Weak Interfaces → Weak Defences

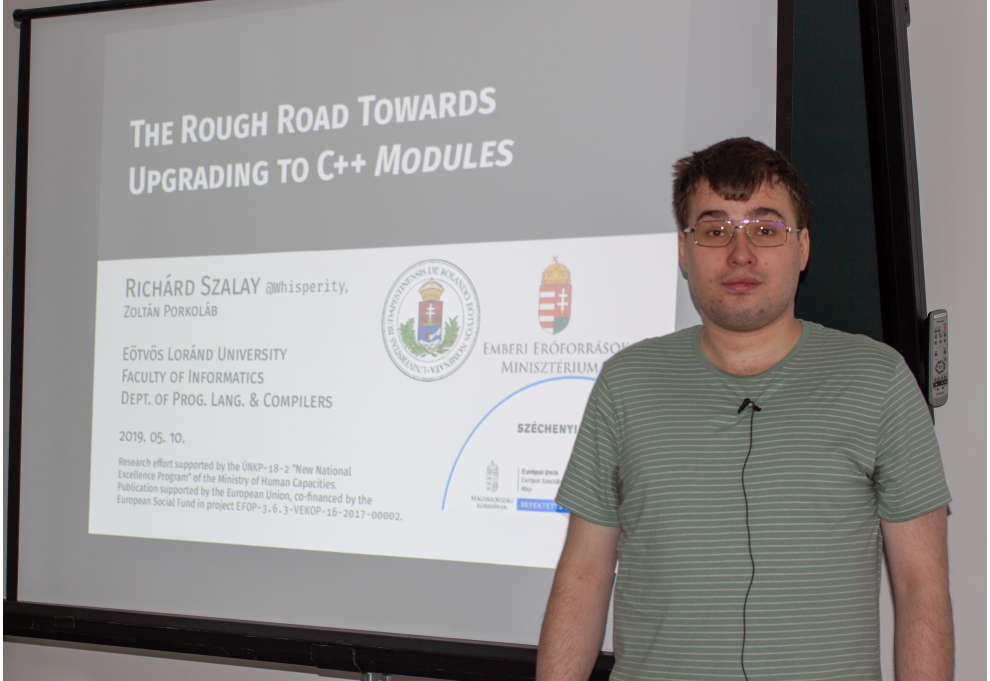
...the Bane of Implicit Conversions in our Function

Richárd Szalay,
Ábel Sinkovics,
Zoltán Porkoláb

Eötvös Loránd University, Faculty of Informatics
Department of Programming Languages & Compilers

C++Now 2021, 2021. 05. 04.

Prepared with the professional support of the Doctoral Student Scholarship Program of the Co-operative Doctoral Program of the Ministry of Innovation and Technology financed from the National Research, Development and Innovation Fund.


The Rough Road Towards
Upgrading to C++ Modules
Richárd Szalay @Whisperity,
Zoltán Porkoláb
Eötvös Loránd University
Faculty of Informatics
Dept. of Prog. Lang. & Compilers
2019. 05. 10.
Research effort supported by the ÚNKP-18-2 "New National
Excellence Program" of the Ministry of Human Capacities.
Publication supported by the European Union, co-financed by the
European Social Fund in project EFOP-3.6.3-VEKOP-16-2017-00002.
Emberi Erőforrások
Minisztérium
Széchenyi

1. Argument swapping
   - “Name-based analysis”
   - Reactiveness $\longrightarrow$ proactiveness
   - Existing guidelines

2. Implicit conversions

3. Type-based guards
   - The Solution?
   - The Analysis
   - Empirica

4. Summary

Argument selection defects

Coined by Rice *et al.*[1], but goes back longer.

Wrong argument (from available “set of expressions”) passed to function call.

```
int f(string host_name, int port, string message);

string author = "Richard.Szalay";
string greeting = "Hello, World!";

f(author, 8080, greeting);
```

[1] Rice et al., “Detecting Argument Selection Defects”.

Argument swaps ($\subset$ argument selection defects)

Special case when arguments are as intended, but out of order.

Previous literature findings:

- adjacency increases chance of mistake
- too many parameters (≥ 5) increase chance of mistake, too

```
int f₂(string message, string hostName, int port);
```

Proper variable names?

Name our variables properly![2][3][4][5]

[2] Liu et al., “Nomen est Omen: Exploring and Exploiting Similarities between Argument and Parameter Names”.

[3] Pradel and Gross, “Name-Based Analysis of Equally Typed Method Arguments”.

[4] Rice et al., “Detecting Argument Selection Defects”.

[5] Scott et al., “Out of Sight, Out of Place: Detecting and Assessing Swapped Arguments”.

Proper variable names?

Name our variables properly![2][3][4][5]

Simple ideas to compare:

```
void F(int μ, int λ) { /* ... */ }
// ⋮
foo(β, ϑ);
```

[2] Liu et al., “Nomen est Omen: Exploring and Exploiting Similarities between Argument and Parameter Names”.

[3] Pradel and Gross, “Name-Based Analysis of Equally Typed Method Arguments”.

[4] Rice et al., “Detecting Argument Selection Defects”.

[5] Scott et al., “Out of Sight, Out of Place: Detecting and Assessing Swapped Arguments”.

“Name-based” analysis

Simple ideas to compare:

```
void F(int μ, int λ) { /* ... */ }
// ⋮
foo(β, ϑ);
```

- String equality, suffix/prefix coverage, pattern containment
- Edit distances (e.g. Levehnstein-distance, …)
- Morpheme extraction[6]
- Typo analysis (e.g. key distance)

[6] Scott et al., “Out of Sight, Out of Place: Detecting and Assessing Swapped Arguments”.

Tools that exist

Link rot affects papers of previous slides… 😕

[7] Scott et al., “Out of Sight, Out of Place: Detecting and Assessing Swapped Arguments”.

Tools that exist

Link rot affects papers of previous slides… 😕

- GrammaTech, *SwapD*[7].
- Clang-Tidy – *hopefully* upcoming: Varjú, Barancsuk, and Whisperity, *Add `readability-suspicious-call-argument` check*

[7] Scott et al., “Out of Sight, Out of Place: Detecting and Assessing Swapped Arguments”.

```
diff --git a/gcc/config/i386/adxintrin.h b/gcc/config/i386/adxintrin.h
index 9c4152b9f360c0f9be408c84da4950ded8ad5654..7acdaf4ab6f7675401eab1c512175e6493620
--- a/gcc/config/i386/adxintrin.h
+++ b/gcc/config/i386/adxintrin.h
@@ -33,7 +33,7 @@ __attribute__((__gnu_inline__, __always_inline__, __artificial__))
 _subborrow_u32 (unsigned char __CF, unsigned int __X,
                 unsigned int __Y, unsigned int *__P)
 {
-  return __builtin_ia32_sbb_u32 (__CF, __Y, __X, __P);
+  return __builtin_ia32_sbb_u32 (__CF, __X, __Y, __P);
 }

 extern __inline unsigned char
@@ -58,7 +58,7 @@ __attribute__((__gnu_inline__, __always_inline__, __artificial__))
 _subborrow_u64 (unsigned char __CF, unsigned long long __X,
                 unsigned long long __Y, unsigned long long *__P)
 {
-  return __builtin_ia32_sbb_u64 (__CF, __Y, __X, __P);
+  return __builtin_ia32_sbb_u64 (__CF, __X, __Y, __P);
 }

 extern __inline unsigned char
```

GNU Compiler Collection

---

andreser, *_subborrow_u64 argument order inconsistent*

dom/html/HTMLInputElement.cpp file | annotate | diff | comparison | revisions

```
1.1   --- a/dom/html/HTMLInputElement.cpp
1.2   +++ b/dom/html/HTMLInputElement.cpp
1.3   @@ -4352,18 +4352,18 @@ HTMLInputElement::HandleTypeChange(uint8
1.4
1.5      // Updating mFocusedValue in consequence:
1.6      // If the new type fires a change event on blur, but the previous type
1.7      // doesn't, we should set mFocusedValue to the current value.
1.8      // Otherwise, if the new type doesn't fire a change event on blur, but the
1.9      // previous type does, we should clear out mFocusedValue.
1.10     if (MayFireChangeOnBlur(mType) && !MayFireChangeOnBlur(oldType)) {
1.11       GetValue(mFocusedValue);
1.12  -  } else if (!IsSingleLineTextControl(mType, false) &&
1.13  -             IsSingleLineTextControl(oldType, false)) {
1.14  +  } else if (!IsSingleLineTextControl(false, mType) &&
1.15  +             IsSingleLineTextControl(false, oldType)) {
1.16       mFocusedValue.Truncate();
1.17     }
1.18
1.19     UpdateHasRange();
1.20
1.21     // Do not notify, it will be done after if needed.
1.22     UpdateAllValidityStates(false);
1.23   }
```

mozilla-central Deployed from e404c980045c at 2021-03-01T18:43:48Z.

