使用されるため、テストは必要ないと考えてしまうのです。そのような関数を多く開発したプログラマーは、別の関数でも同じミスをするでしょうか。もちろん、まったく同じ過ちを犯すこともあるでしょう。

念のために述べておきますが、この記事で説明することは、プログラミングを始めたばかりの初心者が記述したコードだけではありません。GCC、Qt、GDB、LibreOffice、Unreal* Engine、CryEngine* V、Chromium*、MongoDB*、Oracle* VM Virtual Box、FreeBSD*、WinMerge、the CoreCLR、MySQL*、Mono*、CoreFX*、Roslyn、MSBuild のような、本格的なプロジェクトのコードにありがちなミスについて話しています。

では、多くのさまざまなエラーの例を具体的に紹介していきましょう。

比較関数のエラーのパターン

比較関数のすべてのエラーは、いくつかのパターンに区分できます。この記事では、C、C++ および C# プロジェクトのエラーについて紹介しますが、ほとんどのパターンはほかの言語でも類似しているため、言語を区別することは無意味でしょう。

パターン: A < B、B > A

比較関数では、多くの場合、次のようなチェックを行います。

- A < B
- A > B

同じ演算子 < を使用するのではなく、次のように変数を入れ替えてチェックする方法を好むプログラマーもいます。

- A < B
- B < A

しかし、不注意により、次のようなチェックが行われることがあります。

- A < B
- B > A

つまり、同じ比較が 2 回行われます。これだけでは問題点は不明ですが、これから紹介する具体的な例を見れば、問題点は明らかでしょう。

次のエラーは **MongoDB*** プロジェクトのコード (C++) で見つかりました。

```
string _server;
....
bool operator<( const ServerAndQuery& other ) const {
  if ( ! _orderObject.isEmpty() )
    return _orderObject.woCompare( other._orderObject ) < 0;

  if ( _server < other._server )
    return true;
  if ( other._server > _server )
    return false;
  return _extra.woCompare( other._extra ) < 0;
}
```

PVS-Studio 警告: V581 (英語) The conditional expressions of the 'if' operators situated alongside each other are identical. Check lines: 44, 46. parallel.h 46 (V581 両側に位置している 'if' 演算子の条件式が同じです。行 44、46 を確認してください。parallel.h 46)

次の条件

```
if ( other._server > _server )
```

は 2 つ前の行で行われたチェックと同じで、常に偽になります。正しいコードは次のとおりです。

```
if ( _server < other._server )
  return true;
if ( other._server < _server )
  return false;
```

次のエラーは **Chromium*** プロジェクトのコード (C++) で見つかりました。

```
enum ContentSettingsType;
struct EntryMapKey {
```

```
  ContentSettingsType content_type;
  ...
};

bool OriginIdentifierValueMap::EntryMapKey::operator<(
    const OriginIdentifierValueMap::EntryMapKey& other) const {
  if (content_type < other.content_type)
    return true;
  else if (other.content_type > content_type)
    return false;
  return (resource_identifier < other.resource_identifier);
}
```

PVS-Studio 警告: V517 (英語) The use of 'if (A) {...} else if (A) {...}' pattern was detected. There is a probability of logical error presence. Check lines: 61, 63. browser content_settings_origin_identifier_value_map.cc 61 (V517 'if (A) {...} else if (A) {...}' パターンの使用を検出しました。論理的なエラーの可能性があります。行 61、63 を確認してください。browser content_settings_origin_identifier_value_map.cc 61)

C++ だけでなく、C# の例も見てみましょう。次のエラーは **IronPython and IronRuby** プロジェクトのコード (C#) で見つかりました。

```
public static int Compare(SourceLocation left,
                          SourceLocation right) {
  if (left < right) return -1;
  if (right > left) return 1;
  return 0;
}
```

PVS-Studio 警告: V3021 (英語) There are two 'if' statements with identical conditional expressions. The first 'if' statement contains method return. This means that the second 'if' statement is senseless. SourceLocation.cs 156 (V3021 同じの条件式の 2 つの 'if' 文があります。1 つ目の 'if' 文にメソッド return が含まれています。このため 2 つ目の 'if' 文は無意味です。SourceLocation.cs 156)

もはや説明の必要はないでしょう。

注: C# のエラーの例を 1 つ、C++ のエラーの例を 2 つ示しました。一般に、C# コードのほうが C/C++ コードよりもバグの数は少なめです。しかし、C# はより安全であると結論を急がないでください。要するに、PVS-Studio が C# コードをチェックするようになったのは比較的最近のことであり、C や C++ で書かれたプロジェクトよりも C# で書かれたプロジェクトのほうがチェックの数が少ないのです。

パターン: クラスのメンバーと自身との比較

比較関数は通常、構造体/クラスメンバーを続けて比較します。クラスのメンバーを自身と比較すると、コードでエラーが発生しやすくなります。2 種類のエラーを次に示します。

1 つ目は、プログラマーがオブジェクトの名前を指定するのを忘れるようなケースです。

```
return m_x == foo.m_x &&
       m_y == m_y &&            // <=
       m_z == foo.m_z;
```

2 つ目は、同じ名前のオブジェクトを書くケースです。

```
return zzz.m_x == foo.m_x &&
       zzz.m_y == zzz.m_y &&    // <=
       zzz.m_z == foo.m_z;
```

このパターンが当てはまる実際の例を詳しく見てみましょう。不正な比較は類似コードブロックの最後のブロックでよく行われることに注意してください。ここでも、「最後の行問題」があります。

次のエラーは **Unreal* Engine 4** プロジェクトのコード (C++) で見つかりました。

```
bool
Compare(const FPooledRenderTargetDesc& rhs, bool bExact) const
{
  ....
  return Extent == rhs.Extent
    && Depth == rhs.Depth
    && bIsArray == rhs.bIsArray
    && ArraySize == rhs.ArraySize
    && NumMips == rhs.NumMips
    && NumSamples == rhs.NumSamples
    && Format == rhs.Format
    && LhsFlags == RhsFlags
    && TargetableFlags == rhs.TargetableFlags
    && bForceSeparateTargetAndShaderResource ==
         rhs.bForceSeparateTargetAndShaderResource
    && ClearValue == rhs.ClearValue
    && AutoWritable == AutoWritable;           // <=
}
```

PVS-Studio 警告: V501 (英語) There are identical sub-expressions to the left and to the right of the '==' operator: AutoWritable == AutoWritable rendererinterface.h 180 (V501 '==' 演算子の左辺と右辺に同じ部分式があります。AutoWritable == AutoWritable rendererinterface.h 180)

Samba プロジェクトのコード (C)。

```
static int compare_procids(const void *p1, const void *p2)
{
  const struct server_id *i1 = (struct server_id *)p1;
  const struct server_id *i2 = (struct server_id *)p2;

  if (i1->pid < i2->pid) return -1;
  if (i2->pid > i2->pid) return 1;
  return 0;
```

```
}
```

PVS-Studio 警告: V501 (英語) There are identical sub-expressions to the left and to the right of the '>' operator: i2->pid > i2->pid brlock.c 1901 (V501 '>' 演算子の左辺と右辺に同じ部分式があります。i2->pid > i2->pid brlock.c 1901)

MongoDB* プロジェクトのコード (C++)。

```
bool operator==(const MemberCfg& r) const {
  ....
  return _id==r._id && votes == r.votes &&
         h == r.h && priority == r.priority &&
         arbiterOnly == r.arbiterOnly &&
         slaveDelay == r.slaveDelay &&
         hidden == r.hidden &&
         buildIndexes == buildIndexes;        // <=
}
```

