What About C/C++

July 24, 2025

By Aaron Grothe

Introduction

If anybody has any questions or comments at any time please let me know.

If I start to mumble please let me know as well :-)

Slides are on my website https://www.grothe.us

Questions are welcome anytime.

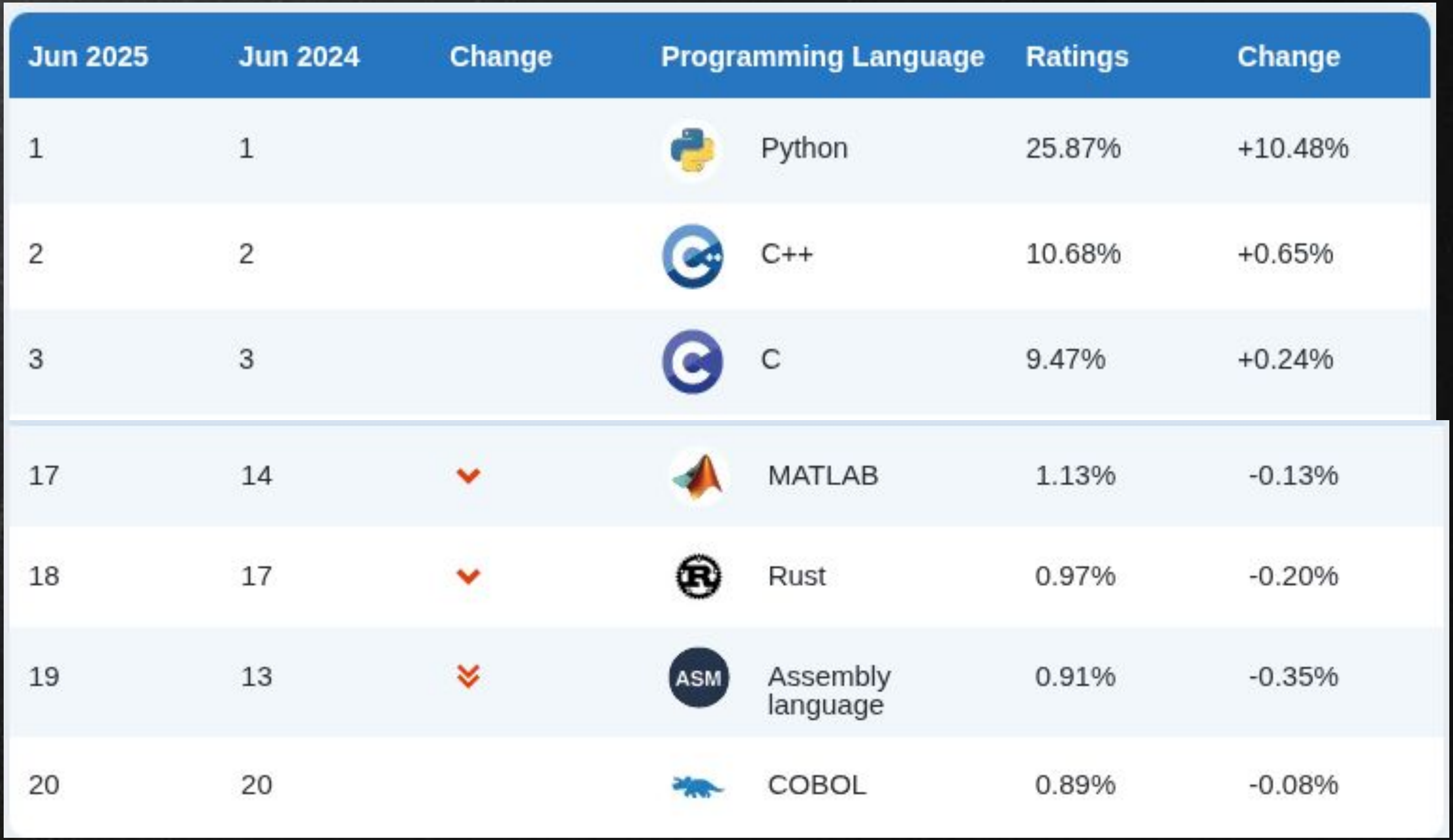
How to Fix C/C++

Rewrite it all in Rust

Thank You.

Goodnight.

TIOBE Report for C/C++/Rust



Jun 2025	Jun 2024	Change	Programming Language	Ratings	Change
1	1		Python	25.87%	+10.48%
2	2		C++	10.68%	+0.65%
3	3		C	9.47%	+0.24%
17	14	⌄	MATLAB	1.13%	-0.13%
18	17	⌄	Rust	0.97%	-0.20%
19	13	⌄⌄	Assembly language	0.91%	-0.35%
20	20		COBOL	0.89%	-0.08%

What does This Mean?