Capella, *Suspicious code with probably reversed parms in call to* `IsSingleLineTextControl(bool, uint32_t)`

The downside of names

- You need to “have them” (or figure them out)

The downside of names

- You need to “have them” (or figure them out)
- Complex/unnamed expressions?

The downside of names

- You need to “have them” (or figure them out)
- Complex/unnamed expressions?
  - `fun(*objw, objw⟶getSome());`

The downside of names

- You need to “have them” (or figure them out)
- Complex/unnamed expressions?
  - `fun(*objw, objw⟶getSome());`
  - `fun(1, 2);`

The downside of names

- You need to “have them” (or figure them out)
- Complex/unnamed expressions?
  - `fun(*objw, objw⟶getSome());`
  - `fun(1, 2);`
  - `fun(variance, 4 + 8);`

The downside of names

- You need to “have them” (or figure them out)
- Complex/unnamed expressions?
  - `fun(*objw, objw⟶getSome());`
  - `fun(1, 2);`
  - `fun(variance, 4 + 8);`
- Oh and...

The downside of names

- You need to “have them” (or figure them out)
- Complex/unnamed expressions?
  - `fun(*objw, objw⟶getSome());`
  - `fun(1, 2);`
  - `fun(variance, 4 + 8);`
- Oh and...? `void someAPIFunction(int, int, long);`

src/backend/executor/execMain.c

```
722 ExecCheckRTEPermsModified(Oid relOid, Oid userid, Bitmapset *modifiedCols,
723                           AclMode requiredPerms)
724 {
725     int         col = -1;
726
727     /*
728      * When the query doesn't explicitly update any columns, allow the query
729      * if we have permission on any column of the rel.  This is to handle
730      * SELECT FOR UPDATE as well as possible corner cases in UPDATE.
731      */
732     if (bms_is_empty(modifiedCols))
733     {
734         if (pg_attribute_aclcheck_all(relOid, userid, requiredPerms,
735                                       ACLMASK_ANY) != ACLCHECK_OK)
736             return false;
737     }
```

2 < 1st argument 'relOid' (passed to 'table_oid') looks like it might be swapped with the 2nd, 'userid' (passed to 'roleid')

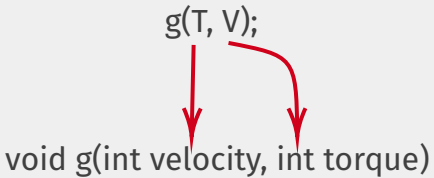
src/include/utils/acl.h

```
260                                         Oid roleid, AclMode mode);
261 extern AclResult pg_attribute_aclcheck_all(Oid table_oid, Oid roleid,
                    1 in the call to 'pg_attribute_aclcheck_all', declared here >
262                                         AclMode mode, AclMaskHow how);
263 extern AclResult pg_class_aclcheck(Oid table_oid, Oid roleid, AclMode mode);
264 extern AclResult pg_database_aclcheck(Oid db_oid, Oid roleid, AclMode mode);
265 extern AclResult pg_proc_aclcheck(Oid proc_oid, Oid roleid, AclMode mode);
266 extern AclResult pg_language_aclcheck(Oid lang_oid, Oid roleid, AclMode mode);
```

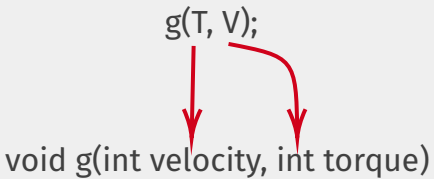
- `reloid` – **rel**ation (table) **o**wner(?) **ID**
- `roleid` – **role** (user) **ID**

The PostgreSQL Global Development Group, *PostgreSQL*

Reactive ⟶ Proactive

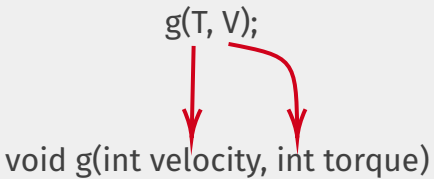


Reactive $\longrightarrow$ Proactive



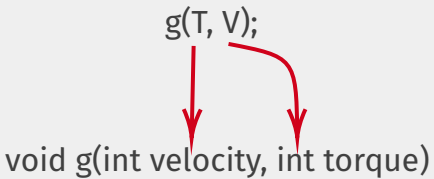
- Detect call sites…after-the-fact

Reactive $\longrightarrow$ Proactive



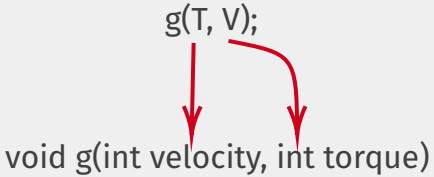
- Detect call sites…after-the-fact
- Significant enough mismatch **in name** $\rightarrow$ report

Reactive $\longrightarrow$ Proactive



- Detect call sites…after-the-fact
- Significant enough mismatch **in name** $\rightarrow$ report
- False positives?

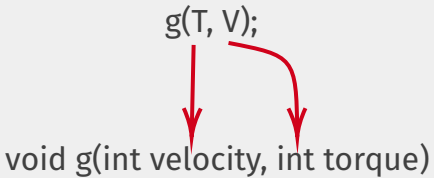
Reactive $\longrightarrow$ Proactive




```
void g₂(velocity_t V,
        torque_t   T);
```

- Detect call sites…after-the-fact
- Significant enough mismatch **in name** $\rightarrow$ report
- False positives?

Reactive $\longrightarrow$ Proactive

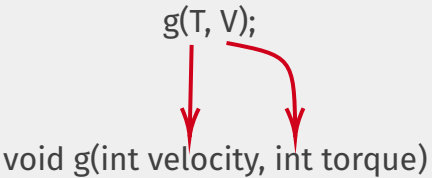



```
void g₂(velocity_t V,
        torque_t   T);
```

- Detect call sites…after-the-fact
- Significant enough mismatch **in name** $\rightarrow$ report
- False positives?

- Use the type system to guard us!

Reactive ⟶ Proactive

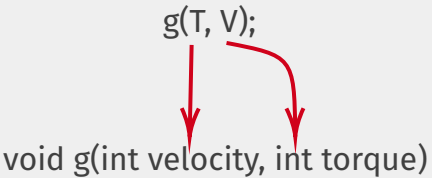


- Detect call sites…after-the-fact
- Significant enough mismatch **in name** → report
- False positives?

```
void g₂(velocity_t V,
        torque_t   T);
```

- Use the type system to guard us!
- Swapped expressions → ↯ *compile error*

Reactive ⟶ Proactive



- Detect call sites…after-the-fact
- Significant enough mismatch **in name** → report
- False positives?

```
void g₂(velocity_t V,
        torque_t   T);
```

- Use the type system to guard us!
- Swapped expressions → ↯ *compile error*
- Pure developer intent!

diff.c

```
3347 static void emit_binary_diff(struct diff_options *o,
3348                  mmfile_t *one, mmfile_t *two)
3349 {
3350     emit_diff_symbol(o, DIFF_SYMBOL_BINARY_DIFF_HEADER, NULL, 0, 0);
3351     emit_binary_diff_body(o, one, two);
3352     emit_binary_diff_body(o, two, one);
```

2 < 2nd argument 'two' (passed to 'one') looks like it might be swapped with the 3rd, 'one' (passed to 'two')

```
3353 }
```

Torvalds et al., *git*

I.24: Avoid adjacent unrelated parameters of the same type

Reason

Adjacent arguments of the same type are easily swapped by mistake.

Example, bad

Consider:

```
void copy_n(T* p, T* q, int n);  // copy from [p:p + n) to [q:q + n)
```

This is a nasty variant of a K&R C-style interface. It is easy to reverse the "to" and "from" arguments.