PVS-Studio 警告: V501 (英語) There are identical sub-expressions to the left and to the right of the '==' operator: buildIndexes == buildIndexes rs_config.h 101 (V501 '==' 演算子の左辺と右辺に同じ部分式があります。buildIndexes == buildIndexes rs_config.h 101)

Geant4 Software プロジェクトのコード (C++)。

```
inline G4bool G4FermiIntegerPartition::
operator==(const G4FermiIntegerPartition& right)
{
  return (total == right.total &&
          enableNull == enableNull &&          // <=
          partition == right.partition);
}
```

PVS-Studio 警告: V501 (英語) There are identical sub-expressions to the left and to the right of the '==' operator: enableNull == enableNull G4hadronic_deex_fermi_breakup g4fermiintegerpartition.icc 58 (V501 '==' 演算子の左辺と右辺に同じ部分式があります。enableNull == enableNull G4hadronic_deex_fermi_breakup g4fermiintegerpartition.icc 58)

LibreOffice プロジェクトのコード (C++)。

```
class SvgGradientEntry
{
  ....
  bool operator==(const SvgGradientEntry& rCompare) const
  {
    return (getOffset() == rCompare.getOffset()
           && getColor() == getColor()            // <=
           && getOpacity() == getOpacity());      // <=
  }
```

```
  ....
}
```

PVS-Studio 警告: V501 (英語) There are identical sub-expressions to the left and to the right of the '==' operator: getColor() == getColor() svggradientprimitive2d.hxx 61 (V501 '==' 演算子の左辺と右辺に同じ部分式があります。getColor() == getColor() svggradientprimitive2d.hxx 61)

Chromium* プロジェクトのコード (C++)。

```
bool FileIOTest::MatchesResult(const TestStep& a,
                               const TestStep& b) {
  ....
  return (a.data_size == a.data_size &&             // <=
          std::equal(a.data, a.data + a.data_size, b.data));
}
```

PVS-Studio 警告: V501 (英語) There are identical sub-expressions to the left and to the right of the '==' operator: a.data_size == a.data_size cdm_file_io_test.cc 367 (V501 '==' 演算子の左辺と右辺に同じ部分式があります。a.data_size == a.data_size cdm_file_io_test.cc 367)

FreeCAD プロジェクトのコード (C++)。

```
bool FaceTypedBSpline::isEqual(const TopoDS_Face &faceOne,
                               const TopoDS_Face &faceTwo) const
{
  ....
  if (surfaceOne->IsURational() !=
      surfaceTwo->IsURational())
    return false;
  if (surfaceTwo->IsVRational() !=         // <=
      surfaceTwo->IsVRational())           // <=
    return false;
  if (surfaceOne->IsUPeriodic() !=
      surfaceTwo->IsUPeriodic())
    return false;
  if (surfaceOne->IsVPeriodic() !=
      surfaceTwo->IsVPeriodic())
    return false;
  if (surfaceOne->IsUClosed() !=
      surfaceTwo->IsUClosed())
    return false;
  if (surfaceOne->IsVClosed() !=
      surfaceTwo->IsVClosed())
    return false;
  if (surfaceOne->UDegree() !=
      surfaceTwo->UDegree())
    return false;
```

```
  if (surfaceOne->VDegree() !=
      surfaceTwo->VDegree())
    return false;
  ....
}
```

PVS-Studio 警告: V501 (英語) There are identical sub-expressions 'surfaceTwo->IsVRational()' to the left and to the right of the '!=' operator. modelrefine.cpp 780 (V501 '!=' 演算子の左辺と右辺に同じ部分式 'surfaceTwo->IsVRational()' があります。modelrefine.cpp 780)

Serious Engine プロジェクトのコード (C++)。

```
class CTexParams {
public:

  inline BOOL IsEqual( CTexParams tp) {
    return tp_iFilter     == tp.tp_iFilter &&
           tp_iAnisotropy == tp_iAnisotropy &&            // <=
           tp_eWrapU      == tp.tp_eWrapU &&
           tp_eWrapV      == tp.tp_eWrapV; };
  ....
};
```

PVS-Studio 警告: V501 (英語) There are identical sub-expressions to the left and to the right of the '==' operator: tp_iAnisotropy == tp_iAnisotropy gfx_wrapper.h 180 (V501 '==' 演算子の左辺と右辺に同じ部分式があります。tp_iAnisotropy == tp_iAnisotropy gfx_wrapper.h 180)

Qt プロジェクトのコード (C++)。

```
inline bool qCompare(QImage const &t1, QImage const &t2, ....)
{
  ....
  if (t1.width() != t2.width() || t2.height() != t2.height()) {
  ....
}
```

PVS-Studio 警告: V501 (英語) There are identical sub-expressions to the left and to the right of the '==' operator: t2.height() != t2.height() qtest_gui.h 101 (V501 '==' 演算子の左辺と右辺に同じ部分式があります。t2.height() != t2.height() qtest_gui.h 101)

FreeBSD* プロジェクトのコード (C)。

```
static int
compare_sh(const void *_a, const void *_b)
{
  const struct ipfw_sopt_handler *a, *b;

  a = (const struct ipfw_sopt_handler *)_a;
```

```
  b = (const struct ipfw_sopt_handler *)_b;
  ....
  if ((uintptr_t)a->handler < (uintptr_t)b->handler)
    return (-1);
  else if ((uintptr_t)b->handler > (uintptr_t)b->handler) // <=
    return (1);

  return (0);
}
```

PVS-Studio 警告: V501 (英語) There are identical sub-expressions '(uintptr_t) b->handler' to the left and to the right of the '>' operator. ip_fw_sockopt.c 2893 (V501 '>' 演算子の左辺と右辺に同じ部分式 '(uintptr_t) b->handler' があります。ip_fw_sockopt.c 2893)

Mono* プロジェクトのコード (C#)。

```
static bool AreEqual (VisualStyleElement value1,
                      VisualStyleElement value2)
{
  return
    value1.ClassName == value1.ClassName && // <=
    value1.Part == value2.Part &&
    value1.State == value2.State;
}
```

PVS-Studio 警告: V3001 (英語) There are identical sub-expressions 'value1.ClassName' to the left and to the right of the '==' operator. ThemeVisualStyles.cs 2141 (V3001 '==' 演算子の左辺と右辺に同じ部分式 'value1.ClassName' があります。ThemeVisualStyles.cs 2141)

Mono* プロジェクトのコード (C#)。

```
public int ExactInference (TypeSpec u, TypeSpec v)
{
  ....
  var ac_u = (ArrayContainer) u;
  var ac_v = (ArrayContainer) v;
  ....
  var ga_u = u.TypeArguments;
  var ga_v = v.TypeArguments;
  ....
  if (u.TypeArguments.Length != u.TypeArguments.Length) // <=
    return 0;

  ....
}
```

PVS-Studio 警告: V3001 (英語) There are identical sub-expressions 'u.TypeArguments.Length' to the left and to the right of the '!=' operator. generic.cs 3135 (V3001 '!=' 演算子の左辺と右辺に同じ部分式 'u.TypeArguments.Length' があります。generic.cs 3135)

MonoDevelop プロジェクトのコード (C#)。

```
Accessibility DeclaredAccessibility { get; }
bool IsStatic { get; }

private bool MembersMatch(ISymbol member1, ISymbol member2)
{
  if (member1.Kind != member2.Kind)
  {
    return false;
  }

  if (member1.DeclaredAccessibility !=          // <=1
      member1.DeclaredAccessibility             // <=1
   || member1.IsStatic != member1.IsStatic)     // <=2
  {
    return false;
  }

  if (member1.ExplicitInterfaceImplementations().Any() ||
      member2.ExplicitInterfaceImplementations().Any())
  {
    return false;
  }

  return SignatureComparer
    .HaveSameSignatureAndConstraintsAndReturnTypeAndAccessors(
       member1, member2, this.IsCaseSensitive);
}
```

