Undefined Behavior
Back To Basics

Barbara Geller & Ansel Sermersheim
CppCon - October 2021

Introduction

- Prologue
- Overview
- Building a definition for Undefined Behavior
- Definitions from the C++ Standard
- Partial List of Common C++ Undefined Behavior
- Undefined Behavior is Not an Error
- When is Undefined Behavior Acceptable
- Compiler Options
- Resolving Undefined Behavior
- Examples

Prologue

- Co-Founders of the following projects
  - CopperSpice
    - cross platform C++ GUI libraries
  - DoxyPress
    - documentation generator for C++ and other languages
  - CsString
    - support for UTF-8 and UTF-16, extensible to other encodings
  - CsSignal
    - thread aware signal / slot library
  - CsLibGuarded
    - library for managing access to data shared between threads

Prologue

- Credentials
  - every library and application is open source
  - projects are developed using cutting edge C++ technology
  - all source code hosted on github
  - prebuilt binaries available on our download site
  - documentation is generated by DoxyPress

  - youtube channel with videos focused mostly on C++
  - speakers at multiple conferences
    - CppCon, CppNow, emBO++, MeetingC++, code::dive
  - numerous presentations for C++ user groups
    - United States, Germany, Netherlands, England

Undefined Behavior

- Overview
  - misconceptions about undefined behavior
    - undefined behavior will be found in a code review
    - debugging undefined behavior just takes a bit of practice
    - good testing will catch undefined behavior

    - they are working on get rid of undefined behavior in C++
    - better compilers will report undefined behavior as an error
    - experienced developers never have the bad undefined behavior

  - what the standard says about undefined behavior
    - if your program has undefined behavior, it is not correct

Undefined Behavior

- Overview
  - compiler developer
    - objective is to leverage every opportunity to optimize
    - undefined behavior is a fun theoretical discussion
    - understanding every aspect of undefined behavior is essential
    - overlooking undefined behavior can impact performance
  - application developer
    - objective is to write code which has zero undefined behavior
    - undefined behavior can be a daunting, intimidating discussion
    - understanding how to avoid undefined behavior is mandatory
    - ignoring undefined behavior is dangerous

Undefined Behavior

- Example 1
    - detailed description
        - declares a vector of strings
        - assigns values using an initializer list
        - names.size() will return 5

    - is there any undefined behavior?

```
std::vector<std::string> names;
names = { “tiger”, “horse”, “ostrich”, “gerenuk”, “jodankee” };
```

Undefined Behavior

- Example 1
  - Webster’s dictionary
    - one of most respected standards for American English
    - about 470,000 entries, around 1000 are added each year
  - according to the Webster’s standard “jodankee” . . .
    - is not a valid word in the dictionary
    - has no meaning and there is no correct pronunciation
  - how accurate are these statements
    - Feeding a jodankee too much chocolate is not harmful
    - My jodankee connects over both USB 3 and WiFi

Undefined Behavior

- Building a definition for Undefined Behavior
  - Webster's dictionary : undefined
    - *not clearly or precisely shown, described, or limited*
  - Webster's dictionary : behavior
    - *the way in which something functions or operates*
  - best practices
    - when writing a story it is customary for the words to be real
    - authors break this rule frequently
    - readers can typically reason though the meaning
    - A Wookie, a Klingon, and a Hobbit, walk into a bar . . .

Undefined Behavior

- Building a definition for Undefined Behavior
  - our interpretation of the phrase "behavior which is undefined"
    - *something which does not function as described*

  - from this definition would you consider these undefined behavior
    - can you play a guitar with missing strings
    - will a keyboard operate as a monitor

  - how many non-words exist
    - with 26 letters in English there are a lot of combinations
    - it would be impossible to list every word which is missing from the Webster's dictionary

Undefined Behavior

- Example 2
  - what happens if you read past the end of an `std::vector<T>`
    - read operation could return a perfectly valid T
    - or it could return a value which is not a T
    - program may crash at runtime
    - read could be optimized out by the compiler

Undefined Behavior

- Example 2
  - what happens if you read past the end of an std::vector<T>
    - read operation could return a perfectly valid T
    - or it could return a value which is not a T
    - program may crash at runtime
    - read could be optimized out by the compiler

  - according to the C++ standard
    - reading past the end of std::vector is undefined behavior

Undefined Behavior

- Definitions from the C++ Standard
  - defined behavior
    - code which has a clear or precise meaning
      - `int sum = 17 + 8;`
      - `printf("Welcome to CppCon 2021");`
      - `auto [first, second] = getPair();`
  - implementation defined behavior
    - code which can have multiple meanings
    - compiler must consistently pick one and document the choice
      - `if ( sizeof(int) < sizeof(long) ) {  }`