The TIOBE Programming Community Index is a popular measure calculating the popularity of programming languages via references in search results

- C++/C are in the number 2, 3 slots
- Rust is in the Number 18 slot, down one from June 2024

TIOBE is just one measure, there are a lot of others as well.

Larger C/C++ Code Bases

- Google's Codebase is reported to be over 2 billion lines of code.
- Windows 10 is estimated to be around 50 million lines of code
  - Majority of the first two are written in C/C++
- Microsoft Office is estimated to be over 100 million lines of C++ code
- C++ codebase for Unreal Engine 4.23.1 is estimated to be around 16 million lines
- C++ codebase for Adobe Photoshop is estimated to be around 16 million lines of code.

Larger C/C++ Code Bases

- The Linux Kernel as of early 2025 is around 40 million lines of code
  - C - 35 million lines
  - C++ - around 600,000 lines in utilities/drivers
  - Assembly - Couple of hundred thousand lines
  - Rust - several thousand lines for Kernel Drivers.

Larger Rust Code Bases

- Mozilla Firefox primarily in C++ and Javascript
    - Approximately 3.8 to 4.9 million lines of Rust code in the Firefox codebase
- Rust Compiler (rustc)
    - Approximately 1.7 million lines of Rust Code
- Redox OS
    - Up to 1 million lines of Rust code when you include kernel and all support systems (display driver, programs, etc)
- Various important utilities such as gnu coreutils and sudo are being rewritten in rust as well.

Most Likely Future Path for C++

Several Potential Ways Future could go for C++

- Worst Case: Cobol like future, current code base survives limited future code
- Most Likely Case: Java like future, considered core part of software development
- Unlikely Case: Rust conversion tools work well and C++ largely disappears
- Best Case: C++ evolves and remains relevant, C++ Profiles, Safe C++, steals features from other languages :-)

We'll see how it goes over the next 5-10 years :-).

Why Do we Need a Change?

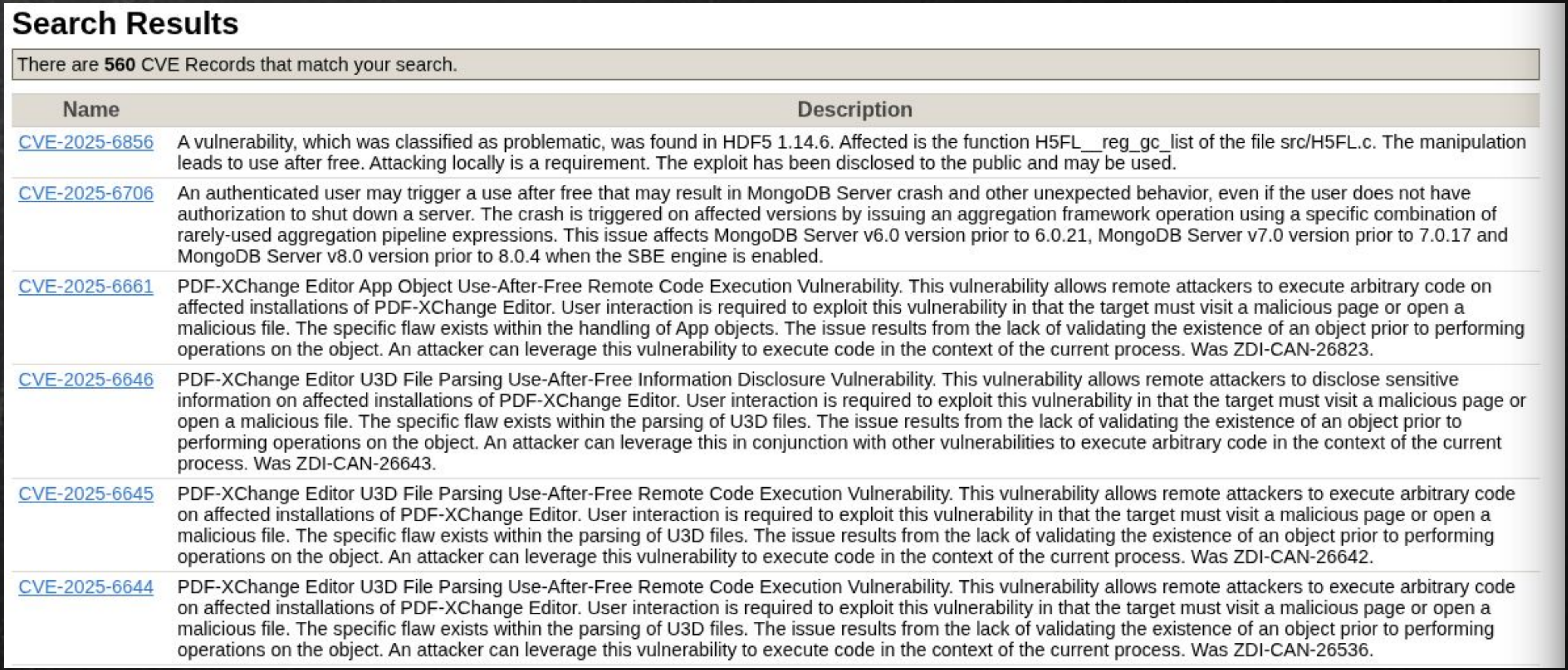
According to Microsoft **70% of the vulnerabilities** Microsoft assigns a CVE each year continue to memory related - https://msrc.microsoft.com/blog/2019/07/a-proactive-approach-to-more-secure-code/