PVS-Studio 警告: V3001 (英語) There are identical sub-expressions 'member1.IsStatic' to the left and to the right of the '!=' operator. CSharpBinding AbstractImplementInterfaceService.CodeAction.cs 545 (V3001 '!=' 演算子の左辺と右辺に同じ部分式 'member1.IsStatic' があります。CSharpBinding AbstractImplementInterfaceService.CodeAction.cs 545)

Haiku プロジェクトのコード (C++)。

```
int __CORTEX_NAMESPACE__ compareTypeAndID(....)
{
  int retValue = 0;
  ....
  if (lJack && rJack)
```

```
  {
    if (lJack->m_jackType < lJack->m_jackType)           // <=
    {
      return -1;
    }
    if (lJack->m_jackType == lJack->m_jackType)          // <=
    {
      if (lJack->m_index < rJack->m_index)
      {
        return -1;
      }
      else
      {
        return 1;
      }
    }
    else if (lJack->m_jackType > rJack->m_jackType)
    {
      retValue = 1;
    }
  }
  return retValue;
}
```

PVS-Studio 警告: V501 (英語) There are identical sub-expressions to the left and to the right of the '<' operator: lJack->m_jackType < lJack->m_jackType MediaJack.cpp 783 (V501 '<' 演算子の左辺と右辺に同じ部分式があります。lJack->m_jackType < lJack->m_jackType MediaJack.cpp 783)

直後にも全く同じエラーがあります。どちらのケースも、プログラマーが lJack を rJack に変更することを忘れています。

CryEngine* V プロジェクトのコード (C++)。

```
bool
CompareRotation(const Quat& q1, const Quat& q2, float epsilon)
{
  return (fabs_tpl(q1.v.x - q2.v.x) <= epsilon)
      && (fabs_tpl(q1.v.y - q2.v.y) <= epsilon)
      && (fabs_tpl(q2.v.z - q2.v.z) <= epsilon)     // <=
      && (fabs_tpl(q1.w - q2.w) <= epsilon);
}
```

PVS-Studio 警告: V501 (英語) There are identical sub-expressions to the left and to the right of the '-' operator: q2.v.z - q2.v.z entitynode.cpp 93 (V501 '-' 演算子の左辺と右辺に同じ部分式があります。q2.v.z - q2.v.z entitynode.cpp 93)

パターン: 構造体/クラスのサイズの代わりにポインターのサイズを評価する

この種のエラーは C および C++ で書かれたプログラムで *sizeof* 演算子の誤った使用により発生します。オブジェクトのサイズではなくポインターのサイズを評価することでエラーになります。次に例を示します。

```
T *a = foo1();
T *b = foo2();
x = memcmp(a, b, sizeof(a));
```

T 構造体のサイズの代わりに、ポインターのサイズが評価されています。ポインターのサイズは使用されるデータモデル (英語) に依存しますが、通常は 4 または 8 です。その結果、構造体のサイズではないメモリーの異なる部分が比較されます。

正しいコードは次のとおりです。

```
x = memcmp(a, b, sizeof(T));
```

または

```
x = memcmp(a, b, sizeof(*a));
```

では、実際に発生したエラーを見てみましょう。次のエラーは **CryEngine* V** プロジェクトのコード (C++) で見つかりました。

```
bool
operator==(const SComputePipelineStateDescription& other) const
{
  return 0 == memcmp(this, &other, sizeof(this));
}
```

PVS-Studio 警告: V579 (英語) The memcmp function receives the pointer and its size as arguments. It is possibly a mistake. Inspect the third argument. graphicspipelinestateset.h 58 (V579 memcmp 関数がポインターとそのサイズを引数として受け取りました。誤りの可能性があります。3 つ目の引数を調べてください。graphicspipelinestateset.h 58)

Unreal* Engine 4 プロジェクトのコード (C++)。

```
bool FRecastQueryFilter::IsEqual(
  const INavigationQueryFilterInterface* Other) const
{
  // @NOTE: not type safe, should be changed when
  // another filter type is introduced
  return FMemory::Memcmp(this, Other, sizeof(this)) == 0;
}
```

PVS-Studio 警告: V579 (英語) The memcmp function receives the pointer and its size as arguments. It is possibly a mistake. Inspect the third argument. pimplrecastnavmesh.cpp 172 (V579 memcmp

関数がポインターとそのサイズを引数として受け取りました。誤りの可能性があります。3 つ目の引数を調べてください。pimplrecastnavmesh.cpp 172)

パターン: 引数の繰り返し Cmp(A, A)

比較関数は通常、ほかの比較関数を呼び出します。ここで発生する可能性があるエラーの 1 つは、参照/ポインターが同じオブジェクトに 2 回渡されるエラーです。次に例を示します。

```
x = memcmp(A, A, sizeof(T));
```

ここで、オブジェクト *A* は自身と比較されます。もちろん無意味です。

次のエラーはデバッガー **GDB** のコード (C) で見つかりました。

```
static int
psymbol_compare (const void *addr1, const void *addr2,
                 int length)
{
  struct partial_symbol *sym1 = (struct partial_symbol *) addr1;
  struct partial_symbol *sym2 = (struct partial_symbol *) addr2;

  return (memcmp (&sym1->ginfo.value, &sym1->ginfo.value,    // <=
                  sizeof (sym1->ginfo.value)) == 0
          && sym1->ginfo.language == sym2->ginfo.language
          && PSYMBOL_DOMAIN (sym1) == PSYMBOL_DOMAIN (sym2)
          && PSYMBOL_CLASS (sym1) == PSYMBOL_CLASS (sym2)
          && sym1->ginfo.name == sym2->ginfo.name);
}
```

PVS-Studio 警告: V549 (英語) The first argument of 'memcmp' function is equal to the second argument. psymtab.c 1580 (V549 'memcmp' 関数の 1 つ目の引数が 2 つ目の引数と同じです。psymtab.c 1580)

CryEngine* SDK プロジェクトのコード (C++)。

```
inline bool operator != (const SEfResTexture &m) const
{
  if (stricmp(m_Name.c_str(), m_Name.c_str()) != 0 ||   // <=
      m_TexFlags != m.m_TexFlags ||
      m_bUTile != m.m_bUTile ||
      m_bVTile != m.m_bVTile ||
      m_Filter != m.m_Filter ||
      m_Ext != m.m_Ext ||
      m_Sampler != m.m_Sampler)
    return true;
  return false;
}
```

PVS-Studio 警告: V549 (英語) The first argument of 'stricmp' function is equal to the second argument. ishader.h 2089 (V549 'stricmp' 関数の 1 つ目の引数が 2 つ目の引数と同じです。ishader.h 2089)

PascalABC.NET プロジェクトのコード (C#)。

```
private List<string> enum_consts = new List<string>();
public override bool IsEqual(SymScope ts)
{
  EnumScope es = ts as EnumScope;
  if (es == null) return false;
  if (enum_consts.Count != es.enum_consts.Count) return false;
  for (int i = 0; i < es.enum_consts.Count; i++)
    if (string.Compare(enum_consts[i],
                       this.enum_consts[i], true) != 0)
      return false;
  return true;
}
```

