Implicit Lifetimes

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

No Implicit Object Creation

```cpp
#include <cstdlib>

void allocate()
{
  void *vp = std::malloc(sizeof(int)*1024);
  int *p = static_cast<int*>(vp);
  p[41] = 6*9;
  std::free(p);
}
```

- Writing to allocated memory of simple types is common practice
  - In 'C'-style legacy code `malloc` is often used
  - In idiomatic C++ this would utilise `std::allocator`
  - ...but either causes undefined behaviour; no `int` object was created
- R. Smith & V. Voutilainen note this longstanding defect in P0593
  - *Implicit creation of objects for low-level object manipulation*

No Implicit Object Creation

```cpp
#include <memory>

constexpr void cpp_allocate()
{
  std::allocator<int> alloc;
  int *p = alloc.allocate(1024);
  p[41] = 6*9;
  alloc.deallocate(p,1024);
}
```

- Writing to allocated memory of simple types is common practice
  - In 'C'-style legacy code `malloc` is often used
  - In idiomatic C++ this would utilise `std::allocator`
  - ...but either causes undefined behaviour; no `int` object was created
- R. Smith & V. Voutilainen note this longstanding defect in P0593
  - *Implicit creation of objects for low-level object manipulation*
- `std::allocator::allocate` became `constexpr` in C++20
- But the code above cannot be constant evaluated

No Implicit Object Creation

```cpp
#include <memory>

constexpr void cpp_allocate()
{
  std::allocator<int> alloc;
  int *p = alloc.allocate(1024);
  p[41] = 6*9;
  alloc.deallocate(p,1024);
}
```

- The adopted paper defines implicit lifetime types; types where:
  - Creating an instance of the type runs no code
  - Destroying an instance of the type runs no code
- Such types are (as of C++20) permitted such implicit creation
  - ...but not during constant evaluation (P0593 Section 3.5)
- P2674 introduced `std::is_implicit_lifetime` in C++23
  - ...alas no compiler has implemented it to date

A Hurdle for constexpr Programs

- Consequently, a suitable constructor must be called
- C++20's `std::construct_at` provides appropriate syntax
  - "Placement new" is still not `constexpr`
  - Barry Revzin's P2747 proposes it for C++26

```
#include <memory>

constexpr void cpp_allocate()
{
  std::allocator<int> alloc;
  int *p = alloc.allocate(1024);




      std::construct_at(&p[41]);

  p[41] = 6*9;
  alloc.deallocate(p,1024);
}
```

A Hurdle for constexpr Programs

- Consequently, a suitable constructor must be called
- C++20's `std::construct_at` provides appropriate syntax
  - "Placement new" is still not `constexpr`
  - Barry Revzin's P2747 proposes it for C++26

```
#include <memory>

constexpr void cpp_allocate()
{
  std::allocator<int> alloc;
  int *p = alloc.allocate(1024);

  if (std::is_constant_evaluated()) { // if consteval in C++23

      std::construct_at(&p[41]);
  }
  p[41] = 6*9;
  alloc.deallocate(p,1024);
}
```

A Hurdle for constexpr Programs

- Typically require to construct all array elements after creation
- While respecting custom allocators via `std::allocator_traits`:

```cpp
#include <memory>

constexpr void cpp_allocate()
{
  std::allocator<int> alloc;
  int *p = alloc.allocate(1024);
  using a_t = std::allocator_traits<decltype(alloc)>;
  if (std::is_constant_evaluated()) { // if consteval in C++23
    for (std::size_t i = 0; i < 1024; i++)
      a_t::construct(alloc, &p[i]);
  }
  p[41] = 6*9;
  alloc.deallocate(p,1024);
}
```

A Hurdle for constexpr Programs

- P2674's `is_implicit_lifetime` trait should allow more precision
- Allow implicit object creation also during constant evaluation?
- Even just a `construct_n`? (akin to `destroy_n`)

```
#include <memory>

constexpr void cpp_allocate()
{
  std::allocator<int> alloc;
  int *p = alloc.allocate(1024);
  using a_t = std::allocator_traits<decltype(alloc)>;
  if (std::is_constant_evaluated()) { // if consteval in C++23
    for (std::size_t i = 0; i < 1024; i++)
      a_t::construct(alloc, &p[i]);
  }
  p[41] = 6*9;
  alloc.deallocate(p,1024);
}
```

A Hurdle for constexpr Programs

- When porting programs to `constexpr` such details require care
- GCC and MSVC are lenient; only Clang throws an error

```
#include <memory>

constexpr void cpp_allocate()
{
  std::allocator<int> alloc;
  int *p = alloc.allocate(1024);
  using a_t = std::allocator_traits<decltype(alloc)>;
  if (std::is_constant_evaluated()) { // if consteval in C++23
    for (std::size_t i = 0; i < 1024; i++)
      a_t::construct(alloc, &p[i]);
  }
  p[41] = 6*9;
  alloc.deallocate(p,1024);
}
```


UWS
UNIVERSITY OF THE
WEST of SCOTLAND


RSE
The Royal Society
of Edinburgh
KNOWLEDGE MADE USEFUL