Quick search of cve.mitre.org for the following terms "use after free c++ 2025"

https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=use+after+free+c\+\++2025 .

Why Do we Need a Change?

Returns 560 Results for This class



Search Results

There are **560** CVE Records that match your search.

Name	Description
CVE-2025-6856	A vulnerability, which was classified as problematic, was found in HDF5 1.14.6. Affected is the function H5FL__reg_gc_list of the file src/H5FL.c. The manipulation leads to use after free. Attacking locally is a requirement. The exploit has been disclosed to the public and may be used.
CVE-2025-6706	An authenticated user may trigger a use after free that may result in MongoDB Server crash and other unexpected behavior, even if the user does not have authorization to shut down a server. The crash is triggered on affected versions by issuing an aggregation framework operation using a specific combination of rarely-used aggregation pipeline expressions. This issue affects MongoDB Server v6.0 version prior to 6.0.21, MongoDB Server v7.0 version prior to 7.0.17 and MongoDB Server v8.0 version prior to 8.0.4 when the SBE engine is enabled.
CVE-2025-6661	PDF-XChange Editor App Object Use-After-Free Remote Code Execution Vulnerability. This vulnerability allows remote attackers to execute arbitrary code on affected installations of PDF-XChange Editor. User interaction is required to exploit this vulnerability in that the target must visit a malicious page or open a malicious file. The specific flaw exists within the handling of App objects. The issue results from the lack of validating the existence of an object prior to performing operations on the object. An attacker can leverage this vulnerability to execute code in the context of the current process. Was ZDI-CAN-26823.
CVE-2025-6646	PDF-XChange Editor U3D File Parsing Use-After-Free Information Disclosure Vulnerability. This vulnerability allows remote attackers to disclose sensitive information on affected installations of PDF-XChange Editor. User interaction is required to exploit this vulnerability in that the target must visit a malicious page or open a malicious file. The specific flaw exists within the parsing of U3D files. The issue results from the lack of validating the existence of an object prior to performing operations on the object. An attacker can leverage this in conjunction with other vulnerabilities to execute arbitrary code in the context of the current process. Was ZDI-CAN-26643.
CVE-2025-6645	PDF-XChange Editor U3D File Parsing Use-After-Free Remote Code Execution Vulnerability. This vulnerability allows remote attackers to execute arbitrary code on affected installations of PDF-XChange Editor. User interaction is required to exploit this vulnerability in that the target must visit a malicious page or open a malicious file. The specific flaw exists within the parsing of U3D files. The issue results from the lack of validating the existence of an object prior to performing operations on the object. An attacker can leverage this vulnerability to execute code in the context of the current process. Was ZDI-CAN-26642.
CVE-2025-6644	PDF-XChange Editor U3D File Parsing Use-After-Free Remote Code Execution Vulnerability. This vulnerability allows remote attackers to execute arbitrary code on affected installations of PDF-XChange Editor. User interaction is required to exploit this vulnerability in that the target must visit a malicious page or open a malicious file. The specific flaw exists within the parsing of U3D files. The issue results from the lack of validating the existence of an object prior to performing operations on the object. An attacker can leverage this vulnerability to execute code in the context of the current process. Was ZDI-CAN-26536.

Why Do we Need a Change?

Results for cve.mitre.org search

883:	Linux C 2025
834:	Buffer Overflow 2025 c++
560:	Use After Free 2025 c++
21:	Double Free 2025 c++

Just a couple of examples.

So what are the Options?

- Boehm Garbage Collection
- Write Better Code
- C/C++ Variants
- C++ to Rust Conversion tools.

Boehm Garbage Collection

Boehm–Demers–Weiser garbage collector

Boehm Garbage Collector is a library that does conservative garbage collection, for C/C++ programs

It is used by the following programs

Inkscape, Cordon high performance Python compiler, GNU Compiler for Java, GNU Guile and others.