PVS-Studio 警告: V3038 (英語) The 'enum_consts[i]' argument was passed to 'Compare' method several times. It is possible that other argument should be passed instead. CodeCompletion SymTable.cs 2206 (V3038 'enum_consts[i]' 引数が 'Compare' メソッドに複数回渡されています。他の引数を代わりに渡す必要がある可能性があります。CodeCompletion SymTable.cs 2206)

少し説明が必要です。これは *Compare* 関数の引数のエラーです。

```
string.Compare(enum_consts[i], this.enum_consts[i], true)
```

enum_consts[i] と *this.enum_consts[i]* は全く同じものです。正しいコードは次のようになります。

```
string.Compare(es.enum_consts[i], this.enum_consts[i], true)
```

または

```
string.Compare(enum_consts[i], es.enum_consts[i], true)
```

パターン: チェックの繰り返し A==B && A==B

同じチェックが 2 回行われることもプログラミングでよくあるエラーです。次に例を示します。

```
return A == B &&
       C == D &&   // <=
       C == D &&   // <=
       E == F;
```

このケースでは、2 つのミスが考えられます。1 つ目は、プログラマーが同じ比較を 2 回書いた場合で、害のないものです。この場合、余分な比較を 1 つ削除すればいいだけです。2 つ目は、ほかの変数を比較すべきところをプログラマーが入力ミスをした場合で、たちの悪いものです。

この種のコードを確認するときは、細心の注意を払う必要があります。脅かすわけではありませんが、このエラーは **GCC** コンパイラーのコード (C) にも含まれていました。

```
static bool
dw_val_equal_p (dw_val_node *a, dw_val_node *b)
{
  ....
  case dw_val_class_vms_delta:
    return (!strcmp (a->v.val_vms_delta.lbl1,
                     b->v.val_vms_delta.lbl1)
            && !strcmp (a->v.val_vms_delta.lbl1,
                        b->v.val_vms_delta.lbl1));
  ....
}
```

PVS-Studio 警告: V501 (英語) There are identical sub-expressions '!strcmp(a->v.val_vms_delta.lbl1, b->v.val_vms_delta.lbl1)' to the left and to the right of the '&&' operator. dwarf2out.c 1428 (V501 '&&' 演算子の左辺と右辺に同じ部分式 '!strcmp(a->v.val_vms_delta.lbl1, b->v.val_vms_delta.lbl1)' があります。dwarf2out.c 1428)

関数 *strcmp* が同じ引数のセットで 2 回呼び出されています。

Unreal* Engine 4 プロジェクトのコード (C++)。

```
FORCEINLINE
bool operator==(const FShapedGlyphEntryKey& Other) const
{
  return FontFace == Other.FontFace
    && GlyphIndex == Other.GlyphIndex   // <=
    && FontSize == Other.FontSize
    && FontScale == Other.FontScale
    && GlyphIndex == Other.GlyphIndex;  // <=
}
```

PVS-Studio 警告: V501 (英語) There are identical sub-expressions 'GlyphIndex == Other.GlyphIndex' to the left and to the right of the '&&' operator. fontcache.h 139 (V501 '&&' 演算子の左辺と右辺に同じ部分式 'GlyphIndex == Other.GlyphIndex' があります。fontcache.h 139)

Serious Engine プロジェクトのコード (C++)。

```
inline BOOL CValuesForPrimitive::operator==(....)
{
  return (
 (....) &&
 (vfp_ptPrimitiveType == vfpToCompare.vfp_ptPrimitiveType) &&
 ....
 (vfp_ptPrimitiveType == vfpToCompare.vfp_ptPrimitiveType) &&
```

```
 ....
);
```

PVS-Studio 警告: V501 (英語) There are identical sub-expressions '(vfp_ptPrimitiveType == vfpToCompare.vfp_ptPrimitiveType)' to the left and to the right of the '&&' operator. worldeditor.h 580 (V501 '&&' 演算子の左辺と右辺に同じ部分式 '(vfp_ptPrimitiveType == vfpToCompare.vfp_ptPrimitiveType)' があります。worldeditor.h 580)

Oracle* VM Virtual Box プロジェクトのコード (C++)。

```
typedef struct SCMDIFFSTATE
{
  ....
  bool  fIgnoreTrailingWhite;
  bool  fIgnoreLeadingWhite;
  ....
} SCMDIFFSTATE;
/* Pointer to a diff state. */

typedef SCMDIFFSTATE *PSCMDIFFSTATE;

/* Compare two lines */
DECLINLINE(bool) scmDiffCompare(PSCMDIFFSTATE pState, ....)
{
  ....
  if (pState->fIgnoreTrailingWhite    // <=
   || pState->fIgnoreTrailingWhite)   // <=
    return scmDiffCompareSlow(....);
  ....
}
```

PVS-Studio 警告: V501 (英語) There are identical sub-expressions 'pState->fIgnoreTrailingWhite' to the left and to the right of the '||' operator. scmdiff.cpp 238 (V501 '||' 演算子の左辺と右辺に同じ部分式 'pState->fIgnoreTrailingWhite' があります。scmdiff.cpp 238)

パターン: memcmp 関数で返される値の不正な使用

memcmp 関数は次の *int* 型の値を返します。

- < 0 - buf1 が buf2 未満の場合
- 0 - buf1 と buf2 が同じ場合
- > 0 - buf1 が buf2 よりも大きい場合

'>0' には 1 以外の任意の数も含まれることに注意してください (2、3、100、256、1024、5555、65536 など)。このため、この結果を *char* 型および *short* 型の変数に格納すると、(上位) ビットが失われ、プログラム実行のロジックに反することがあります。

また、結果を定数 1 または -1 で比較できないことも意味します。つまり、次のコードは間違いです。

```
if (memcmp(a, b, sizeof(T)) == 1)
if (memcmp(x, y, sizeof(T)) == -1)
```

正しいコードは次のとおりです。

```
if (memcmp(a, b, sizeof(T)) > 0)
if (memcmp(a, b, sizeof(T)) < 0)
```

このコードが危険なのは、誤った状態でも動作する可能性があることです。このエラーは、新しいプラットフォームへ移行したり、コンパイラーのバージョンを変更したときに、明らかになることがあります。

ReactOS* プロジェクトのコード (C++)。

```
HRESULT WINAPI CRecycleBin::CompareIDs(....)
{
  ....
  return MAKE_HRESULT(SEVERITY_SUCCESS, 0,
   (unsigned short)memcmp(pidl1->mkid.abID,
                          pidl2->mkid.abID,
                          pidl1->mkid.cb));
}
```

PVS-Studio 警告: V642 (英語) Saving the 'memcmp' function result inside the 'unsigned short' type variable is inappropriate. The significant bits could be lost breaking the program's logic. recyclebin.cpp 542 (V642 'memcmp' 関数の結果を 'unsigned short' 型の変数内に保存することは不適切です。有効ビットが失われプログラムのロジックに影響を与える可能性があります。recyclebin.cpp 542)

Firebird* プロジェクトのコード (C++)。

```
SSHORT TextType::compare(ULONG len1, const UCHAR* str1,
ULONG len2, const UCHAR* str2)
{
  ....
  SSHORT cmp = memcmp(str1, str2, MIN(len1, len2));

  if (cmp == 0)
    cmp = (len1 < len2 ? -1 : (len1 > len2 ? 1 : 0));
  return cmp;
}
```

PVS-Studio 警告: V642 (英語) Saving the 'memcmp' function result inside the 'short' type variable is inappropriate. The significant bits could be lost breaking the program's logic. texttype.cpp 338 (V642 'memcmp' 関数の結果を 'short' 型の変数内に保存することは不適切です。有効ビットが失われプログラムのロジックに影響を与える可能性があります。texttype.cpp 338)