Undefined Behavior

- Definitions from the C++ Standard
    - unspecified behavior
        - code which could have multiple meanings
        - compiler is allowed to choose one at random
            - comparing string literals
                - `if ("abc" == "abc") {  }`
    - undefined behavior
        - code which has no meaning
            - invoking the destructor of an object twice
            - doing a bit shift by a negative value
            - converting a double to a float when the value is too large

Undefined Behavior

- Example 3
    - does the following code compile

```
int * varA = nullptr;                  // line 1
*varA = 17;                            // line 2

int varB;                              // line 3
varA = &varB;                          // line 4

std::cout << *varA;                    // line 5
std::cout << varB;                     // line 6
```

Undefined Behavior

- Example 3
  - does the following code compile
    - (line 2) dereferencing a null pointer is UB

```
int * varA = nullptr;                 // line 1
*varA = 17;                           // line 2

int varB;                             // line 3
varA = &varB;                         // line 4

std::cout << *varA;                   // line 5
std::cout << varB;                    // line 6
```

Undefined Behavior

- Example 3
    - does the following code compile
        - (line 2) dereferencing a null pointer is UB
        - (line 5) accessing an uninitialized variable is UB

```
int * varA = nullptr;                   // line 1
*varA = 17;                             // line 2

int varB;                               // line 3
varA = &varB;                           // line 4, address of varB is valid

std::cout << *varA;                     // line 5, dereference is valid
std::cout << varB;                      // line 6
```

Undefined Behavior

- Example 3
  - does the following code compile
    - (line 2) dereferencing a null pointer is UB
    - (line 5) accessing an uninitialized variable is UB
    - (line 6) accessing an uninitialized variable is UB

```
int * varA = nullptr;              // line 1
*varA = 17;                        // line 2

int varB;                          // line 3
varA = &varB;                      // line 4, address of varB is valid

std::cout << *varA;                // line 5, dereference is valid
std::cout << varB;                 // line 6
```

Undefined Behavior

- Awkward Syntax in C++
  - does this function have undefined behavior

```
template <typename T1, typename T2>
void doLessThanLessThan(T1 &x, T2 &y)
{
  x << y;
}
```

Undefined Behavior

- Awkward Syntax in C++
    - does this function have undefined behavior

```
template <typename T1, typename T2>
void doLessThanLessThan(T1 &x, T2 &y)
{
  x << y;
}


doLessThanLessThan(250, 75);               // bit shift, undefined behavior

doLessThanLessThan(std::cout, “cat”);  // write to standard out
```

Undefined Behavior

- How is Undefined Behavior Defined in C++
  - result of attempting to execute source code whose behavior is not defined in the C++ standard
  - responsibility of the programmer to write code which never causes undefined behavior
  - a correct program will operate as written
    - only if the code is free of undefined behavior
  - guarantees made by the C++ standard
    - none, if you have any undefined behavior

Undefined Behavior

- Partial list of common C++ Undefined Behavior
  - access to an element of an std::vector beyond the end
  - de-reference of a null pointer
  - use of an uninitialized variable
  - calling a pure virtual function from a constructor or destructor
  - use of an object after it has been destroyed (use after free)
  - casting a pointer to an incompatible type and then using the result
  - infinite loop without side effects
  - modifying a string literal or any other const object
  - failing to return a value from a value-returning function
  - any race condition
  - integer divide by zero
  - signed integer overflow

Undefined Behavior

- Example 4
    - signed integer arithmetic
        - if the result is beyond the range of representable values then “signed integer overflow” occurs and is undefined behavior
    - unsigned integer arithmetic
        - according to the standard, this operation never overflows and is defined behavior

```
int volume( int length )
{
  return length * length * length;
}
```

Undefined Behavior

- Undefined Behavior is Not an Error
    - no overlap between undefined behavior and an error
    - something defined as an error, is not undefined behavior
    - undefined behavior is not something your code can test for

    - code which produces an error at compile time
        - missing semicolon or unbalanced curly braces
        - method signature incompatible with the declaration
        - no matching candidate found for function call
        - adding the values of two pointers

    - code which results in a run time error
        - calling `myString.erase(10)` when the index is out of range

Undefined Behavior

- When is Undefined Behavior Acceptable
  - ?