Boehm Garbage Collection

Make the following Changes to your source code

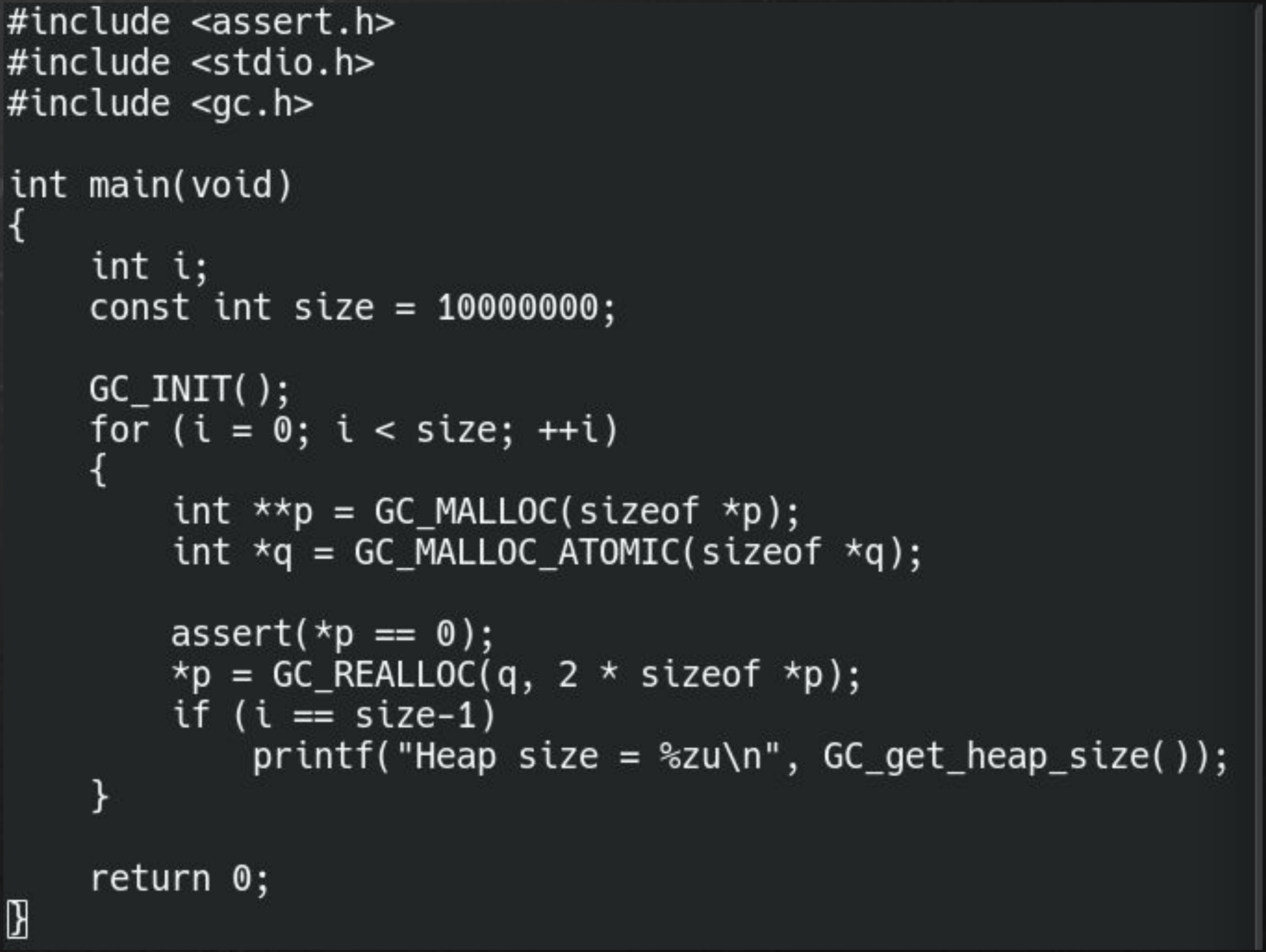
#include <gc.h>

Replace calls malloc, new, realloc with GC_MALLOC, GC_NEW, and GC_REALLOC

Remove delete and free calls :-)

Link the program with -lgc library.

Boehm Garbage Collection (Example)



```c
#include <assert.h>
#include <stdio.h>
#include <gc.h>

int main(void)
{
    int i;
    const int size = 10000000;

    GC_INIT();
    for (i = 0; i < size; ++i)
    {
        int **p = GC_MALLOC(sizeof *p);
        int *q = GC_MALLOC_ATOMIC(sizeof *q);

        assert(*p == 0);
        *p = GC_REALLOC(q, 2 * sizeof *p);
        if (i == size-1)
            printf("Heap size = %zu\n", GC_get_heap_size());
    }

    return 0;
}
```

Boehm Garbage Collection

Cons

- Is a conservative garbage collector, can have some leaks
- Overhead of GC running (you can control it though)

Pros

- Can be used as a debugging library as well for detecting leaks
- Has been around for 20+ years and is used by some large programs
- Can be combined with stack smashing protection.