Use `const` for the "from" argument:

```
void copy_n(const T* p, T* q, int n);  // copy from [p:p + n) to [q:q + n)
```

Exception

If the order of the parameters is not important, there is no problem:

```
int max(int a, int b);
```

Alternative

Don't pass arrays as pointers, pass an object representing a range (e.g., a `span`):

```
void copy_n(span<const T> p, span<T> q);  // copy from p to q
```

Alternative

Define a `struct` as the parameter type and name the fields for those parameters accordingly:

```
struct SystemParams {
    string config_file;
    string output_path;
    seconds timeout;
};
void initialize(SystemParams p);
```

This tends to make invocations of this clear to future readers, as the parameters are often filled in by name at the call site.

Enforcement

(Simple) Warn if two consecutive parameters share the same type.

I.24: Avoid adjacent parameters that can be invoked by the same arguments in either order with different meaning

Reason

Adjacent arguments of the same type are easily swapped by mistake.

Example, bad

Consider:

```
void copy_n(T* p, T* q, int n);  // copy from [p:p + n) to [q:q + n)
```

This is a nasty variant of a K&R C-style interface. It is easy to reverse the "to" and "from" arguments.

Use `const` for the "from" argument:

```
void copy_n(const T* p, T* q, int n);  // copy from [p:p + n) to [q:q + n)
```

Exception

If the order of the parameters is not important, there is no problem:

```
int max(int a, int b);
```

Alternative

Don't pass arrays as pointers, pass an object representing a range (e.g., a `span`):

```
void copy_n(span<const T> p, span<T> q);  // copy from p to q
```

Alternative

Define a `struct` as the parameter type and name the fields for those parameters accordingly:

```
struct SystemParams {
    string config_file;
    string output_path;
    seconds timeout;
};
void initialize(SystemParams p);
```

This tends to make invocations of this clear to future readers, as the parameters are often filled in by name at the call site.

Note

Only the interface's designer can adequately address the source of violations of this guideline.

Enforcement strategy

(Simple) Warn if two consecutive parameters share the same type

We are still looking for a less-simple enforcement.

TYPES!

C++ is unlike Java[8]

TYPES!

- **`typedef`/`using`** 

[8] Most of the existing splutions are for Java

C++ is unlike Java[8]

TYPES!

- **typedef**/**using** 
- **const** T& troubles!

[8] Most of the existing splutions are for Java

C++ is unlike Java[8]

TYPES!

- **typedef**/**using** 
- **const** T& troubles!
- **const** T ≡ T 

[8] Most of the existing splutions are for Java

C++ is unlike Java[8]

TYPES!

- **typedef**/**using** 
- **const** T& troubles!
- **const** T ≡ T 
- Implicit conversions…

[8] Most of the existing splutions are for Java

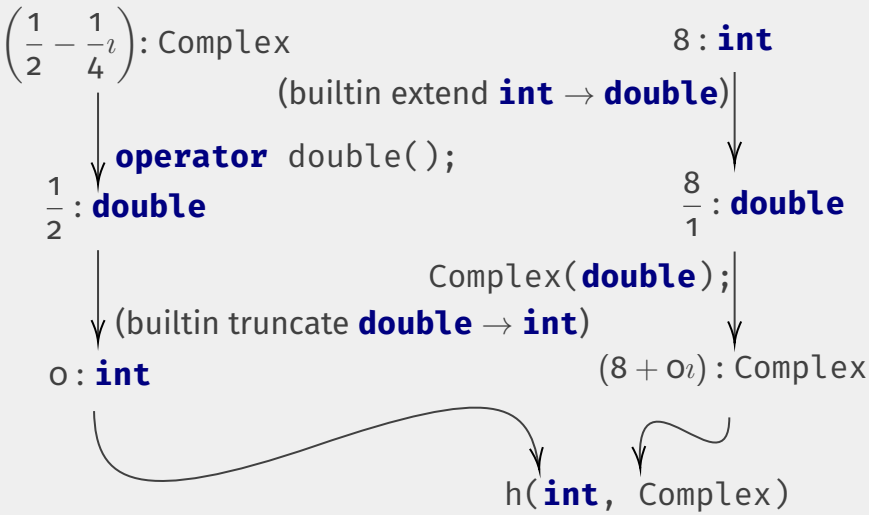
Will this be caught?

```
struct Complex { double Re, Im; /* ... */ };
void h(int Scalar, Complex Comp);

void test() {
    int     S = 8;
    Complex C = Complex{.5f, -.25f}; // = (1/2 - 1/4 i)

    h(C, S); // ← ?
}
```

It depends ...

```
struct Complex {
    double R, I;
    Complex(double real) : R(real), I(0.0) {}
    operator double() const { return R; }
    // ⋮
};
void h(int Scalar, Complex Comp);
void test() {
    int     S = 8;
    Complex C = Complex{.5f, -.25f}; // = (1/2 − 1/4 i)

    h(C, S); // ✓
}
```

Implicit conversions



Subtle mistake, expanded

```
struct Complex {
    double R, I;
    Complex(double real) : R(real), I(0.0) {}
    operator double() const { return R; }
    // ⋮
};
void h(int Scalar, Complex Comp);

void test() {
    h(Complex{.5f, -.25f}, 8); // ≡ h(C, S); from before...
    h(0, Complex{8, 0});
}
```

Implicit conversions

An ***implicit conversion sequence*** $\mathcal{T}_1 \to \mathcal{T}_2$ exists and defined as:

1. *Maybe* **standard conversion sequence**:

Implicit conversions

An ***implicit conversion sequence*** $\mathcal{T}_1 \to \mathcal{T}_2$ exists and defined as:

1. *Maybe* **standard conversion sequence**:
   1.1 *Maybe* decay (lvalue$\to$rvalue, array/function$\to$pointer)
   1.2 *Maybe* numeric adjustment

Implicit conversions

An ***implicit conversion sequence*** $\mathcal{T}_1 \to \mathcal{T}_2$ exists and defined as:

1. *Maybe* **standard conversion sequence**:
   - 1.1 *Maybe* decay (lvalue→rvalue, array/function→pointer)
   - 1.2 *Maybe* numeric adjustment
     - integral ↔ integral, enum → integral

Implicit conversions

An ***implicit conversion sequence*** $\mathcal{T}_1 \to \mathcal{T}_2$ exists and defined as:

1. *Maybe* **standard conversion sequence**:
   - 1.1 *Maybe* decay (lvalue→rvalue, array/function→pointer)
   - 1.2 *Maybe* numeric adjustment
     - integral ↔ integral, enum → integral
     - floating ↔ integral

Implicit conversions

An ***implicit conversion sequence*** $\mathcal{T}_1 \to \mathcal{T}_2$ exists and defined as:

1. *Maybe* **standard conversion sequence**:
   - 1.1 *Maybe* decay (lvalue→rvalue, array/function→pointer)
   - 1.2 *Maybe* numeric adjustment
     - integral ↔ integral, enum → integral
     - floating ↔ integral
     - derived → base class, T* → `void*`

Implicit conversions

An ***implicit conversion sequence*** $\mathcal{T}_1 \to \mathcal{T}_2$ exists and defined as:

1. *Maybe* **standard conversion sequence**:
   - 1.1 *Maybe* decay (lvalue→rvalue, array/function→pointer)
   - 1.2 *Maybe* numeric adjustment
     - integral ↔ integral, enum → integral
     - floating ↔ integral
     - derived → base class, T* → **`void`***
     - **`null`**-constants → pointer (?!)

Implicit conversions

An ***implicit conversion sequence*** $\mathcal{T}_1 \to \mathcal{T}_2$ exists and defined as:

1. *Maybe* **standard conversion sequence**:
   - 1.1 *Maybe* decay (lvalue→rvalue, array/function→pointer)
   - 1.2 *Maybe* numeric adjustment
     - integral ↔ integral, enum → integral
     - floating ↔ integral
     - derived → base class, T* → **`void`***
     - **`null`**-constants → pointer (?!)
     - Anything you can imagine → **`bool`** (!!)