Undefined Behavior

- When is Undefined Behavior Acceptable
  - in our opinion it is never acceptable

Undefined Behavior

- When is Undefined Behavior Acceptable
  - in our opinion it is never acceptable

  - signed integer overflow
    - if you believe it is unlikely to happen with your expected data set, do you still need to validate the input

Undefined Behavior

- When is Undefined Behavior Acceptable
  - in our opinion it is never acceptable

  - signed integer overflow
    - if you believe it is unlikely to happen with your expected data set, do you still need to validate the input

  - adding extraneous mutexes or locks can prevent a race condition
    - could introduce a deadlock, starvation, or a slow down

Undefined Behavior

- When is Undefined Behavior Acceptable
  - in our opinion it is never acceptable

  - signed integer overflow
    - if you believe it is unlikely to happen with your expected data set, do you still need to validate the input

  - adding extraneous mutexes or locks can prevent a race condition
    - could introduce a deadlock, starvation, or a slow down

  - accessing an inactive member of a union
    - reading an int after a float was saved, returns some raw data
    - maybe the read of the int occurs before the write of the float

Undefined Behavior

- Case Study
  - description
    - developer discovered undefined behavior in their code base
    - however all units tests were passing

    - they removed the undefined behavior from the application
    - noticed some of the units tests now fail

Undefined Behavior

- Case Study
  - description
    - developer noticed undefined behavior in their code base
    - however all units tests were passing

    - they removed the undefined behavior from the application
    - noticed some of the units tests now fail

    - if your code base has undefined behavior, all of your unit tests could be meaningless

Undefined Behavior

- Case Study
  - possible solutions
    - put the undefined behavior back in the code base so all the unit tests will pass
    - mark the failing unit tests as “flaky”

    - try a different compiler or platform
    - test with a sanitizer
    - debug the unit tests until they pass
    - figure out if the unit tests were always incorrect

Undefined Behavior

- Case Study
    - reasoning
        - unit tests were calling functionality in the application
        - with undefined behavior in the code base the unit tests should be considered meaningless
        - unit tests are part of the code base
        - full debugging can not happen - until all undefined behavior is removed from the application and unit tests

Undefined Behavior

- Software Design Philosophy
  - since the compiler can do anything, you may as well imagine that it will do something bad
  - if your code works with all current compilers then whatever you are doing is likely to become part of the standard
  - let people try it their way until the code crashes during a test
  - undefined behavior should exist only as an opt in feature, for those who care about speed
  - eventually the committee will finish their job and get rid of UB
  - programmers should provide a justifiable argument to use undefined behavior in their code base

Undefined Behavior

- Software Design Philosophy
  - reading from a file or a stream
    - did it open, is it empty, is the format correct
  - multi threaded application
    - what data should be atomic or guarded by a mutex
  - class design
    - which members should be marked const
  - for all code you write
    - does this code have any undefined behavior
    - checking for undefined behavior is not an extra step

Undefined Behavior

- Compiler Options
  - when optimization is turned off the compiler
    - does almost nothing special with your code
    - translates your code as near to literal as possible
    - undefined behavior may do what you expect so it appears your code is working as intended
  - normally optimization will be enabled
    - unreachable code can be removed
    - compilers are not required to diagnose undefined behavior
    - code can be “inlined” and then optimized
    - may produce unexpected results when a program has undefined behavior

Undefined Behavior

- Example 5
  - return statement missing from a “value returning function”
    - undefined behavior
    - some compilers provide a warning
    - detected by some sanitizers at run time
  - common outcome during program execution
    - may result in a crash
    - could return true every time
    - might proceed to the “next function” in the executable

```
bool monthOfCppCon21() {
  someData == “October”;
}
```

Undefined Behavior

- Example 6
  - operator[ ] returns a reference to an element in the string
  - this code has no test to verify index + 1 and index + 2 are in range
  - what happens when the loop reaches the end of the string

```
// QString did not originally provide null termination


QString inputStr = “class std::vector<int>“;
QString result;

for (int index = 0; index < inputStr.size(); ++index) {
  if (inputStr[index+1] == ‘:’ && inputStr[index+2] == ‘:’) {
    index += 2;
    result = inputStr.mid(index);          // expected “vector<int>”
  }
}
```

Undefined Behavior

- Example 7
  - some operations on a container invalidate iterators
  - there is no general rule and you need to verify for every operation

  - std::vector::insert() invalidates all iterators
    - iterators in a range based for loop are hidden
    - what does the current iterator point to after line A

```
std::vector<int> myContainer = { 42, 14, 5, 31, 9 };

for (auto &item : myContainer) {
  if (item == 5) {
    myContainer.insert(myContainer.begin(), -5);      // line A
  }
}
```

Undefined Behavior

- Example 8
  - keyword const_cast removes the "constness" of an object
  - modifying input is undefined behavior ***if*** the passed argument was originally declared as const

```
const std::string value = “tiger”;                          // line A
doThing8(value);


void doThing8(const std::string & input) {
  std::string &tmp = const_cast<std::string &>(input);      // line B
  tmp = “bear”;                                             // line C
}
```

Undefined Behavior

- Example 9
  - specializing a type trait which exists in the std namespace is UB
  - writing your own type traits is perfectly acceptable and they can be in any namespace other than std::

```
namespace std {

   template<>
   struct is_pointer<int>
      : public std::true_type                          // defines a type trait as true
   { };

}

bool var2 = std::is_pointer<int>::value;
```