Boehm Garbage Collection

Is Boehm a total solution?

No. It is part of a potential solution in depth. Would be curious to see Chromium/Firefox rebuilt with Boehm GC and see how it compares to regular Chromium/Firefox.

Write Better Code

C++ has Tools to help you prevent writing memory leaks

- Embrace Smart Pointers
- Avoid Raw new and delete
- Use Standard Library Containers
- RAII - Resource Acquisition Is Initialization
- Static Analysis Tools and Memory Profilers.

Embrace Smart Pointers (C++ 11 onwards)

std::unique_ptr

- Exclusive pointer to a resource
- Automatically frees resource when leaving scope
- Even works for exceptions

std::shared_ptr

- Allows multiple pointers to a resource
- Free resource when last pointer leaves scope
- Can result in leaks with circular references.

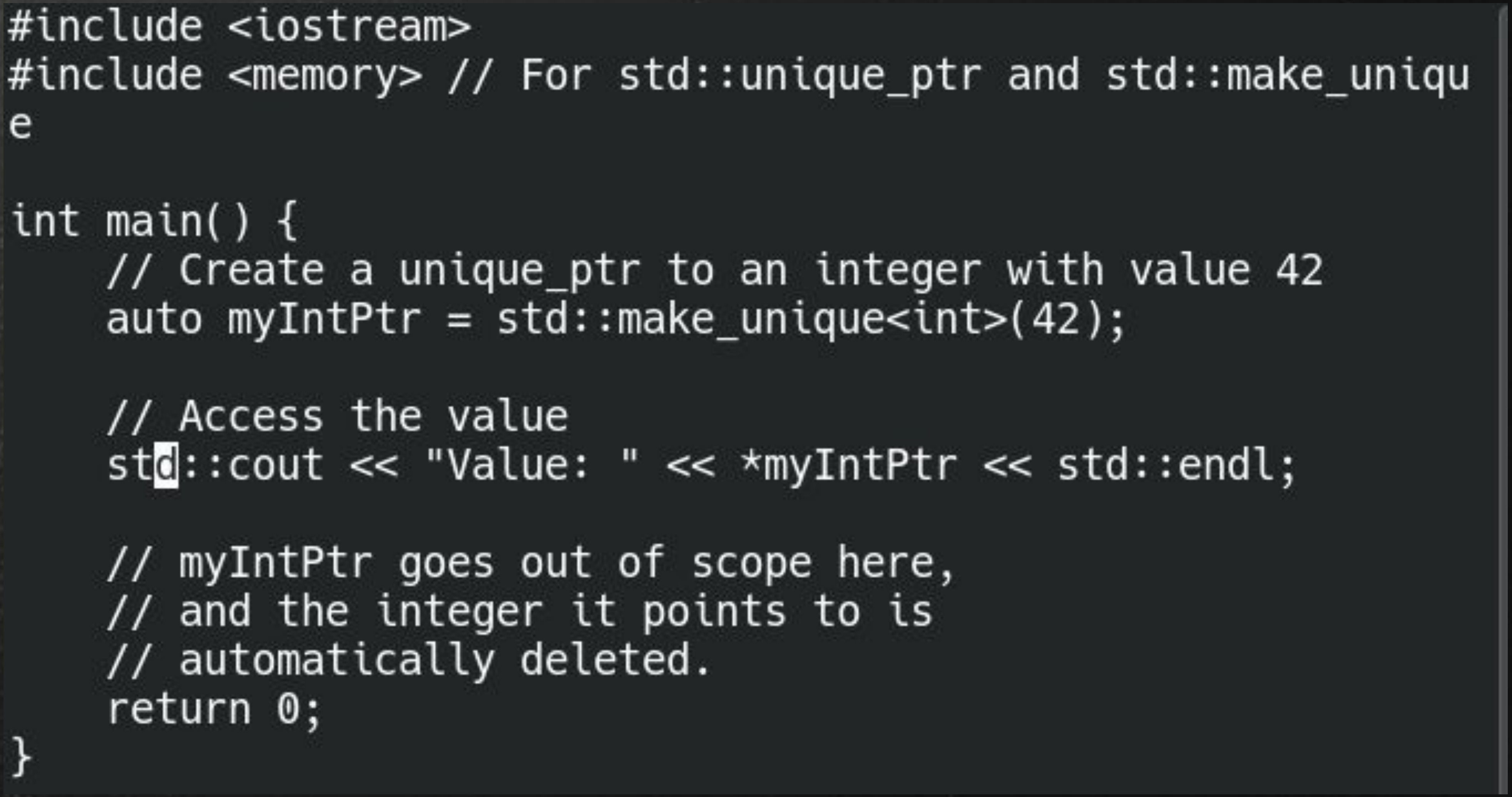
Embrace Smart Pointers (C++ 11 onwards)

std::weak_ptr

- Use to break circular references between objects

Hopefully, you'll mostly use unique_ptr and shared_ptr .

