Undefined Behaviour

Tools and Libraries for Compile-time Software Engineering
HiPEAC Conference 2024
Munich, Germany

Paul Keir [1]
Joel FALCOU [2]

[1]School of Computing, Engineering & Physical Sciences
University of the West of Scotland, Paisley, UK

[2]Le Laboratoire Interdisciplinaire des Sciences du Numérique (LISN)
Université Paris-Saclay, Paris, France

January 17th, 2024

Classes of Observable Behaviour

- The C++ standard does not define the observable behaviour of *all* programs
- Many programs exhibit:
  - *unspecified behaviour*; or
  - *undefined behaviour* *
- Both programs opposite include *unspecified behaviour*
  - At issue is the pointer comparison (see here)
- Both compile and run with no warnings
- We don't care if the `assert` passes or fails

```
#include <cassert>

int main()
{
  int a1[4]{}, a2[4]{};
  assert(&a1[0] > &a2[0]);
}
```

```
#include <cassert>

int main()
{
  int* p1 = new int[4]{};
  int* p2 = new int[4]{};
  assert(&p1[0] < &p2[0]);
  delete [] p1;
  delete [] p2;
}
```

* Additionally, non-standard programs may also be classed as *ill-formed*; or as containing *implementation-defined* behaviour.

Restrictions During Constant Evaluation

- A core constant expression cannot contain a three-way comparison, relational, or equality operator where the result is *unspecified* (see here)
- Compare the functions opposite to those on the previous slide
- Neither pointer is required to compare greater than the other
- Consequently, neither compiles
- n.b. The error is issued before either `static_assert` considers its argument

```
constexpr bool cexpr_cmpr1()
{
  int a1[4]{}, a2[4]{};
  return &a1[0] > &a2[0];
}

constexpr bool cexpr_cmpr2()
{
  int* p1 = new int[4]{};
  int* p2 = new int[4]{};
  bool b = &p1[0] < &p2[0];
  delete [] p1;
  delete [] p2;
  return b;
}

static_assert(cexpr_cmpr1);
static_assert(cexpr_cmpr2);
```

Undefined Behaviour

- The program opposite contains *undefined behaviour*

  `noinit`: Uninitialised read

  `bounds`: Read outside array bounds
- Both compile and run; with warnings
- Again, we don't care if the `assert` passes or fails

```cpp
#include <cassert>

bool noinit()
{
  int i;
  return i==42;
}

bool bounds()
{
  int arr[4]{};
  return arr[4]==42;
}

int main()
{
  assert(noinit() && bounds());
}
```

* GCC and Clang trunk are used for testing throughout.

Undefined Behaviour during Constant Evaluation

- Prohibited within a *core constant expression*:
  - specific *unspecified behaviour* (previous slide);
  - most *undefined behaviour* (see here)
- In some cases it is unspecified whether an expression containing UB is a core constant expression (see here)
- The undefined behaviour opposite is not permitted; and neither function compiles

```
constexpr bool cexpr_noinit()
{
  int i;
  return i==42;
}

constexpr bool cexpr_bounds()
{
  int arr[4]{};
  return arr[4]==42;
}

static_assert(cexpr_noinit());
static_assert(cexpr_bounds());
```

Undefined Behaviour Oracle?

- You will consequently be more aware of undefined behaviour (UB) when working with compile-time programs
- Though the compiler may not always intervene
- e.g. The program opposite has more than one modification to the same scalar in an expression that is unsequenced

  ...and so contains UB
- The program compiles and runs; with warnings

```
#include <cassert>

constexpr bool eval_order()
{
  int i{0};

  if (std::is_constant_evaluated())
  {
    return 3 == ++i + ++i;
  }

#ifdef __clang__
  return 3 == ++i + ++i;
#else
  return 4 == ++i + ++i;
#endif
}

int main()
{
  assert(eval_order());
  static_assert(eval_order());
}
```

UWS UNIVERSITY OF THE WEST *of* SCOTLAND

RSE *The Royal Society of Edinburgh*
KNOWLEDGE MADE USEFUL