CoreCLR プロジェクトのコード (C++)。

```
bool operator( )(const GUID& _Key1, const GUID& _Key2) const
  { return memcmp(&_Key1, &_Key2, sizeof(GUID)) == -1; }
```

PVS-Studio 警告: V698 (英語) Expression 'memcmp(....) == -1' is incorrect. This function can return not only the value '-1', but any negative value. Consider using 'memcmp(....) < 0' instead. sos util.cpp 142 (V698 式 'memcmp(....) == -1' は不正です。この関数は '-1' だけでなく任意の負の値を返すことができます。代わりに 'memcmp(....) < 0' の使用を検討してください。sos util.cpp 142)

OpenToonz* プロジェクトのコード (C++)。

```
bool TFilePath::operator<(const TFilePath &fp) const
{
  ....
  char differ;
  differ = _wcsicmp(iName.c_str(), jName.c_str());
  if (differ != 0)
    return differ < 0 ? true : false;
  ....
}
```

PVS-Studio 警告: V642 (英語) Saving the '_wcsicmp' function result inside the 'char' type variable is inappropriate. The significant bits could be lost breaking the program's logic. tfilepath.cpp 328 (V642 '_wcsicmp' 関数の結果を 'char' 型の変数内に保存することは不適切です。有効ビットが失われプログラムのロジックに影響を与える可能性があります。tfilepath.cpp 328)

パターン: null 参照の不正なチェック

このエラーは C# プログラムでよく見かけられます。プログラマーは比較関数で *as* 演算子による型のキャストを書くことがあります。エラーになるのは、不用意にプログラマーが元の *null* 参照ではなく新しい null 参照を確認した場合です。例を見てみましょう。

```
ChildT foo = obj as ChildT;
if (obj == null)
  return false;
if (foo.zzz()) {}
```

obj 変数が null 参照を含む場合、*if (obj == null)* を確認してもエラーになりません。しかし、*as* 演算子が null 参照を返す場合はエラーになる可能性があります。正しいコードは次のようになります。

```
ChildT foo = obj as ChildT;
if (foo == null)
  return false;
if (foo.zzz()) {}
```

一般に、このエラーはプログラマーの不注意により発生します。同様のエラーは C および C++ プログラムでも起こる可能性がありますが、まだデータベースには含まれていません。

MonoDevelop プロジェクトのコード (C#)。

```
public override bool Equals (object o)
{
  SolutionItemReference sr = o as SolutionItemReference;
  if (o == null)
    return false;
  return (path == sr.path) && (id == sr.id);
}
```

PVS-Studio 警告: V3019 (英語) Possibly an incorrect variable is compared to null after type conversion using 'as' keyword. Check variables 'o', 'sr'. MonoDevelop.Core SolutionItemReference.cs 81 (V3019 'as' キーワードを使用した型変換の後に不正な変数を null と比較している可能性があります。変数 'o'、'sr' を確認してください。MonoDevelop.Core SolutionItemReference.cs 81)

CoreFX* プロジェクトのコード (C#)。

```
public override bool Equals(object comparand)
{
  CredentialHostKey comparedCredentialKey =
                                  comparand as CredentialHostKey;

  if (comparand == null)
  {
    // This covers also the compared == null case
    return false;
  }

  bool equals = string.Equals(AuthenticationType,
        comparedCredentialKey.AuthenticationType, ....
  ....
}
```

PVS-Studio 警告: V3019 (英語) Possibly an incorrect variable is compared to null after type conversion using 'as' keyword. Check variables 'comparand', 'comparedCredentialKey'. CredentialCache.cs 4007 (V3019 'as' キーワードを使用した型変換の後に不正な変数を null と比較している可能性があります。変数 'comparand'、'comparedCredentialKey' を確認してください。CredentialCache.cs 4007)

Roslyn プロジェクトのコード (C#)。

```
public override bool Equals(object obj)
{
  var d = obj as DiagnosticDescription;

  if (obj == null)
    return false;
```

```
  if (!_code.Equals(d._code))
    return false;
  ....
}
```

PVS-Studio 警告: V3019 (英語) Possibly an incorrect variable is compared to null after type conversion using 'as' keyword. Check variables 'obj', 'd'. DiagnosticDescription.cs 201 (V3019 'as' キーワードを使用した型変換の後に不正な変数を null と比較している可能性があります。変数 'obj'、'd' を確認してください。DiagnosticDescription.cs 201)

Roslyn プロジェクトのコード (C#)。

```
protected override bool AreEqual(object other)
{
  var otherResourceString = other as LocalizableResourceString;
  return
    other != null &&
    _nameOfLocalizableResource ==
      otherResourceString._nameOfLocalizableResource &&
    _resourceManager == otherResourceString._resourceManager &&
    _resourceSource == otherResourceString._resourceSource &&
    ....
}
```

PVS-Studio 警告: V3019 (英語) Possibly an incorrect variable is compared to null after type conversion using 'as' keyword. Check variables 'other', 'otherResourceString'. LocalizableResourceString.cs 121 (V3019 'as' キーワードを使用した型変換の後に不正な変数を null と比較している可能性があります。変数 'other'、'otherResourceString' を確認してください。LocalizableResourceString.cs 121)

MSBuild プロジェクトのコード (C#)。

```
public override bool Equals(object obj)
{
   AssemblyNameExtension name = obj as AssemblyNameExtension;
   if (obj == null)  // <=
   {
     return false;
   }
   ....
}
```

PVS-Studio 警告: V3019 (英語) Possibly an incorrect variable is compared to null after type conversion using 'as' keyword. Check variables 'obj', 'name'. AssemblyRemapping.cs 64 (V3019 'as' キーワードを使用した型変換の後に不正な変数を null と比較している可能性があります。変数 'obj'、'name' を確認してください。AssemblyRemapping.cs 64)

Mono* プロジェクトのコード (C#)。

```
public override bool Equals (object o)
{
  UrlMembershipCondition umc = (o as UrlMembershipCondition);
  if (o == null)                                        // <=
    return false;

  ....

  return (String.Compare (u, 0, umc.Url, ....) == 0); // <=
}
```

PVS-Studio 警告: V3019 (英語) Possibly an incorrect variable is compared to null after type conversion using 'as' keyword. Check variables 'o', 'umc'. UrlMembershipCondition.cs 111 (V3019 'as' キーワードを使用した型変換の後に不正な変数を null と比較している可能性があります。変数 'o'、'umc' を確認してください。UrlMembershipCondition.cs 111)

Media Portal 2 プロジェクトのコード (C#)。

```
public override bool Equals(object obj)
{
  EpisodeInfo other = obj as EpisodeInfo;
  if (obj == null) return false;
  if (TvdbId > 0 && other.TvdbId > 0)
    return TvdbId == other.TvdbId;
  ....
}
```

PVS-Studio 警告: V3019 (英語) Possibly an incorrect variable is compared to null after type conversion using 'as' keyword. Check variables 'obj', 'other'. EpisodeInfo.cs 560 (V3019 'as' キーワードを使用した型変換の後に不正な変数を null と比較している可能性があります。変数 'obj'、'other' を確認してください。EpisodeInfo.cs 560)

NASA* World Wind プロジェクトのコード (C#)。

```
public int CompareTo(object obj)
{
  RenderableObject robj = obj as RenderableObject;
  if(obj == null)                                  // <=
    return 1;
  return this.m_renderPriority.CompareTo(robj.RenderPriority);
}
```