A std:unique_ptr example



```
#include <iostream>
#include <memory> // For std::unique_ptr and std::make_uniqu
e

int main() {
    // Create a unique_ptr to an integer with value 42
    auto myIntPtr = std::make_unique<int>(42);

    // Access the value
    std::cout << "Value: " << *myIntPtr << std::endl;

    // myIntPtr goes out of scope here,
    // and the integer it points to is
    // automatically deleted.
    return 0;
}
```

Embrace Smart Pointers

Cons

- Reference counting, small overhead can matter in performance-critical situations
- Size overhead - typically twice size of raw pointers
- Learning curve

Pros

- Automatic memory management
- Reduces risk of dangling pointers and double deletions.

Avoid Raw new and delete

Everytime you write a new you should be writing a delete.

Instead of

```
 MyClass* obj = new MyClass();
// ... potentially complex code or exception here ...
delete obj; // Might not be reached!
```

How about

```
std::unique_ptr<MyClass> obj =
std::make_unique<MyClass>();
// ... complex code, no worries about deallocation ...
```

Avoid Raw new and delete

Or better yet, don't do a new at all

Instead of

Player *p = new Player();

How about

Player p;.

Avoid Raw new and delete

Cons

- Reworking of code
- Can introduce bugs if you don't take care
- Trying to put stuff on the stack instead of heap can cause issues with legacy code

Pros

- Done correctly can significantly reduce the number of new/delete calls in code.

Use Standard Library Containers

Standard Library Containers std::vector, std::string, std::map, etc. all automatically manage the memory they use.

std::vector<MyClass> vec(10); // Automatically manages MyClass objects .

Use Standard Library Containers

Cons

- Can result in memory fragmentation, frequent insertions/deletions can cause memory fragmentation of heap
- Additional overhead of STL containers, usually minor

Pros

- Reduces memory leaks
- Exception safety (guaranteed cleanup)
- Well tested/developed code .

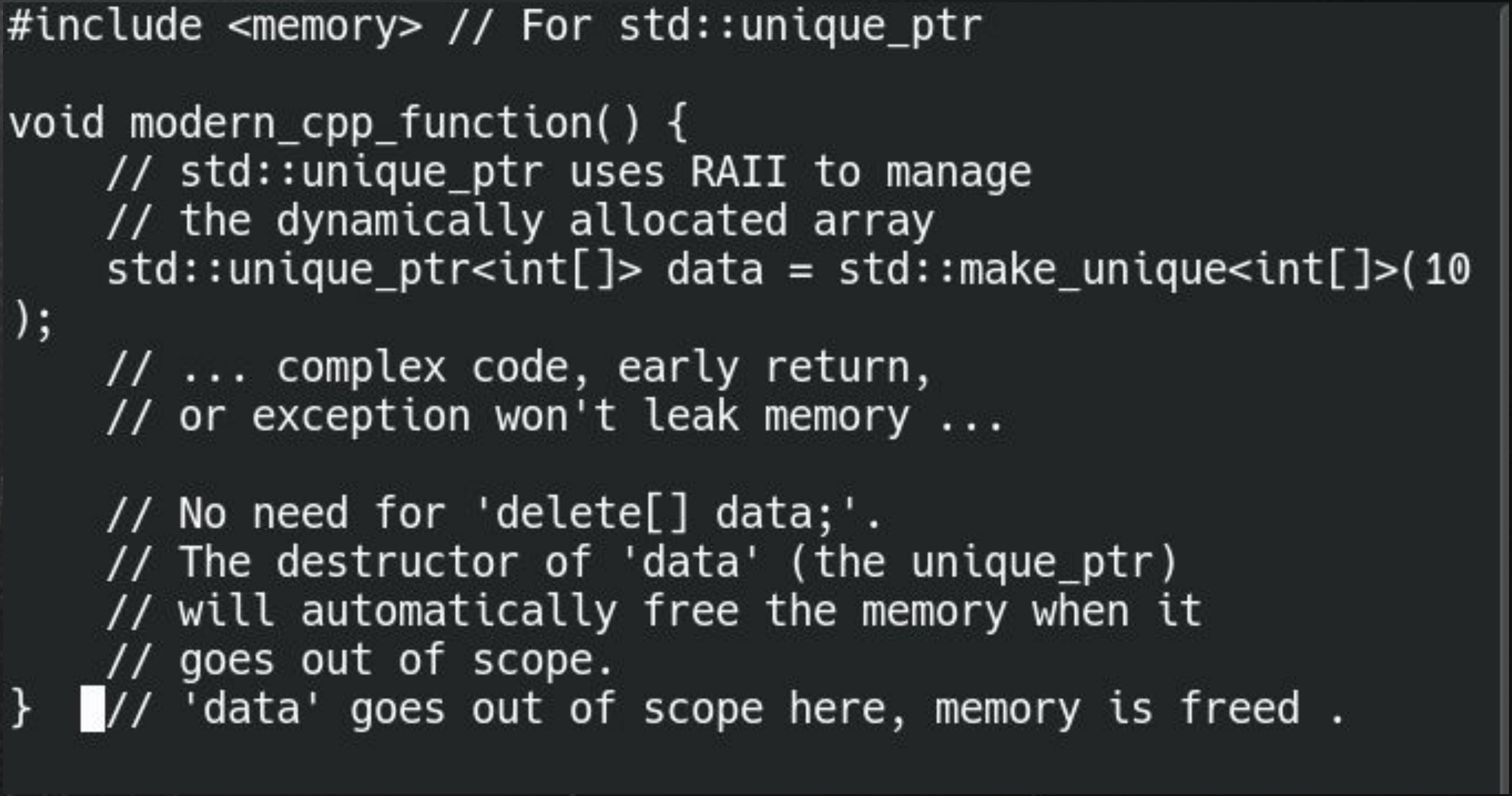
RAII - Resource Acquisition is Initialization