Implicit conversions

An ***implicit conversion sequence*** $\mathcal{T}_1 \to \mathcal{T}_2$ exists and defined as:

1. *Maybe* **standard conversion sequence**:
   - 1.1 *Maybe* decay (lvalue→rvalue, array/function→pointer)
   - 1.2 *Maybe* numeric adjustment
     - integral ↔ integral, enum → integral
     - floating ↔ integral
     - derived → base class, T* → **`void`***
     - **`null`**-constants → pointer (?!)
     - Anything you can imagine → **`bool`** (!!)
   - 1.3 *Maybe* function pointer adjustment (“lose **`noexcept`**”)
   - 1.4 *Maybe* qualifier adjustment (“gain **`const volatile`**”)

Implicit conversions

An ***implicit conversion sequence*** $\mathcal{T}_1 \to \mathcal{T}_2$ exists and defined as:

1. *Maybe* **standard conversion sequence**:
   - 1.1 *Maybe* decay (lvalue→rvalue, array/function→pointer)
   - 1.2 *Maybe* numeric adjustment
     - integral ↔ integral, enum → integral
     - floating ↔ integral
     - derived → base class, T* → **`void`***
     - **`null`**-constants → pointer (?!)
     - Anything you can imagine → **`bool`** (!!)
   - 1.3 *Maybe* function pointer adjustment (“lose **`noexcept`**”)
   - 1.4 *Maybe* qualifier adjustment (“gain **`const volatile`**”)
2. *Maybe* user-defined conversion (one function call!)
3. *Maybe* standard conversion sequence (same as above)

Implicit conversions

An ***implicit conversion sequence*** $\mathcal{T}_1 \to \mathcal{T}_2$ exists and defined as:

1. *Maybe* **standard conversion sequence**:
   - 1.1 *Maybe* decay (lvalue→rvalue, array/function→pointer)
   - 1.2 *Maybe* numeric adjustment
     - integral ↔ integral, enum → integral
     - floating ↔ integral
     - derived → base class, T* → **`void`***
     - **`null`**-constants → pointer (?!)
     - Anything you can imagine → **`bool`** (!!)
   - 1.3 *Maybe* function pointer adjustment (“lose **`noexcept`**”)
   - 1.4 *Maybe* qualifier adjustment (“gain **`const volatile`**”)
2. *Maybe* user-defined conversion (one function call!)
3. *Maybe* standard conversion sequence (same as above)

…**if** the path taken *uniquely exists*.

Implicit conversions

An ***implicit conversion sequence*** is a unique path from $\mathcal{T}_1 \to \mathcal{T}_2$ s.t.:

1. *Maybe* **standard conversion sequence**:
   - 1.1 *Maybe* decay (lvalue→rvalue, array/function→pointer)
   - 1.2 *Maybe* numeric adjustment
     - integral ↔ integral, enum → integral
     - floating ↔ integral
     - derived → base class, T* → **`void`***
     - **`null`**-constants → pointer
     - Anything you can imagine → **`bool`**
   - 1.3 *Maybe* function pointer adjustment (“lose **`noexcept`**”)
   - 1.4 *Maybe* qualifier adjustment (“gain **`const volatile`**”)
2. *Maybe* user-defined conversion (one function call!)
3. *Maybe* standard conversion sequence (same as above)

Implicit conversions

An ***implicit conversion sequence*** is a unique path from $\mathcal{T}_1 \to \mathcal{T}_2$ s.t.:

1. *Maybe* **standard conversion sequence**:
   - 1.1 *Maybe* decay (lvalue→rvalue, array/function→pointer)
   - 1.2 *Maybe* numeric adjustment
     - ~~integral ↔ integral, enum → integral~~
     - ~~floating ↔ integral~~
     - derived → base class, T* → **`void`***
     - **`null`**-constants → pointer
     - Anything you can imagine → **`bool`**
   - 1.3 *Maybe* function pointer adjustment (“lose **`noexcept`**”)
   - 1.4 *Maybe* qualifier adjustment (“gain **`const volatile`**”)
2. *Maybe* user-defined conversion (one function call!)
3. *Maybe* standard conversion sequence (same as above)

Implicit conversions

An ***implicit conversion sequence*** is a unique path from $\mathcal{T}_1 \to \mathcal{T}_2$ s.t.:

1. *Maybe* **standard conversion sequence**:
   - 1.1 *Maybe* decay (lvalue→rvalue, array/function→pointer)
   - 1.2 *Maybe* numeric adjustment
     - ~~integral ↔ integral, enum → integral~~
     - ~~floating ↔ integral~~
     - derived → base class, ~~T* → **void***~~
     - **`null`**-constants → pointer
     - Anything you can imagine → **`bool`**
   - 1.3 *Maybe* function pointer adjustment (“lose **`noexcept`**”)
   - 1.4 *Maybe* qualifier adjustment (“gain **`const volatile`**”)
2. *Maybe* user-defined conversion (one function call!)
3. *Maybe* standard conversion sequence (same as above)

Implicit conversions

An ***implicit conversion sequence*** is a unique path from $\mathcal{T}_1 \to \mathcal{T}_2$ s.t.:

1. *Maybe* **standard conversion sequence**:
   - 1.1 *Maybe* decay (lvalue→rvalue, array/function→pointer)
   - 1.2 *Maybe* numeric adjustment
     - ~~integral ↔ integral, enum → integral~~
     - ~~floating ↔ integral~~
     - derived → base class, ~~`T*` → `void*`~~
     - **`null`**-constants → pointer
     - ~~Anything you can imagine → **`bool`**~~
   - 1.3 *Maybe* function pointer adjustment ("lose **`noexcept`**")
   - 1.4 *Maybe* qualifier adjustment ("gain **`const volatile`**")
2. *Maybe* user-defined conversion (one function call!)
3. *Maybe* standard conversion sequence (same as above)

Implicit conversions

An ***implicit conversion sequence*** is a unique path from $\mathcal{T}_1 \to \mathcal{T}_2$ s.t.:

1. *Maybe* **standard conversion sequence**:
   - 1.1 *Maybe* decay (lvalue→rvalue, ~~array/~~function→pointer)
   - 1.2 *Maybe* numeric adjustment
     - ~~integral ↔ integral, enum → integral~~
     - ~~floating ↔ integral~~
     - derived → base class, ~~`T*` → `void*`~~
     - **`null`**-constants → pointer
     - ~~Anything you can imagine → **`bool`**~~
   - 1.3 *Maybe* function pointer adjustment (“lose **`noexcept`**”)
   - 1.4 *Maybe* qualifier adjustment (“gain **`const volatile`**”)
2. *Maybe* user-defined conversion (one function call!)
3. *Maybe* standard conversion sequence (same as above)

Implicit conversions

An ***implicit conversion sequence*** is a unique path from $\mathcal{T}_1 \to \mathcal{T}_2$ s.t.:

1. *Maybe* **standard conversion sequence**:
   - 1.1 *Maybe* decay (lvalue→rvalue, ~~array/~~function→pointer)
   - 1.2 *Maybe* numeric adjustment
     - ~~integral ↔ integral, enum → integral~~
     - ~~floating ↔ integral~~
     - derived → base class, ~~`T*` → `void*`~~
     - **`null`**-constants → pointer
     - ~~Anything you can imagine → **`bool`**~~
   - 1.3 *Maybe* function pointer adjustment (“lose **`noexcept`**”)
   - 1.4 *Maybe* qualifier adjustment (“gain **`const volatile`**”)
2. *Maybe* user-defined conversion (one function call!)
3. *Maybe* standard conversion sequence (same as above)

Types?

Strong types!

Stronger types!

Strong…err…types?

The Solution

Tuesday, May 4

7:00am MDT

Converting a State Machine to a C++ 20 Coroutine
Steve Downey

Range-Based Text Formatting - For a Future Range-Based Standard Library
Arno Schödl

Code Analysis++
Anastasia Kazakova

9:00am MDT

C++ Standard Parallelism
Bryce Adelstein Lelbach

Interactive C++ in a Jupyter Notebook Using Modules for Incremental Compilation
Steven Brandt

Simplest Safe Integers
Peter Sommerlad

Thursday, May 6

7:00am MDT

So You Think You Know How to Work With Concepts?
Andrzej Krzemieński

CMake + Conan: 3 Years Later
Mateusz Pusz

Simplest Strong Typing instead of Language Proposal (P0109)
Peter Sommerlad

9:00am MDT

Algorithms from a Compiler Developer's Toolbox
Gabor Horvath

Windows, MacOS and the Web: Lessons from Cross-platform Development at think-cell
Sebastian Theophil

Variations on variants
Roi Barkan

11:30am MDT

Designing Concurrent C++ Applications
Lucian Radu Teodorescu

C++ Insights: How Stuff Works, Lambdas and More!
Andreas Fertig

Library Approaches for Strong Type Aliases
Anthony Williams

Mixable adjacent parameters

```
void
p (int i, int j, double d, Complex c, std::string s);
```

Mixable adjacent parameters

```
void
p (int i, int j, double d, Complex c, std::string s);
```

- Is **int** mixable with **int**?

