C++ Undefined Behavior What is it, and why should I care?

Marshall Clow
Qualcomm
marshall@idio.com
http://cplusplusmusings.wordpress.com
(intermittent)
Twitter: @mclow

C++Now 2014 May 2014

What is Undefined Behavior?

(from N3691 - C++)

1.3.24: undefined behavior

Behavior for which this International Standard imposes no requirements.

[Note: Undefined behavior may be expected when this International Standard omits any explicit definition of behavior or when a program uses an erroneous construct or erroneous data. Permissible undefined behavior ranges from ignoring the situation completely with unpredictable results, to behaving during translation or program execution in a documented manner characteristic of the environment (with or without the issuance of a diagnostic message), to terminating a translation or execution (with the issuance of a diagnostic message). Many erroneous program constructs do not engender undefined behavior; they are required to be diagnosed. — end note]

Some examples of UB

- Program crashes
- Program gives unexpected results
- Computer catches fire
- Cat gets pregnant
- Program appears to work fine
- There are no wrong answers!

Example #1:
No Wrong Answers!

```
#include <iostream>
// g++     sequence_points.cpp && ./a.out --> 10
// clang++ sequence_points.cpp && ./a.out --> 9

using namespace std;
int main () {
  int arr [] = { 0, 2, 4, 6, 8 };
  int i = 1;
  cout << i + arr[++i] + arr[i++] << endl;
  }
```

How can I get UB? (I)

- Signed integer overflow (but not unsigned!)
- Dereferencing NULL pointer or result of malloc(0)
- Shift greater than (or equal to) the width of the operand
- Reading from uninitialized variables
- Modifying a variable more than once in an expression

How can I get UB? (II)

- Buffer overflow
- Comparing pointers into two different data structures
- Pointer overflow
- Modifying a const object (C++) or a string literal

How can I get UB? (III)

- Negating INT_MIN
- Data races
- Mismatch between new and delete
- Calling a library routine w/o fulfilling the prerequisites
  - `memcpy` with overlapping buffers

Yikes!

```
#include <new>

class Foo {}; // complicated class

int main ( int argc, char *argv[] )
{
  int *p = new Foo [4];
// ..much later..
  delete p;

  return 0;
}
```

atomic_is_lock_free

20.9.2.5 shared_ptr atomic access

```
template<class T>
bool atomic_is_lock_free
        (const shared_ptr<T>* p);
```

Requires: p shall not be null.

Arithmetic Operations

3.9.1.4: If during the evaluation of an expression, the result is not mathematically defined or not in the range of representable values for its type, the behavior is undefined.

[Note: most existing implementations of C++ ignore integer overflows. Treatment of division by zero, forming a remainder using a zero divisor, and all floating point exceptions vary among machines, and is usually adjustable by a library function. —endnote]

Example #2: No Wrong Answers!

```
#include <stdio.h>
#include <stdbool.h>

int main ( int argc, char *argv[] )
{
  bool b;
  if (  b ) printf ( "true\n" );
  if ( !b ) printf ( "false\n" );
  return 0;
}
```

Why does C and C++ do this?

- It gives the compiler leeway to generate smaller code, by omitting checks
- By assuming no UB, the compiler can generate simpler, faster, smaller code.

Why is this important?

- Because compilers know it - and optimizers take advantage of it.
- It is perfectly legal to transform a program exhibiting UB into any other program.
- Remember - in UB, there are no wrong answers!

Different kinds of routines

- Type 1 - no UB, no matter what the inputs
- Type 2 - UB for some subset of all possible inputs.
- Type 3 - UB every time, no matter what the inputs.

Example #3

```
int * do_something ( int *p )
{
  log ( "do_something %d", *p );
  if ( !p )
  {
    // code here
    p = malloc ( ... );
    // more code here
  }
return p;
}
```

Example #4

```
#include <stdio.h>

int main ()
{
  int i = 0x10000000;
  int c = 0;
  do
  {
    c++;
    i += i;
    printf ("%d\n", i);
  } while (i > 0);
  printf ("%d iterations\n", c);
}
```

Why do we care? (I)

- It's surprisingly easy to write code with undefined behavior.
  - http://code.google.com/p/nativeclient/issues/detail?id=245
- UB code may "work" for a while, and then "break" when the optimization level is increased or the compiler is upgraded.
  - This is what the STACK people call "optimization-unstable code". (remember, no wrong answers!)

Why do we care? (II)

- UB shows up in "tricky" code; frequently code that is attempting security checks.
  - http://gcc.gnu.org/bugzilla/show_bug.cgi?id=30475
  - Bugs that STACK found in Postgres

Example #5

```
// Checks are hard to write correctly
// From “Apple Secure Coding Guidelines”
size_t bytes = n * m;
if (n > 0 && m > 0 && SIZE_MAX/n >= m) {
    ... /* allocate "bytes" space */
}
```

Aliasing

```
struct Foo {
    int a;
};
struct Bar {
    int a;
    int b;
};

Foo f{3};
Bar *p = (Bar *)&f;
p->a = 4;
cout << f.a;
```

Don't be this guy!

https://www.youtube.com/watch?v=HRJ-VLehcJg

What can I do about UB?

- Be aware of UB.
- Don't blame the compiler (AKA "don't shoot the messenger")
- if you're doing "something tricky" think about UB.
- Build your code with several compilers/ different optimization levels.

You can't check for UB after the fact

- It's too late
- The damage has already been done

If you write this:

```
bool WillThisOverflow ( int a )
{ return a + 100 < a; }
```

the compiler can/may/will optimize it to:

```
bool WillThisOverflow ( int a )
{ return false; } // Why?
```

Instead, you should write:

```
bool WillThisOverflow ( int a )
{ return a < ( INT_MAX - 100 ); }
```

Are there any tools to help detect UB?

- Tools are starting to appear
- clang has -fsanitize=undefined
  - See http://blog.llvm.org/2013/04/testing-libc-with-fsanitizeundefined.html
- John Regehr's Integer Overflow Checker
- STACK (this past summer from MIT)

Quiz

```
// Optimize this code
void
contains_null_check(int *P)
{
  int dead = *P;
  if (P == 0)
    return;
  *P = 4;
}
```

Quiz

```
// Optimize this code
void
contains_null_check(int *P)
{
   int dead = *P;
   if (P == 0)
      return;
   *P = 4;
}
```

Questions?

References

- A Guide to Undefined Behavior in C and C++, Part I http://blog.regehr.org/archives/213 (links to II and III)
- Towards optimization-safe systems http://pdos.csail.mit.edu/papers/stack:sosp13.pdf
- http://clang.llvm.org/docs/UsersManual.html#controlling-code-generation
- What every C programmer should know about undefined behavior http://blog.llvm.org/2011/05/what-every-c-programmer-should-know.html (with link to parts II and III)

References

- It's Time to Get Serious About Exploiting Undefined Behavior http://blog.regehr.org/archives/761
- Finding Undefined Behavior Bugs by Finding Dead Code http://blog.regehr.org/archives/970
- About unspecified and undefined behavior in C (ACCU 2013) http://www.pvv.org/~oma/UnspecifiedAndUndefined_ACCU_Apr2013.pdf