PVS-Studio 警告: V3019 (英語) Possibly an incorrect variable is compared to null after type conversion using 'as' keyword. Check variables 'obj', 'robj'. RenderableObject.cs 199 (V3019 'as'

キーワードを使用した型変換の後に不正な変数を null と比較している可能性があります。変数 'obj'、'robj' を確認してください。RenderableObject.cs 199)

パターン: 不正なループ

一部の関数では、項目のコレクションが比較されます。もちろん、比較にはループの異なるバリエーションが使用されます。プログラマーが不用意にコードを書くと、比較関数の書き方を間違えることになります。例をいくつか見てみましょう。

Trans-Proteomic Pipeline プロジェクトのコード (C++)。

```
bool Peptide::operator==(Peptide& p) {
  ....
  for (i = 0, j = 0;
       i < this->stripped.length(), j < p.stripped.length();
       i++, j++) {
  ....
}
```

PVS-Studio 警告: V521 (英語) Such expressions using the ',' operator are dangerous. Make sure the expression is correct. tpplib peptide.cpp 191 (V521 ',' 演算子を使用した式は危険です。式が正しいことを確認してください。tpplib peptide.cpp 191)

カンマ演算子は条件で使用されることに注意してください。カンマの左に書かれた条件は無視されるため、このコードが正しくないのは明らかです。左の条件は評価されますが、条件を評価した結果は使用されません。

Qt プロジェクトのコード (C++)。

```
bool equals( class1* val1, class2* val2 ) const
{
  ...
  size_t size = val1->size();
  ...
  while ( --size >= 0 ){
    if ( !comp(*itr1,*itr2) )
      return false;
    itr1++;
    itr2++;
  }
  ...
}
```

PVS-Studio 警告: V547 (英語) Expression '-- size >= 0' is always true. Unsigned type value is always >= 0. QtCLucene arrays.h 154 (V547 式 '-- size >= 0' は常に真です。符号なし型の値は常に >= 0 になります。QtCLucene arrays.h 154)

CLucene プロジェクトのコード (C++)。

```
class Arrays
{
  ....
   bool equals( class1* val1, class2* val2 ) const{
     static _comparator comp;
     if ( val1 == val2 )
       return true;
     size_t size = val1->size();
     if ( size != val2->size() )
       return false;
     _itr1 itr1 = val1->begin();
     _itr2 itr2 = val2->begin();
     while ( --size >= 0 ){
       if ( !comp(*itr1,*itr2) )
         return false;
       itr1++;
       itr2++;
     }
   return true;
  }
  ....
}
```

PVS-Studio 警告: V547 (英語) Expression '-- size >= 0' is always true. Unsigned type value is always >= 0. arrays.h 154 (V547 式 '-- size >= 0' は常に真です。符号なし型の値は常に >= 0 になります。arrays.h 154)

Mono* プロジェクトのコード (C#)。

```
public override bool Equals (object obj)
{
  ....
  for (int i=0; i < list.Count; i++) {
    bool found = false;
    for (int j=0; i < ps.list.Count; j++) {     // <=
      if (list [i].Equals (ps.list [j])) {
        found = true;
        break;
      }
    }
    if (!found)
      return false;
  }
  return true;
}
```

PVS-Studio 警告: V3015 (英語) It is likely that a wrong variable is being compared inside the 'for' operator. Consider reviewing 'i' corlib-net_4_x PermissionSet.cs 607 (V3015 誤った変数を 'for' 演算子内で比較している可能性があります。'i' を確認してください。corlib-net_4_x PermissionSet.cs 607)

入力ミスにより、入れ子のループ内で変数 *i* の代わりに変数 *j* が使用されたと考えられます。

```
for (int j=0; j < ps.list.Count; j++)
```

パターン: A = getA()、B = GetA()

プログラマーは、比較変数でこのようなコードを書くことがよくあります。

```
if (GetA().x == GetB().x && GetA().y == GetB().y)
```

中間変数は条件のサイズを減らすためや最適化に使用されます。

```
Type A = GetA();
Type B = GetB();
if (A.x == B.x && A.y == B.y)
```

しかし、不用意に、同じ値で一時変数を初期化してしまうことがあります。

```
Type A = GetA();
Type B = GetA();
```

実際のアプリケーションのエラーを見てみましょう。

LibreOffice プロジェクトのコード (C++)。

```
bool CmpAttr(
  const SfxPoolItem& rItem1, const SfxPoolItem& rItem2)
{
  ....
  bool bNumOffsetEqual = false;
  ::boost::optional<sal_uInt16> oNumOffset1 =
        static_cast<const SwFmtPageDesc&>(rItem1).GetNumOffset();
  ::boost::optional<sal_uInt16> oNumOffset2 =
        static_cast<const SwFmtPageDesc&>(rItem1).GetNumOffset();

  if (!oNumOffset1 && !oNumOffset2)
  {
    bNumOffsetEqual = true;
  }
  else if (oNumOffset1 && oNumOffset2)
  {
    bNumOffsetEqual = oNumOffset1.get() == oNumOffset2.get();
  }
  else
```

```
  {
    bNumOffsetEqual = false;
  }
  ....
}
```

PVS-Studio 警告: V656 (英語) Variables 'oNumOffset1', 'oNumOffset2' are initialized through the call to the same function. It's probably an error or un-optimized code. Check lines: 68, 69. findattr.cxx 69 (V656 変数 'oNumOffset1'、'oNumOffset2' が同じ関数の呼び出しで初期化されています。エラーまたは最適化されていないコードです。行 68、69 を確認してください。findattr.cxx 69)

Qt プロジェクトのコード (C++)。

```
AtomicComparator::ComparisonResult
IntegerComparator::compare(const Item &o1,
                           const AtomicComparator::Operator,
                           const Item &o2) const
{
  const Numeric *const num1 = o1.as<Numeric>();
  const Numeric *const num2 = o1.as<Numeric>();

  if(num1->isSigned() || num2->isSigned())
  ....
}
```

PVS-Studio 警告: V656 (英語) Variables 'num1', 'num2' are initialized through the call to the same function. It's probably an error or un-optimized code. Consider inspecting the 'o1.as < Numeric > ()' expression. Check lines: 220, 221. qatomiccomparators.cpp 221 (V656 変数 'oNumOffset1'、'oNumOffset2' が同じ関数の呼び出しで初期化されています。エラーまたは最適化されていないコードです。'o1.as < Numeric > ()' 式を確認してください。行 220、221 を確認してください。qatomiccomparators.cpp 221)

パターン: コードのずさんなコピー

これまで引用した多くのエラーは、ずさんなコピーペーストの結果と言えます。分類可能なエラーについては、いくつかのカテゴリーに分類し、対応するセクションで説明することが理にかなってると考えました。しかし、ずさんなコードのコピーが原因であることが明白でも、分類方法が思い浮かばないエラーもいくつかあります。ここでは、これらのエラーを集めてみました。

CoreCLR プロジェクトのコード (C++)。

```
int __cdecl Compiler::RefCntCmp(const void* op1, const void* op2)
{
  ....
  if (weight1)
  {
    ....
```

```
    if (varTypeIsGC(dsc1->TypeGet()))
    {
      weight1 += BB_UNITY_WEIGHT / 2;
    }
    if (dsc1->lvRegister)
    {
      weight1 += BB_UNITY_WEIGHT / 2;
    }
  }

  if (weight1)
  {
    ....
    if (varTypeIsGC(dsc2->TypeGet()))
    {
      weight1 += BB_UNITY_WEIGHT / 2;       // <=
    }
    if (dsc2->lvRegister)
    {
      weight2 += BB_UNITY_WEIGHT / 2;
    }
  }
  ....
}
```

