Cobol is the Original Safe Language

James K. Lowden
Cobolworx

Symas Corporation

Brussels, Saturday, 1 February 2023

6:55pm UTC in Room K.3.201

Why is Safety the New Shiny?

What Means “Safe”?

Safety via Compiler

- Runtime, Compile-time

Cobol Vs. C++

Worse is Not Better

Why is Safety the New Shiny

The Fierce Urgency of Security

Old Story, True Story

Cyberattacks have focussed nontechnical minds

Security includes

- Physical security
- People (Kim Philby)
- Policy
- Enforcement
- Software

Language Safety $\Rightarrow$ Security

What Means “Safe”?

No Language Is Safe

```
$ crash() {
    mkdir crash;
    cd crash && crash;
}
$ cd /tmp/
$ crash
Segmentation fault (core dumped)
```

Any language can exhaust resources

Any language can loop infinitely

Interpreted languages defer errors to run time

What Means “Safe”?

Compilers *Could* Help

```
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

void crash() {
   static const char name[][8] = { "crash" };
   mkdir(name[16], 0777);
   chdir(name[16]);
   crash();
}
```

But often don’t (neither gcc nor clang)

```
scan-build clang -c recurse.c
scan-build: Using ’/usr/lib/llvm-14/bin/clang’ for static analysis
scan-build: Analysis run complete.
scan-build: No bugs found.
```

What Means “Safe”?

Runtime Errors, Apparently

> The immediate problem “is” that it’s Too Easy By Default™ to write security and safety vulnerabilities in C++ that would have been caught by stricter enforcement of known rules for *type*, *bounds*, *initialization*, and *lifetime* language safety.
>
> — Herb Sutter https://herbsutter.com/2024/03/11/safety-in-context/

MITRE 2023 Common Weakness Enumeration

Rank	ID	Name	Score
1	CWE-787	Out-of-bounds Write	63.72
4	CWE-416	Use After Free	16.71
7	CWE-125	Out-of-bounds Read	14.60
12	CWE-476	NULL Pointer Dereference	6.59

What Means “Safe”?

COBOL Defines Runtime Exception Policy

IBM COBOL EF (1973) had neither

- dynamic memory
- pointers

ISO COBOL 2023 Defines *Exception Conditions*

- Out-of-bounds Write, Out-of-bounds Read
  - ☞ EC-BOUND, 7 conditions
  - ☞ EC-RANGE, 7 conditions
- Use After Free: *Impossible*
  - ☞ COBOL `FREE` sets pointer to NULL
  - ☞ free before allocate: EC-STORAGE-NOT-ALLOC
- NULL Pointer Dereference
  - ☞ allocation failure: EC-STORAGE-NOT-AVAIL
  - ☞ dereference: EC-DATA-PTR-NULL

Runtime Exceptions

COBOL `Exception Conditions`

120 *Exception Conditions* in 25 categories

- Fatal and non-fatal
- Raised during execution of a statement
- Include I/O, memory, execution, and computation

- ☞ EC-BOUND-REF-MOD, substring
- ☞ EC-DATA-CONVERSION
- ☞ EC-DATA-OVERFLOW, *Exponent overflow during* `MOVE`
- ☞ EC-EXTERNAL-FILE-MISMATCH, wrong file type
- ☞ EC-FUNCTION-NOT-FOUND and EC-PROGRAM-NOT-FOUND, dynamic call
- ☞ EC-SIZE-TRUNCATION, *Significant digits truncated in store*

Runtime Exception Policy

COBOL `DECLARATIVES`

Each program may define *Declaratives*

- procedures to handle *exception conditions*

Handled specifically, or by category

Recovery from *fatal* Exception Condition possible with `RESUME`

COBOL *Conditionals*

Most *statements* have a “conditional form” that defines logic

- to handle exception conditions the statement may raise, or
- to execute when no exception is raised, because
- `RESUME` continues at next statement

COBOL defines which exception conditions each statement can raise

Programmer relieved of enumerating exception conditions per statement

Compile-time Policy

Nail It Down

COBOL defines

- Variables, to exact byte size and precision
- Files by type and record type
- Semantics for
  - Computation and Rounding: `COMPUTE`
  - Memory-to-memory: `MOVE`
    - `MOVE` converts between types
  - File operations, including `OPEN`
  - Character output, `DISPLAY`

No `memset`, `memcpy`, or `stdio`, or `errno`

Compile-time Policy

Rules for Data

RAII

- Each field has a size, and may define an initial value
- A field may have an enumerated valid domain †
- Default blanks for characters and zeroes for numbers.

Numeric types may be

- native integer or floating point (of defined size)
- machine-independent integer or fixed-point

Examples

```
77      A06THREES-DS-03V03      PICTURE S999V999 VALUE 333.333.
77      A08TWOS-DS-02V06        PICTURE S99V9(6) VALUE 22.222222.
77      WRK-XN-00001            PICTURE X.
77      A10ONES-DS-10V00        PICTURE S9(10)
                                VALUE 1111111111.
```

† *Unlike other obsolete features, it is intended that interest in this facility will be reevaluated during the next revision of standard COBOL....*

Compile-time Policy

Rules for Files

COBOL *language* (not library) defines I/O Statements

- `OPEN`
- `START` (seek)
- `READ`
- `WRITE`
- `DELETE` (record)
- `CLOSE`

Invalid file operations disallowed by compiler

- Cannot use random access on sequential file

Network Policy

Rules for Communication

Communication is defined by a superior system, e.g., CICS

COBOL application runs as loaded inferior module

Each I/O operation is defined

- Appearance
- Size
- Structure

Network definition is static

- To add a phone, everyone must hang up

Network devices can respond only to what is sent

Permissions controlled by superior

Module Policy

Macros, not libraries

Preprocessors, not libraries, add functionality to COBOL

EXEC CICS
: make COBOL records from application UI forms

EXEC SQL
: make COBOL records from SQL rows

The *whole application* is under compiler control

A Safe Language

Safety by Definition

Because COBOL defines

- Size, initial value, and valid domain of every variable
- Semantics for `MOVE` and `COMPUTE`
- Exact numeric computation, including rounding
- Records for every I/O operation
  - File
  - Network
- File types

Therefore

- Many operations are disallowed by the compiler
- Many more runtime exceptions can be raised (and handled)

COBOL vs. C++

A scoreboard

Feature	COBOL	C++
RIAA by default	Yes	No
Field domains	Yes	No
Conversion semantics	Yes	No
Computation exceptions	Yes	No
File semantics	Yes	No
I/O record definitions	Yes	No
Declarative exception handling	Yes	No
Runtime Exceptions	120	10

COBOL is (Still) the Safest Language

A Checklist

- ✓ Memory always defined
- ✓ Checked iterators
- ✓ Checked computation
- ✓ I/O always defined
- ✓ Centralized and per-statement exception handling

---

https://herbsutter.com/2024/03/11/safety-in-context/ 2024-03-11, Herb Sutter

https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2024/p3274r0.pdf 2024-04-09, Bjarne Stroustrup

A Big Language is Safe Language

The compiler can't check
what it does not see

COBOL defines the *whole* application

- System interface regulated by compiler

"Modern" languages limit compiler's role

- System interface regulated by programmer
- Function calls limit static analysis feasibility

Maybe the next modern language

- Needs to be less modern