Mixable adjacent parameters

```
void
p (int i, int j, double d, Complex c, std::string s);
```

- Is **`int`** mixable with **`int`**?  ***Same type, trivially***

Mixable adjacent parameters

```
void
p (int i, int j, double d, Complex c, std::string s);
```

- Is **int** mixable with **int**?  ***Same type, trivially***
- Is **int** mixable with **double**?

Mixable adjacent parameters

```
void
p (int i, int j, double d, Complex c, std::string s);
```

- Is **`int`** mixable with **`int`**?  ***Same type, trivially***
- Is **`int`** mixable with **`double`**?  ***standard conversion***

Mixable adjacent parameters

```
void
p (int i, int j, double d, Complex c, std::string s);
```

- Is **`int`** mixable with **`int`**?  ***Same type, trivially***
- Is **`int`** mixable with **`double`**?  ***standard conversion***
- Is **`int`** mixable with `Complex`?

Mixable adjacent parameters

```
void
p (int i, int j, double d, Complex c, std::string s);
```

- Is **`int`** mixable with **`int`**? ✓ *Same type, trivially*
- Is **`int`** mixable with **`double`**? ↷ *standard conversion*
- Is **`int`** mixable with `Complex`? ↷ *standard + user*

Mixable adjacent parameters

```
void
p (int i, int j, double d, Complex c, std::string s);
```

- Is **int** mixable with **int**? ✓ *Same type, trivially*
- Is **int** mixable with **double**? *standard conversion*
- Is **int** mixable with Complex? *standard + user*
- Is **double** mixable with Complex?

Mixable adjacent parameters

```
void
p (int i, int j, double d, Complex c, std::string s);
```

- Is **`int`** mixable with **`int`**? ✓ *Same type, trivially*
- Is **`int`** mixable with **`double`**? *standard conversion*
- Is **`int`** mixable with `Complex`? *standard + user*
- Is **`double`** mixable with `Complex`? *user conversion*

Mixable adjacent parameters

```
void
p (int i, int j, double d, Complex c, std::string s);
```

- Is **`int`** mixable with **`int`**? ✓ *Same type, trivially*
- Is **`int`** mixable with **`double`**? ↷ *standard conversion*
- Is **`int`** mixable with `Complex`? ↷ *standard + user*
- Is **`double`** mixable with `Complex`? ↷ *user conversion*
- Is `Complex` mixable with `std::string`?

Mixable adjacent parameters

```
void
p (int i, int j, double d, Complex c, std::string s);
   ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~^
```

- Is **`int`** mixable with **`int`**?  *Same type, trivially*
- Is **`int`** mixable with **`double`**?  *standard conversion*
- Is **`int`** mixable with `Complex`?  *standard + user*
- Is **`double`** mixable with `Complex`?  *user conversion*
- Is `Complex` mixable with `std::string`?  *No*

Mixable adjacent parameters

```
void
p (int i, int j, double d, Complex c, std::string s);
   ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~^
```

- Is **`int`** mixable with **`int`**? ✓ *Same type, trivially*
- Is **`int`** mixable with **`double`**? ↷ *standard conversion*
- Is **`int`** mixable with `Complex`? ↷ *standard + user*
- Is **`double`** mixable with `Complex`? ↷ *user conversion*
- Is `Complex` mixable with `std::string`? ✗ *No*
- … (at most $\mathcal{O}\left(\frac{n(n-1)}{2}\right)$ checks)

Analysis in Clang-Tidy

Implemented automated analysis rule in Clang-Tidy.[9]
In the following we'll discuss 7 C and 9 C++ projects:

- curl
- git
- netdata
- PHP
- PostgreSQL
- Redis
- tmux
- Bitcoin
- CodeCompass
- guetzli
- LLVM
- OpenCV
- Protobuf
- Tesseract
- Xerces
- Z3

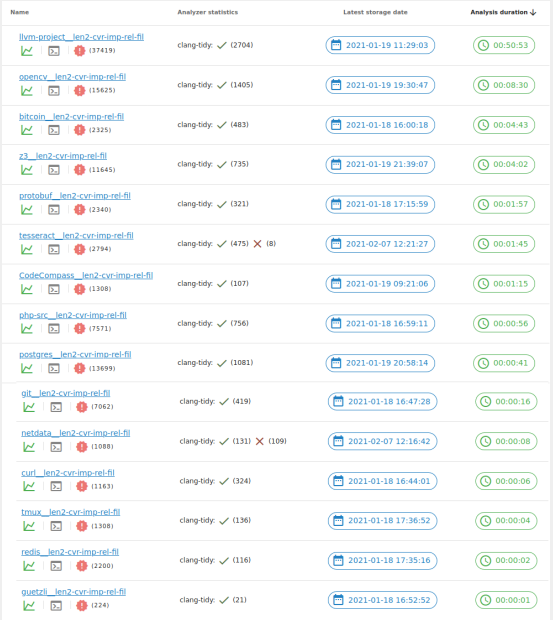
[9] Whisperity, *Add `bugprone-easily-swappable-parameters` check*.

Analysis in Clang-Tidy

Implemented automated analysis rule in Clang-Tidy.[9]
In the following we'll discuss 7 **C??** and 9 C++ projects:

- curl
- git
- netdata
- PHP
- PostgreSQL
- Redis
- tmux
- Bitcoin
- CodeCompass
- guetzli
- LLVM
- OpenCV
- Protobuf
- Tesseract
- Xerces
- Z3

[9] Whisperity, *Add `bugprone-easily-swappable-parameters` check*.



Name	Analyzer statistics	Latest storage date	Analysis duration ↓
llvm-project__len2-cvr-imp-rel-fil (37419)	clang-tidy: ✓ (2704)	2021-01-19 11:29:03	00:50:53
opencv__len2-cvr-imp-rel-fil (15625)	clang-tidy: ✓ (1405)	2021-01-19 19:30:47	00:08:30
bitcoin__len2-cvr-imp-rel-fil (2325)	clang-tidy: ✓ (483)	2021-01-18 16:00:18	00:04:43
z3__len2-cvr-imp-rel-fil (11645)	clang-tidy: ✓ (735)	2021-01-19 21:39:07	00:04:02
protobuf__len2-cvr-imp-rel-fil (2340)	clang-tidy: ✓ (321)	2021-01-18 17:15:59	00:01:57
tesseract__len2-cvr-imp-rel-fil (2794)	clang-tidy: ✓ (475) ✗ (8)	2021-02-07 12:21:27	00:01:45
CodeCompass__len2-cvr-imp-rel-fil (1308)	clang-tidy: ✓ (107)	2021-01-19 09:21:06	00:01:15
php-src__len2-cvr-imp-rel-fil (7571)	clang-tidy: ✓ (756)	2021-01-18 16:59:11	00:00:56
postgres__len2-cvr-imp-rel-fil (13699)	clang-tidy: ✓ (1081)	2021-01-19 20:58:14	00:00:41
git__len2-cvr-imp-rel-fil (7062)	clang-tidy: ✓ (419)	2021-01-18 16:47:28	00:00:16
netdata__len2-cvr-imp-rel-fil (1088)	clang-tidy: ✓ (131) ✗ (109)	2021-02-07 12:16:42	00:00:08
curl__len2-cvr-imp-rel-fil (1163)	clang-tidy: ✓ (324)	2021-01-18 16:44:01	00:00:06
tmux__len2-cvr-imp-rel-fil (1308)	clang-tidy: ✓ (136)	2021-01-18 17:36:52	00:00:04
redis__len2-cvr-imp-rel-fil (2200)	clang-tidy: ✓ (116)	2021-01-18 17:35:16	00:00:02
quetzal__len2-cvr-imp-rel-fil (224)	clang-tidy: ✓ (21)	2021-01-18 16:52:52	00:00:01

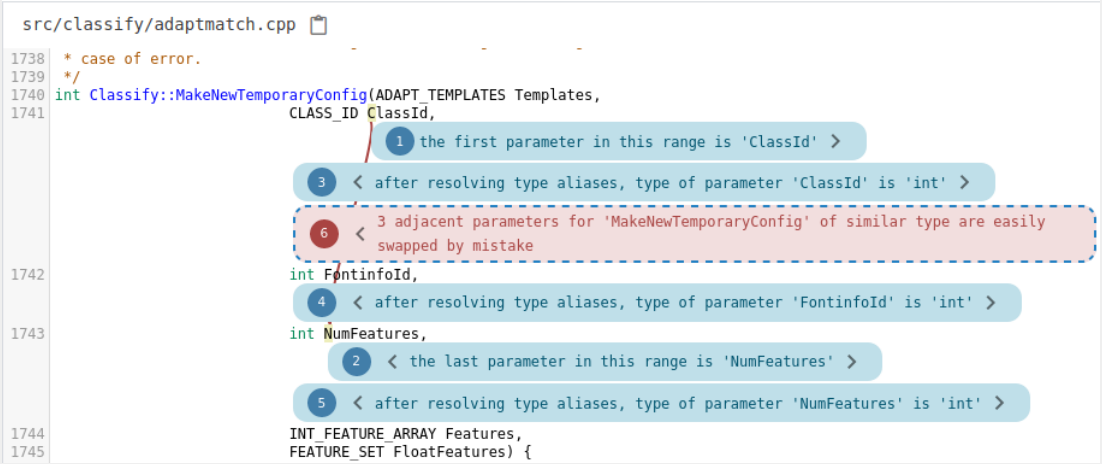


Figure: One finding from Tesseract[10] visualised using *CodeChecker*[11].