Resource Acquisition - is done in the constructor
Resource Initialization - is initialized with acquired resource
Resource Release - handled in destructor

Ideally acquisition is done with managed pointers: e.g. std:unique_ptr or standard template classes e.g. std::vector

Shifts responsibility from programmer to explicitly handle the free to the system automatically handling it .

RAII - Example



```
#include <memory> // For std::unique_ptr

void modern_cpp_function() {
    // std::unique_ptr uses RAII to manage
    // the dynamically allocated array
    std::unique_ptr<int[]> data = std::make_unique<int[]>(10
);
    // ... complex code, early return,
    // or exception won't leak memory ...

    // No need for 'delete[] data;'.
    // The destructor of 'data' (the unique_ptr)
    // will automatically free the memory when it
    // goes out of scope.
}   // 'data' goes out of scope here, memory is freed .
```

RAII - Example

Cons

- Minor overhead
- Same as they are for shared pointers
- Learning curve

Pros

- RAII handles: memory, file handles, network resources, locks, etc.
- Composability: Objects can contain multiple objects that are automatically cleaned up .

Static Analysis Tools and Memory Profilers

These are tools that can help you improve your code

Static Analyzers: tools such as clang-tidy, smatch, sonarqube, lint and other tools that can analyze source code

Compiler Warnings: -Wall, -Wextra, additional compiler flags to generate better warnings, prevent bad behaviours

Memory profilers/leak detectors: Boehm in detection mode, Valgrind, Purify, Leaksanitizer, Addresssanitizer, etc.

C/C++ Variants

We'll look at a couple of C/C++ variants that attempt to solve the memory issues

- Cyclone
- Trap-C
- Fil-C
- Cpp2
- carbon
- Mini-c / convert to rust
- C++ Profiles
- Safe C++ .

C/C++ Variants

Common concepts

- Most are actually a superset of c++ that implements a safe subset of c++
- Reduce/restrict pointer logic
- Removes goto, unions
- Implement garbage collection or reference counting system for objects .

C/C++ Variants - TrapC

Developed by Robin Rowe

- Is actually an extension to C with C++-like features added: objects, classes, and so on
- ABI (Application Binary Interface) compatibility, so you can combine C++ and Trapc
- No unsafe keyword so no option to turn off memory safety (unlike rust) .

C/C++ Variants - Fil-C

Developed by Filip Pizlo, senior director of language engineering at Epic Games

- Aims to be 100% compatible with C++
- Overhead tends to be 1.5x to 5x times slower than regular code, aiming for 1.2x overhead
- C or C++ code may behave differently in FilC environment
- Not working towards ABI compatibility, so you can not combine a C++ and FilC library necessarily
- Has been able to compile Cpython
- Only targeting Linux/x64 currently .

C/C++ Variants - Cpp2

Developed by Herb Sutter, C++ Expert

- Cpp2 is a compiler for experimental features that may make it into a future C++ release
- It is designed to take the code convert it into standard C++ code and then compile it. Similar to cfront which compiled C++ code into C in the early C++ days
- New memory safety features are being trialed in this software base .