PVS-Studio 警告: V778 (英語) Two similar code fragments were found. Perhaps, this is a typo and 'weight2' variable should be used instead of 'weight1'. clrjit lclvars.cpp 2702 (V778 2 つの類似コードが見つかりました。入力ミスと思われます。'weight2' 変数を 'weight1' の代わりに使用する必要があります。clrjit lclvars.cpp 2702)

この関数は長いため、記事では一部を省略しています。コードの一部がコピーされ、プログラマーが変数 *weight1* を *weight2* に置換することを忘れています。

WPF samples by Microsoft* プロジェクトのコード (C#)。

```
public int Compare(GlyphRun a, GlyphRun b)
{
  ....
  if (aPoint.Y > bPoint.Y)      // <=
  {
    return -1;
  }
  else if (aPoint.Y > bPoint.Y) // <=
  {
    result = 1;
  }
  else if (aPoint.X < bPoint.X)
```

```
  {
    result = -1;
  }
  else if (aPoint.X > bPoint.X)
  {
    result = 1;
  }
  ....
}
```

PVS-Studio 警告: V3003 (英語) The use of 'if (A) {...} else if (A) {...}' pattern was detected. There is a probability of logical error presence. Check lines: 418, 422. txtserializerwriter.cs 418 (V517 'if (A) {...} else if (A) {...}' パターンの使用を検出しました。論理的なエラーの可能性があります。行 418、422 を確認してください。txtserializerwriter.cs 418)

PascalABC.NET プロジェクトのコード (C#)。

```
public void CompareInternal(....)
{
  ....
  else if (left is int64_const)
    CompareInternal(left as int64_const, right as int64_const);
  ....
  else if (left is int64_const)
    CompareInternal(left as int64_const, right as int64_const);
  ....
}
```

PVS-Studio 警告: V3003 (英語) The use of 'if (A) {...} else if (A) {...}' pattern was detected. There is a probability of logical error presence. Check lines: 597, 631. ParserTools SyntaxTreeComparer.cs 597 (V517 'if (A) {...} else if (A) {...}' パターンの使用を検出しました。論理的なエラーの可能性があります。行 597、631 を確認してください。ParserTools SyntaxTreeComparer.cs 597)

SharpDevelop プロジェクトのコード (C#)。

```
public int Compare(SharpTreeNode x, SharpTreeNode y)
{
  ....
  if (typeNameComparison == 0) {
    if (x.Text.ToString().Length < y.Text.ToString().Length)
      return -1;
    if (x.Text.ToString().Length < y.Text.ToString().Length)
      return 1;
  }
  ....
}
```

PVS-Studio 警告: V3021 (英語) There are two 'if' statements with identical conditional expressions. The first 'if' statement contains method return. This means that the second 'if' statement is senseless. NamespaceTreeNode.cs 87 (V3021 同じの条件式の 2 つの 'if' 文があります。1 つ目の 'if' 文にメソッド return が含まれています。このため 2 つ目の 'if' 文は無意味です。NamespaceTreeNode.cs 87)

Coin3D プロジェクトのコード (C++)。

```
int
SbProfilingData::operator == (const SbProfilingData & rhs) const
{
  if (this->actionType != rhs.actionType) return FALSE;
  if (this->actionStartTime != rhs.actionStopTime) return FALSE;
  if (this->actionStartTime != rhs.actionStopTime) return FALSE;
  ....
}
```

PVS-Studio 警告: V649 (英語) There are two 'if' statements with identical conditional expressions. The first 'if' statement contains function return. This means that the second 'if' statement is senseless. Check lines: 1205, 1206. sbprofilingdata.cpp 1206 (V649 同じの条件式の 2 つの 'if' 文があります。1 つ目の 'if' 文に関数 return が含まれています。このため 2 つ目の 'if' 文は無意味です。行 1205、1206 を確認してください。sbprofilingdata.cpp 1206)

Spring* プロジェクトのコード (C++)。

```
bool operator < (const aiFloatKey& o) const
  {return mTime < o.mTime;}
bool operator > (const aiFloatKey& o) const
  {return mTime < o.mTime;}
```

PVS-Studio 警告: V524 (英語) It is odd that the body of '>' function is fully equivalent to the body of '<' function. assimp 3dshelper.h 470 (V524 '>' 関数の本体が '<' 関数の本体と全く同じです。assimp 3dshelper.h 470)

このセクションの最後のエラーとして、**MySQL*** プロジェクトのコード (C++) で見つかったエラーを紹介します。

```
static int rr_cmp(uchar *a,uchar *b)
{
  if (a[0] != b[0])
    return (int) a[0] - (int) b[0];
  if (a[1] != b[1])
    return (int) a[1] - (int) b[1];
  if (a[2] != b[2])
    return (int) a[2] - (int) b[2];
  if (a[3] != b[3])
    return (int) a[3] - (int) b[3];
  if (a[4] != b[4])
```

```
    return (int) a[4] - (int) b[4];
  if (a[5] != b[5])
    return (int) a[1] - (int) b[5]; // <=
  if (a[6] != b[6])
    return (int) a[6] - (int) b[6];
  return (int) a[7] - (int) b[7];
}
```

PVS-Studio 警告: V525 (英語) The code containing the collection of similar blocks. Check items '0', '1', '2', '3', '4', '1', '6' in lines 680, 682, 684, 689, 691, 693, 695. sql records.cc 680 (V525 コードに類似ブロックのコレクションが含まれています。行 680、682、684、689、691、693、695 の項目 '0'、'1'、'2'、'3'、'4'、'1'、'6' を確認してください。sql records.cc 680)

恐らく、プログラマーは、最初の比較を書いた後、2 つ目以降の比較を書くのが面倒になり、テキストブロックをバッファーにコピーしたのでしょう。

```
if (a[1] != b[1])
  return (int) a[1] - (int) b[1];
```

そして、必要な回数だけプログラムのテキストに貼り付けた後、インデックスを変更し、その際に 1 カ所変更するのを忘れたと考えられます。

```
if (a[5] != b[5])
  return (int) a[1] - (int) b[5];
```

注: このエラーの詳細は、「The Ultimate Question of Programming, Refactoring, and Everything」(英語) の「1. Don't do the compiler's job」を参照してください。

パターン: Equals メソッドによる null 参照の不正な処理

C# では null 参照を使用した Equals メソッドの実装が行われており、null 参照が引数として渡された場合は正しく処理されます。しかし、残念ながら、すべてのメソッドがこの規則に従って実装されているとは限りません。

GitExtensions プロジェクトのコード (C#)。

```
public override bool Equals(object obj)
{
  return GetHashCode() == obj.GetHashCode(); // <=
}
```

PVS-Studio 警告: V3115 (英語) Passing 'null' to 'Equals(object obj)' method should not result in 'NullReferenceException'. Git.hub Organization.cs 14 (V3115 'null' を 'Equals(object obj)' メソッドに渡して 'NullReferenceException' をスローすることは推奨されません。Git.hub Organization.cs 14)

PascalABC.NET プロジェクトのコード (C#)。

```
public override bool Equals(object obj)
{
```

```
  var rhs = obj as ServiceReferenceMapFile;
  return FileName == rhs.FileName;
}
```

PVS-Studio 警告: V3115 (英語) Passing 'null' to 'Equals' method should not result in 'NullReferenceException'. ICSharpCode.SharpDevelop ServiceReferenceMapFile.cs 31 (V3115 'null' を 'Equals' メソッドに渡して 'NullReferenceException' をスローすることは推奨されません。ICSharpCode.SharpDevelop ServiceReferenceMapFile.cs 31)