[10] Smith, Google, Inc., et al., *Tesseract OCR Engine*.
[11] Ericsson AB, *CodeChecker – an analyzer tooling, defect database and viewer*.

modules/calib3d/src/calibration.cpp

```
4167 float cv::rectify3Collinear( InputArray _cameraMatrix1, InputArray _distCoeffs1,
4168                              InputArray _cameraMatrix2, InputArray _distCoeffs2,
4169                              InputArray _cameraMatrix3, InputArray _distCoeffs3,
4170                              InputArrayOfArrays _imgpt1,
4171                              InputArrayOfArrays _imgpt3,
4172                              Size imageSize, InputArray _Rmat12, InputArray _Tmat12,
4173                              InputArray _Rmat13, InputArray _Tmat13,
4174                              OutputArray _Rmat1, OutputArray _Rmat2, OutputArray _Rmat3,
4175                              OutputArray _Pmat1, OutputArray _Pmat2, OutputArray _Pmat3,
4176                              OutputArray _Qmat,
4177                              double alpha, Size newImgSize,
4178                              Rect* roi1, Rect* roi2, int flags )
4179 {
4180     // first, rectify the 1-2 stereo pair
```

1 the first parameter in this range is '_cameraMatrix1'

5 8 adjacent parameters for 'rectify3Collinear' of similar type are easily swapped by mistake

3 at a call site, 'const _InputArray &' might bind with same force as 'cv::InputArray'

2 the last parameter in this range is '_imgpt3'

4 at a call site, 'const _InputArray &' might bind with same force as 'cv::InputArray'

More parameters → more chance of mistake

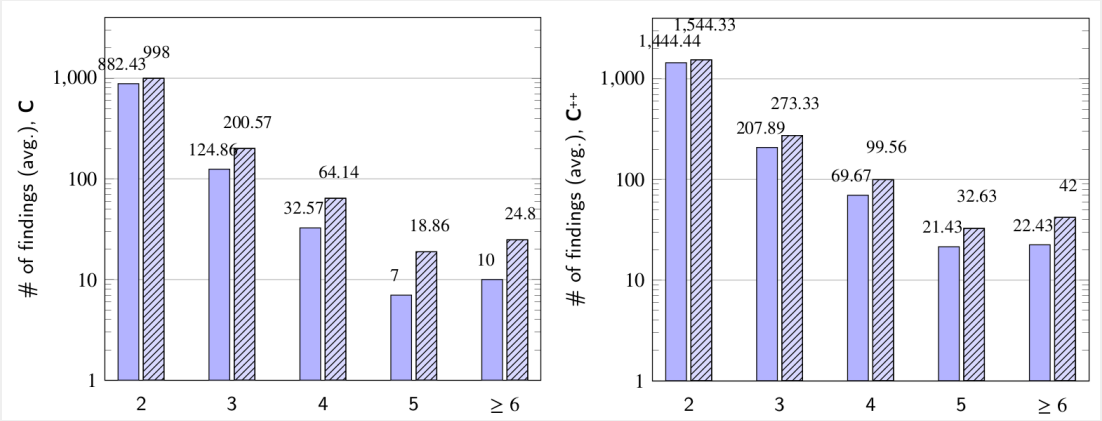
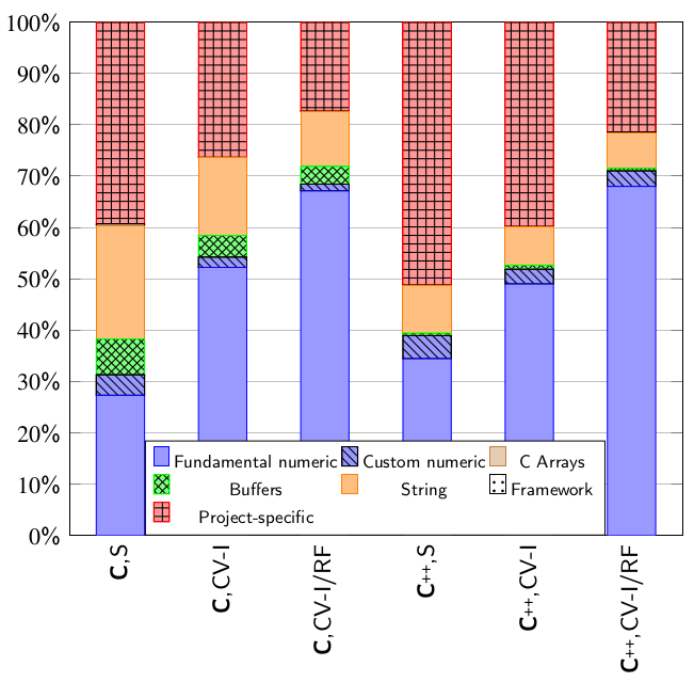
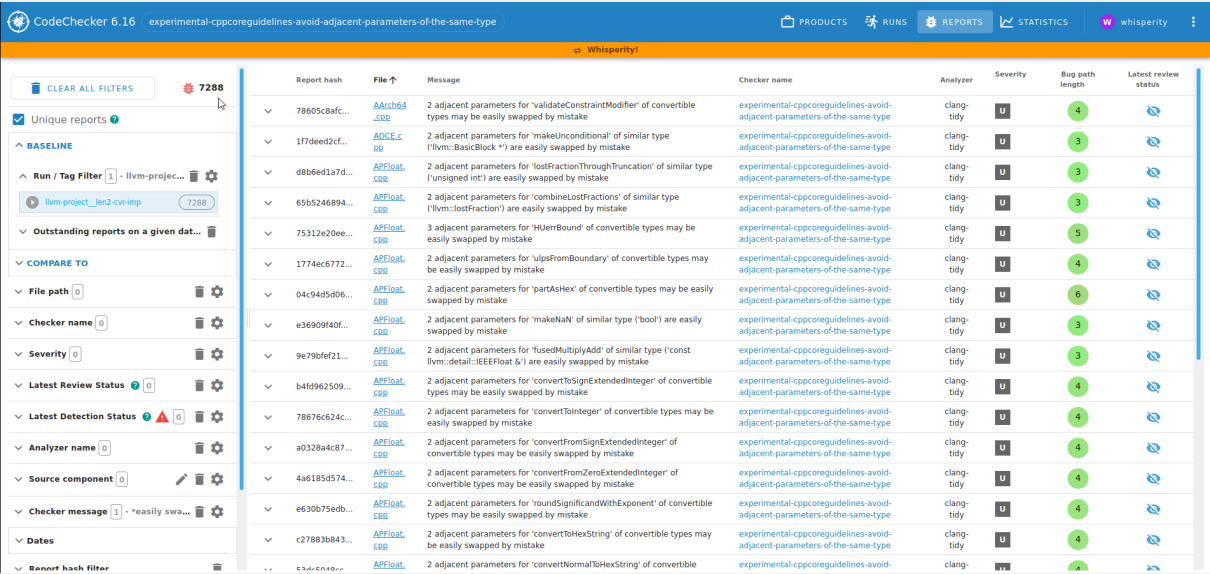


Figure: Avg. # of functions with problem. Strict mode, Generous `const` and implicit conversions.