Undefined Behavior

- Example 10
    - are either of the following expressions undefined behavior

```
int varA = 5;
varA = ++varA + 2;                 // pre increment


int varB = 3;
varB = varB++ + 2;                 // post increment
```

Undefined Behavior

- Example 10
  - pre increment and assignment to the same variable is undefined behavior in some versions of the standard

```
int varA = 5;
varA = ++varA + 2;                // C++03, undefined behavior
varA == 8;                        // C++11 and newer, defined


int varB = 3;
varB = varB++ + 2;                // post increment
```

Undefined Behavior

- Example 10
  - pre/post increment and assignment to the same variable is undefined behavior in some versions of the standard

```
int varA = 5;
varA = ++varA + 2;                 // C++03, undefined behavior
varA == 8;                         // C++11 and newer, defined


int varB = 3;
varB = varB++ + 2;                 // C++03 and C++11, undefined behavior
varB == 5;                         // C++17 and newer, defined
```

Undefined Behavior

- Resolving Undefined Behavior
  - tools to help locate UB in your code base
    - Address Sanitizer
    - Memory Sanitizer
    - Undefined Behavior Sanitizer
    - Thread Sanitizer
  - code reviews
    - institute a policy which exclusively checks for UB
  - pay attention to compiler warnings
  - build your code with multiple compilers
  - test crazy corner cases
  - treat undefined behavior as a critical bug

Undefined Behavior

- Back to the Basics . . .
  - undefined behavior can not be treated like an error
  - dealing with undefined behavior is not a sometimes thing
  - this is not a simple topic
  - projects can opt out of C++ features like exceptions, but you can not ignore undefined behavior
  - undefined behavior is the responsibility of every developer and you accepted it when choosing C++

Presentations - www.youtube.com/copperspice

Things every C++ programmer should know . . .

- ❑ Modern C++ Data Types (data types, references)
- ❑ Modern C++ Data Types (value categories)
- ❑ Modern C++ Data Types (move semantics, perfect forwarding)

- ❑ Learn Programming, then Learn How to Be a Programmer (CppOnSea Keynote) https://www.youtube.com/watch?v=jIa17JCaNvo

---

- ❑ What is the C++ Standard Library
- ❑ CsString library - Intro to Unicode
- ❑ char8_t
- ❑ Multithreading in C++
- ❑ Modern C++ Threads
- ❑ C++ Memory Model

Presentations - www.youtube.com/copperspice

- ❑ Why CopperSpice, Why DoxyPress
- ❑ Compile Time Counter
- ❑ Multithreading using CsLibGuarded
- ❑ Signals and Slots
- ❑ Templates in the Real World
- ❑ Copyright Copyleft
- ❑ What’s in a Container
- ❑ C++ Undefined Behavior
- ❑ Regular Expressions
- ❑ Type Traits
- ❑ C++ Tapas (typedef, forward declarations)
- ❑ C++ Tapas (typename, virtual, pure virtual)
- ❑ Lambdas in C++
- ❑ Overload Resolution
- ❑ Futures & Promises
- ❑ Thread Safety
- ❑ Constexpr Static Const
- ❑ When Your Codebase is Old Enough to Vote
- ❑ Sequencing
- ❑ Linkage
- ❑ Inheritance
- ❑ Evolution of Graphics Technology
- ❑ GPU, Pipeline, and the Vector Graphics API
- ❑ Declarations and Type Conversions
- ❑ Lambdas in Action
- ❑ Any Optional
- ❑ Variant
- ❑ std::visit
- ❑ CsPaint Library
- ❑ Moving to C++17
- ❑ Attributes
- ❑ Copy Elision
- ❑ Time Complexity
- ❑ Qualifiers
- ❑ Concepts in C++20
- ❑ Atomics
- ❑ Memory Model to Mutexes
- ❑ Mutexes + Lock = CsLibGuarded
- ❑ Variable Templates
- ❑ Paradigms and Polymorphism

Libraries

- CopperSpice
  - libraries for developing GUI applications
- CsSignal Library
  - standalone thread aware signal/slot library
- CsString Library
  - standalone unicode aware string library
- CsLibGuarded
  - standalone multithreading library for shared data

Libraries

- CsCrypto
    - C++ interface to the Botan and OpenSSL libraries

- CsPaint Library
    - standalone C++ library for rendering graphics on the GPU

Applications

- KitchenSink
  - contains over 30 demos, uses almost every CopperSpice library
- Diamond
  - programmers editor which uses the CopperSpice libraries
- DoxyPress & DoxyPressApp
  - application for generating source code and API documentation

Where to find CopperSpice

- www.copperspice.com
- twitter: @copperspice_cpp

- ansel@copperspice.com
- barbara@copperspice.com

- source, binaries, documentation files
  - download.copperspice.com

- source code repository
  - github.com/copperspice

- discussion
  - forum.copperspice.com