C/C++ Variants - Carbon

Developed by Chandler Carruth, Google

- Carbon is being developed as a potential successor to C++
- Is in an experimental phase
- Enforces initialization of variables
- Automatic memory management (in-development).

C/C++ Variants - Mini-C

Developed by France's Inria and Microsoft

- Subset of C code that can be converted into Rust code automatically
- Removes dangerous features such as pointer arithmetic
- Has been able to convert HACL (High-Assurance Crypto Library) An 80,000 line C project to Rust code.

C/C++ Variants - C++ Profiles

Developed by Several people including Bjarne Stroustrup and Herb Sutter

- Subset of C++ with new syntax/semantics for safety
- ISO C++ Committee appears to be choosing C++ profiles as way forward
- Goal is that the majority of regular C++ code can be moved to it gradually over time
- Probably the simplest changes to C++ to get memory safety
- Relies on compiler/runtime for most of its enforcement.

C/C++ Variants - Safe C++

Developed by Several people

- Superset of C++ with new syntax/semantics for safety
- Adds new keywords such as safe/unsafe
- Goal is that the majority of regular C++ code can be moved to it gradually over time
- Will have unsafe specifiers to allow gradual migration to type safety
- Lifts the Borrow checking syntax from Rust.

C/C++ Variants - Potential Issues

Cons

- C++ is an ISO standard language, adding features can take years
- Most of the variants will not gain market share, potentially leaving developers on isolated islands
- C++ efforts will be divided until standards evolve

Pros

- One of these might be the way forward for C++.

C/C++ Variants - Potential Issues

Which ones are worth consideration?

- Right now I'm thinking C++ Profiles is the most likely winner
  - Has a good chunk of the C++ ISO committee supporting it
  - Herb Sutter and the cpp2 front end are supporting it
  - Is less ambitious than alternatives such as Safe C++.

C++ to Rust Conversion tools

Approaches being evaluated

- Automated transpilers
- Mixing C++ and Rust
- Manual Migration Strategies.

Automated Transpilers

These tools attempt to convert C/C++ code into Rust code

- CRUST - C to Rust Transpiler
- Corrode - Converts C and some C++ to Rust
- TRACTOR - TRanslating All C to Rust - spec, no code yet.

Automated Transpilers

Cons

- Resulting code will need major additional work
- Usually take defaults like unsafe mode, and let programmers convert the code to safe mode
- Tools are still evolving and need to see what comes out of projects like TRACTOR

Pros

- Can do a lot of the translation without user intervention
- Provides a starting place for the conversion.

Mixing C++ and Rust

These tools are designed to allow you to combine C++ and Rust code

- Bindgen - generates Rust FFI (Foreign Function Interface) bindings from C++ header files
- Cbindgen - generates C and C++ header files from Rust code
- CXX - C++/Rust FFI Framework
- Manual C++/Rust wrapper functions.

Mixing C++ and Rust

Cons

- Requires knowledge of both C++ and Rust
- Rust and C++ hand offs might not always be smooth

Pros

- Allow orderly migration from C++ to Rust
- Linux kernel will be large test case for this
  - Bindgen being used to generate FFI bindings.

Manual Migration Strategies

This is the process of manually migrating programs from C/C++ to Rust

- Several approaches
  - Go back to the initial specs for the program and fully rewrite in Rust, using C/C++ version for comparison
  - Start rewriting core components in Rust over time potentially combining with C/C++ code
  - Manually convert code line by line from C/C++ to Rust.

Manual Migration Strategies

Cons

- Time/Costs can be difficult to estimate
- High Risk/High Reward

Pros

- Most flexible approach - you can do anything
- Keeps programmers employed :-) .

Where are we at?

C++ is at an interesting point. The U.S. Government and many large companies are suggesting that we quit using it.

A lot of bad code has been written in C++ and every other language over the years.

A lot of C++ code will be tough to get rid of for years.

C++ is not Latin and it will evolve, it will grab features from other languages in a borg-like assimilation and keep trying to survive .

Where are we going?

Summary

Will be interesting to see how next 12-18 months go

- C++ profiles and Safe C++ will continue to evolve
- Other options TrapC, Fil-C will continue to get better
- CRUST, Corrode, and Tractor will be tested/evaluated
- Cosmic, Redox will both show Rust at scale
- Oxidier, SudoRS will be shipped with Ubuntu 25.10
- Linux rust driver support will show what it takes to integrate Rust into a large legacy code base .

Advice For Today

- Use the tools available Valgrind, AddressSantitizer, LeakSanitizer, Compiler options, etc. to write better code
- Follow future developments of C++ Profiles/CPP2, and Safe C++ working groups
- Consider using Boehm GC library if that meets your needs

Don't panic, Write better code, and carry on :-) .

Thanks and Q&A

That’s all I’ve got for today.

Thanks for Listening.

Any Questions??? .