Lang.	Project	Functions analysed (A)	Strict (S)		CV			Imp			CV ∪ Imp			
			T (total)	*% of A*	*T*	*% of A*	*+ vs. S*	*T*	*% of A*	*+ vs. S*	*T*	*% of A*	*+ vs. CV*	*+ vs. Imp*
C	curl	865	134	15.49%	153	17.69%	19	209	24.16%	75	228	26.36%	75	19
	git	5 641	1 418	25.14%	1 466	25.99%	48	1 595	28.28%	177	1 644	29.14%	178	49
	netdata	719	227	31.57%	243	33.80%	16	294	40.89%	67	308	42.84%	65	14
	PHP	5 984	1 272	21.26%	1 304	21.79%	32	1 509	25.22%	237	1 539	25.72%	235	30
	Postgres	9 436	2 696	28.57%	2 804	29.72%	108	3 708	39.30%	1 012	3 820	40.48%	1 016	112
	Redis	1 745	393	22.52%	418	23.95%	25	454	26.02%	61	484	27.74%	66	36
	TMux	1 032	248	24.03%	259	25.10%	11	298	28.88%	50	306	29.65%	47	8
C++	Bitcoin	1 773	394	22.22%	412	23.24%	18	499	28.14%	105	512	28.88%	109	22
	CodeCompass	191	27	14.14%	27	14.14%	0	28	14.66%	1	28	14.66%	1	0
	guetzli	153	72	47.06%	76	49.67%	4	75	49.02%	3	81	52.94%	5	6
	LLVM-CTE	32 339	6 109	18.89%	6 181	19.11%	72	6 817	21.08%	704	6 898	21.33%	713	81
	OpenCV	6 746	3 286	48.71%	3 414	50.61%	128	3 590	53.22%	304	3 714	55.05%	300	124
	ProtoBuf	1 997	313	15.67%	317	15.87%	4	386	19.33%	73	392	19.63%	75	6
	Tesseract	1 962	793	40.42%	797	40.62%	4	874	44.55%	81	880	44.85%	83	6
	Xerces	1 594	446	27.98%	462	28.98%	16	465	29.17%	19	532	33.38%	70	67
	Z3	9 673	2 600	26.88%	2 608	26.96%	8	2 793	28.87%	193	2 801	28.96%	193	8


100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
Fundamental numeric
Custom numeric
C Arrays
Buffers
String
Framework
Project-specific
C,S
C,CV-I
C,CV-I/RF
C++,S
C++,CV-I
C++,CV-I/RF


CodeChecker 6.16
experimental-cppcoreguidelines-avoid-adjacent-parameters-of-the-same-type
PRODUCTS
RUNS
REPORTS
STATISTICS
whisperity
Whisperity!
CLEAR ALL FILTERS
7288
Unique reports
BASELINE
Run / Tag Filter 1 - llvm-projec...
llvm-project_len2-cvr-imp
7288
Outstanding reports on a given dat...
COMPARE TO
File path 0
Checker name 0
Severity 0
Latest Review Status 0
Latest Detection Status 0
Analyzer name 0
Source component 0
Checker message 1 - *easily swa...
Dates
Report hash filter
Report hash
File
Message
Checker name
Analyzer
Severity
Bug path length
Latest review status
78605c8afc... AArch64.cpp 2 adjacent parameters for 'validateConstraintModifier' of convertible types may be easily swapped by mistake experimental-cppcoreguidelines-avoid-adjacent-parameters-of-the-same-type clang-tidy U 4
1f7deed2cf... ADCE.cpp 2 adjacent parameters for 'makeUnconditional' of similar type ('llvm::BasicBlock *') are easily swapped by mistake experimental-cppcoreguidelines-avoid-adjacent-parameters-of-the-same-type clang-tidy U 3
d8b6ed1a7d... APFloat.cpp 2 adjacent parameters for 'lostFractionThroughTruncation' of similar type ('unsigned int') are easily swapped by mistake experimental-cppcoreguidelines-avoid-adjacent-parameters-of-the-same-type clang-tidy U 3
65b5246894... APFloat.cpp 2 adjacent parameters for 'combineLostFractions' of similar type ('llvm::lostFraction') are easily swapped by mistake experimental-cppcoreguidelines-avoid-adjacent-parameters-of-the-same-type clang-tidy U 3
75312e20ee... APFloat.cpp 3 adjacent parameters for 'HUerrBound' of convertible types may be easily swapped by mistake experimental-cppcoreguidelines-avoid-adjacent-parameters-of-the-same-type clang-tidy U 5
1774ec6772... APFloat.cpp 2 adjacent parameters for 'ulpsFromBoundary' of convertible types may be easily swapped by mistake experimental-cppcoreguidelines-avoid-adjacent-parameters-of-the-same-type clang-tidy U 4
04c94d5d06... APFloat.cpp 2 adjacent parameters for 'partAsHex' of convertible types may be easily swapped by mistake experimental-cppcoreguidelines-avoid-adjacent-parameters-of-the-same-type clang-tidy U 6
e36909f40f... APFloat.cpp 2 adjacent parameters for 'makeNaN' of similar type ('bool') are easily swapped by mistake experimental-cppcoreguidelines-avoid-adjacent-parameters-of-the-same-type clang-tidy U 3
9e79bfef21... APFloat.cpp 2 adjacent parameters for 'fusedMultiplyAdd' of similar type ('const llvm::detail::IEEEFloat &') are easily swapped by mistake experimental-cppcoreguidelines-avoid-adjacent-parameters-of-the-same-type clang-tidy U 3
b4fd962509... APFloat.cpp 2 adjacent parameters for 'convertToSignExtendedInteger' of convertible types may be easily swapped by mistake experimental-cppcoreguidelines-avoid-adjacent-parameters-of-the-same-type clang-tidy U 4
78676c624c... APFloat.cpp 2 adjacent parameters for 'convertToInteger' of convertible types may be easily swapped by mistake experimental-cppcoreguidelines-avoid-adjacent-parameters-of-the-same-type clang-tidy U 4
a0328a4c87... APFloat.cpp 2 adjacent parameters for 'convertFromSignExtendedInteger' of convertible types may be easily swapped by mistake experimental-cppcoreguidelines-avoid-adjacent-parameters-of-the-same-type clang-tidy U 4
4a6185d574... APFloat.cpp 2 adjacent parameters for 'convertFromZeroExtendedInteger' of convertible types may be easily swapped by mistake experimental-cppcoreguidelines-avoid-adjacent-parameters-of-the-same-type clang-tidy U 4
e630b75edb... APFloat.cpp 2 adjacent parameters for 'roundSignificandWithExponent' of convertible types may be easily swapped by mistake experimental-cppcoreguidelines-avoid-adjacent-parameters-of-the-same-type clang-tidy U 4
c27883b843... APFloat.cpp 2 adjacent parameters for 'convertToHexString' of convertible types may be easily swapped by mistake experimental-cppcoreguidelines-avoid-adjacent-parameters-of-the-same-type clang-tidy U 4
APFloat. 2 adjacent parameters for 'convertNormalToHexString' of convertible experimental-cppcoreguidelines-avoid- clang- U 4

Too much reports

- `int max(int left, int right)`
- `string msgEmit(string txt1, string txt2, string txt3)`

“Avoid adjacent unrelated parameters of same type”

"Avoid adjacent unrelated parameters of same type"

… can be invoked … either order … different meaning …

“Avoid adjacent unrelated parameters of same type”

… can be invoked … either order … different meaning …

```
if (a < b) {
    x + y;
}
```

"Avoid adjacent unrelated parameters of same type"

… can be invoked … either order … different meaning …

```
if (a < b) {
    x + y;
}
```

```
foo(1, a);
foo(2, b);
```

“Avoid adjacent unrelated parameters of same type”

… can be invoked … either order … different meaning …

```
if (a < b) {
    x + y;
}
```

```
foo(1, a);
foo(2, b);
```

```
if (!xyz)
    return Default;
return *xyz;
```

"Avoid adjacent unrelated parameters of same type"

… can be invoked … either order … different meaning …

```
if (a < b) {
    x + y;
}
```

```
foo(1, a);
foo(2, b);
```

```
if (!xyz)
    return Default;
return *xyz;
```

```
std::string LName = LHS→Name;
std::string RName = RHS→Name;
if (LName ≥ RName) { ...
```

Make use of names...

```
string concat(string str1, string str2)
```

What if we somehow realised that the order doesn't matter?...