その他のエラー

G3D* Content Pak プロジェクトのコード (C++)。

```
bool Matrix4::operator==(const Matrix4& other) const {
  if (memcmp(this, &other, sizeof(Matrix4) == 0)) {
    return true;
  }
  ...
}
```

PVS-Studio 警告: V575 (英語) The 'memcmp' function processes '0' elements. Inspect the 'third' argument. graphics3D matrix4.cpp 269 (V575 'memcmp' 関数が '0' の要素を処理しています。3 つ目の引数を確認してください。graphics3D matrix4.cpp 269)

閉じ括弧が誤った場所に挿入されています。その結果、比較される部分が文 *sizeof(Matrix4) == 0* により評価されています。すべてのクラスのサイズは 0 よりも大きいため、式の結果は 0 になります。つまり、0 の部分が比較されます。

正しいコードは次のとおりです。

```
if (memcmp(this, &other, sizeof(Matrix4)) == 0) {
```

Wolfenstein* 3D プロジェクトのコード (C++)。

```
inline int operator!=( quat_t a, quat_t b )
{
  return ( ( a.x != b.x ) || ( a.y != b.y ) ||
           ( a.z != b.z ) && ( a.w != b.w ) );
}
```

PVS-Studio 警告: V648 (英語) Priority of the '&&' operation is higher than that of the '||' operation. math_quaternion.h 167 (V648 '&&' 演算子の優先順位は '||' 演算子の優先順位より上です。math_quaternion.h 167)

コードの一部で *&&* 演算子が || の代わりに誤って記述されたと考えられます。

FlightGear プロジェクトのコード (C)。

```
static int tokMatch(struct Token* a, struct Token* b)
{
  int i, l = a->strlen;
  if(!a || !b) return 0;
  ....
}
```

PVS-Studio 警告: V595 (英語) The 'a' pointer was utilized before it was verified against nullptr. Check lines: 478, 479. codegen.c 478 (V595 ポインターが null かどうか検証される前に 'a' ポインターが使用されています。行 478、479 を確認してください。codegen.c 478)

プログラマーは関数が *0* を返すと想定していますが、1 つ目の引数として *null* を関数に渡すと、null ポインターの逆参照になります。

WinMerge プロジェクトのコード (C++)。

```
int TimeSizeCompare::CompareFiles(int compMethod,
                                  const DIFFITEM &di)
{
  UINT code = DIFFCODE::SAME;
  ...
  if (di.left.size != di.right.size)
  {
    code &= ~DIFFCODE::SAME;
    code = DIFFCODE::DIFF;
  }
  ...
}
```

PVS-Studio 警告: V519 (英語) The 'code' variable is assigned values twice successively. Perhaps this is a mistake. Check lines: 79, 80. Merge timesizecompare.cpp 80 (V519 'code' 変数に 2 回連続して値が代入されています。ミスの可能性があります。行 79、80 を確認してください。Merge timesizecompare.cpp 80)

ReactOS* プロジェクトのコード (C++)。

```
#define IsEqualGUID(rguid1, rguid2) \
  (!memcmp(&(rguid1), &(rguid2), sizeof(GUID)))

static int ctl2_find_guid(....)
{
  MSFT_GuidEntry *guidentry;
  ...
  if (IsEqualGUID(guidentry, guid)) return offset;
  ...
}
```

PVS-Studio 警告: V512 (英語) A call of the 'memcmp' function will lead to underflow of the buffer 'guidentry'. oleaut32 typelib2.c 320 (V512 A 'memcmp' 関数を呼び出すとバッファー 'guidentry' がアンダーフローになります。oleaut32 typelib2.c 320)

ポインターは 1 つ目の引数として書かれています。この場合、ポインターのアドレスが評価されるため、無意味です。

正しいコードは次のとおりです。

```
if (IsEqualGUID(*guidentry, guid)) return offset;
```

IronPython and IronRuby プロジェクトのコード (C#)。

```
public static bool Equals(float x, float y) {
  if (x == y) {
    return !Single.IsNaN(x);
  }
  return x == y;
}
```

PVS-Studio 警告: V3024 (英語) An odd precise comparison: x == y. Consider using a comparison with defined precision: Math.Abs(A - B) < Epsilon. FloatOps.cs 1048 (V3024 奇妙な精度の比較 x == y が行われています。定義された精度 Math.Abs(A - B) < Epsilon で比較することを検討してください。FloatOps.cs 1048)

NaN に対するチェックを行っている理由は不明です。条件 *(x == y)* が真の場合、*NaN* は自身を含むほかの値と等しくないため、*x* と *y* はどちらも *NaN* とは異なります。NaN に対するチェックは必要なく、コードは次のように短縮できると考えられます。

```
public static bool Equals(float x, float y) {
  return x == y;
}
```

Mono* プロジェクトのコード (C#)。

```
public bool Equals (CounterSample other)
{
  return
    rawValue         == other.rawValue         &&
    baseValue        == other.counterFrequency &&   // <=
    counterFrequency == other.counterFrequency &&   // <=
    systemFrequency  == other.systemFrequency  &&
    timeStamp        == other.timeStamp        &&
    timeStamp100nSec == other.timeStamp100nSec &&
    counterTimeStamp == other.counterTimeStamp &&
    counterType      == other.counterType;
}
```

PVS-Studio 警告: V3112 (英語) An abnormality within similar comparisons. It is possible that a typo is present inside the expression 'baseValue == other.counterFrequency'. System-net_4_x CounterSample.cs 139 (V3112 同様の比較内に異常な点があります。式 'baseValue == other.counterFrequency' に入力ミスがないか確認してください。System-net_4_x CounterSample.cs 139)

エラーのあるプログラムがなぜ動作するのか

比較関数は、プログラムで非常に重要な、責任の重いタスクを行っていることを考えると、この記事で紹介したエラーを含むプログラムが、エラーになることなく動作しているのは奇跡的と言えます。

これらのプログラムがエラーになることなく動作している理由をいくつか考えてみましょう。

1. 多くの関数で、オブジェクトの一部のみ正しくない比較が行われている。プログラムの多くのタスクでは、部分的な比較しか行われていない。
2. 関数が不正に動作する状況に (まだ) 陥っていない。例えば、*memcmp* 関数呼び出しの結果が *char* 型の変数に格納された場合、null ポインターによるエラーが発生する可能性があります。プログラムが動作するのは単に運がいいだけです。
3. この記事で紹介した比較関数を、ほとんど使用していないか、全く使用していない。
4. プログラムはエラーにならないが間違った動作をしている。

推奨事項

この記事では、比較関数に多くのエラーが含まれることを示しました。これらの関数は単体テストを使用して確認すべきであることも示しました。

特に、*Equals* 関数その他では、比較演算子の単体テストを書くべきです。

私は、この記事を読むまで、プログラマーの間で「比較関数は一目で分かる単純な関数だから、これらの関数の単体テストは余計な手間にすぎない。どうせエラーは検出されないはず。」という認識があったと確信しています。この記事を読んだ後もこの認識を変えないプログラマーはいないでしょう。

コードのレビューや統計解析ツールもエラーの発見に大いに役立ちます。

まとめ

この記事では、非常に優れたエキスパートにより開発されている多くの有名なプロジェクトを取り上げました。これらのプロジェクトは異なる手法を使用して徹底的にテストされます。しかし、それでも PVS-Studio がエラーを発見することを止められませんでした。PVS-Studio は、テストに使用されているほかの手法を補完し、コードの品質と信頼性の向上に役立つ、優れたツールであると言えるでしょう。

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。