Make use of names...

```
string concat(string str1, string str2)
error format(info errorA, info errorB)
ordering compare(string LHS, string RHS)
matrix rectify(matrix Qmat, matrix Rmat, matrix Tmat)
```

Lang.	Project	Strict (S)	CV ∪ Imp	No-`bool`		Rel				Fil				Rel ∩ Fil			
		Total	*Total*	*- vs S.*		*- vs. S*		*- vs. CV ∪ Imp*		*- vs. S*		*- vs. CV ∪ Imp*		*- vs. S*		*- vs. CV ∪ Imp*	
C	curl	134	228	1	0.75%	61	45.52%	80	35.09%	4	2.99%	4	1.75%	64	47.76%	83	36.40%
	git	1 418	1 644	0		826	58.25%	887	53.95%	142	10.01%	119	7.24%	902	63.61%	953	57.97%
	netdata	227	308	0		123	54.19%	134	43.51%	6	2.64%	5	1.62%	124	54.63%	135	43.83%
	PHP	1 272	1 539	6	0.47%	628	49.37%	699	45.42%	165	12.97%	164	10.66%	697	54.80%	766	49.77%
	Postgres	2 696	3 820	157	5.82%	1 449	53.75%	1 730	45.29%	336	12.46%	272	7.12%	1 572	58.31%	1 820	47.64%
	Redis	393	484	0		226	57.51%	251	51.86%	51	12.98%	41	8.47%	238	60.56%	261	53.93%
	TMux	248	306	0		138	55.65%	141	46.08%	81	32.66%	77	25.16%	169	68.15%	170	55.56%
C++	Bitcoin	394	521	28	7.11%	264	67.01%	318	61.04%	29	7.36%	27	5.18%	274	69.54%	328	62.96%
	CodeCompass	27	28	0		12	44.44%	12	42.86%	0		0		12	44.44%	12	42.86%
	guetzli	72	81	0		44	61.11%	45	55.56%	19	26.39%	17	20.99%	47	65.28%	48	59.26%
	LLVM-CTE	6 109	6 898	577	9.45%	4 055	66.38%	4 294	62.25%	818	13.39%	801	11.61%	4 160	68.10%	4 399	63.77%
	OpenCV	3 286	3 714	48	1.46%	2 320	70.60%	2 462	66.29%	533	16.22%	418	11.25%	2 415	73.49%	2 527	68.04%
	ProtoBuf	313	392	21	6.71%	193	61.66%	214	54.59%	57	18.21%	57	14.54%	201	64.22%	222	56.63%
	Tesseract	793	880	26	3.28%	461	58.13%	461	52.39%	106	13.37%	96	10.91%	493	62.17%	490	55.68%
	Xerces	446	532	23	5.16%	303	67.94%	349	65.60%	66	14.80%	64	12.03%	319	71.52%	365	68.61%
	Z3	2 600	2 801	122	4.69%	2 022	77.77%	2 091	74.65%	861	33.12%	841	30.02%	2 134	82.08%	2 194	78.33%

References I

- andreser. *`_subborrow_u64 argument order inconsistent`*. accessed 2019-12-16. Free Software Foundation - GNU GCC. 2017. URL: http://gcc.gnu.org/bugzilla/show_bug.cgi?id=81294.
- Mark Capella. *Suspicious code with probably reversed parms in call to `IsSingleLineTextControl(bool, uint32_t)`*. accessed 2019-11-23. Mozilla. 2016. URL: http://bugzilla.mozilla.org/show_bug.cgi?id=1253534.
- Ericsson AB. *CodeChecker – an analyzer tooling, defect database and viewer*. accessed 2020-09-18. 2014. URL: http://github.com/Ericsson/CodeChecker.
- GrammaTech. *SwapD*. (accessed 2021-04-09). 2019. URL: http://github.com/GrammaTech/swap-detector.

References II

- Hui Liu et al. “Nomen est Omen: Exploring and Exploiting Similarities between Argument and Parameter Names”. In: *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. May 2016, pp. 1063–1073. DOI: 10.1145/2884781.2884841. URL: http://ieeexplore.ieee.org/document/7886980.
- Michael Pradel and Thomas R. Gross. “Name-Based Analysis of Equally Typed Method Arguments”. In: *IEEE Transactions on Software Engineering* 39.8 (Aug. 2013), pp. 1127–1143. ISSN: 2326-3881. DOI: 10.1109/TSE.2013.7. URL: http://ieeexplore.ieee.org/document/6419711.

References III

Andrew Rice et al. “Detecting Argument Selection Defects”. In: *Proceedings of the ACM on Programming Languages* 1.OOPSLA (Oct. 2017), 104:1–104:22. ISSN: 2475-1421. DOI: 10.1145/3133928. URL: http://doi.acm.org/10.1145/3133928.

Roger Scott et al. “Out of Sight, Out of Place: Detecting and Assessing Swapped Arguments”. In: *2020 IEEE 20th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. Sept. 2020, pp. 227–237. DOI: 10.1109/SCAM51674.2020.00031. URL: http://ieeexplore.ieee.org/document/9252035.

Ray Smith, Google, Inc., et al. *Tesseract OCR Engine*. version 4.1.0 (5280bbc), accessed 2019-12-30. 2006-. URL: http://github.com/tesseract-ocr/tesseract.

References IV

The PostgreSQL Global Development Group. *PostgreSQL*. version 12.1 (578a551), accessed 2019-12-30. 1996-. URL: http://postgresql.org.

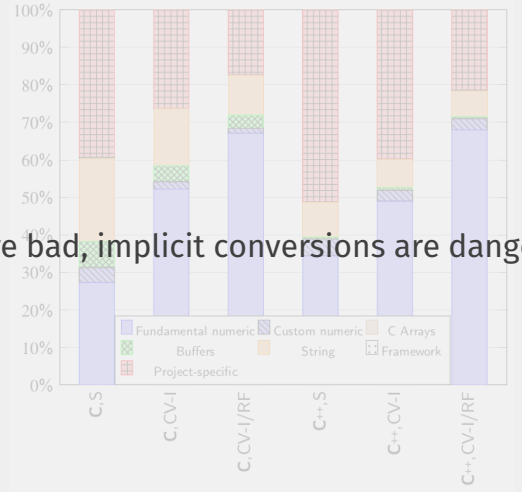
Linus Torvalds et al. *git*. version 2.24.1 (53a06cf), accessed 2019-12-30. 2005-. URL: http://git-scm.org.

János Varjú, Lilla Barancsuk, and Whisperity. *Add readability-suspicious-call-argument check*. accessed 2021-04-13. The LLVM Foundation. 2016. URL: http://reviews.llvm.org/D20689.

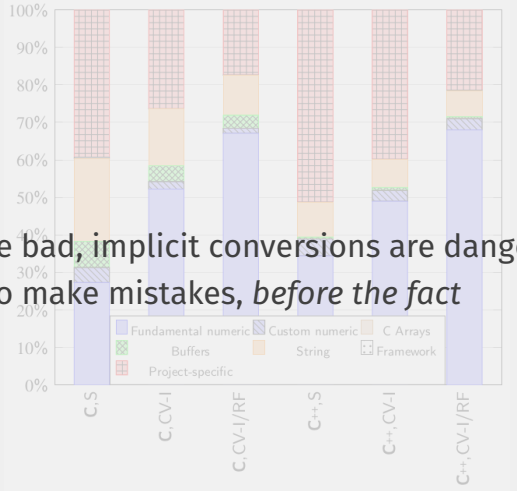
Whisperity. *Add bugprone-easily-swappable-parameters check*. accessed 2021-04-13. The LLVM Foundation. Oct. 2019. URL: http://reviews.llvm.org/D69560.

Conclusion

- Built-in types are bad, implicit conversions are dangerous

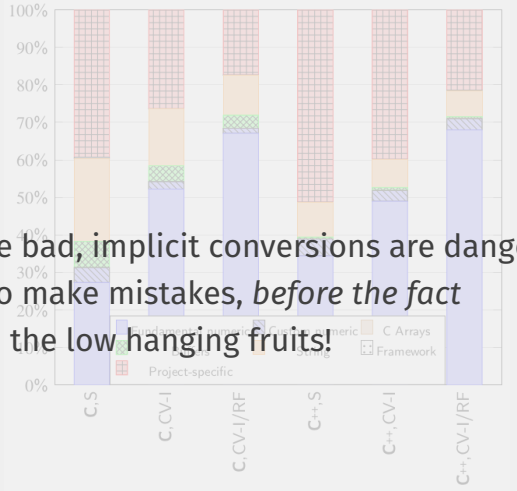


Conclusion



- Built-in types are bad, implicit conversions are dangerous
- Help users not to make mistakes, *before the fact*

Conclusion



- Built-in types are bad, implicit conversions are dangerous
- Help users not to make mistakes, *before the fact*
- Use tools to find the low hanging fruits!