Constexercise

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

Problem Description

The code here...

1. Allocates "a lot" of untyped memory
2. Assigns `int` 42 at the start
3. Increments the `int` via its untyped address

(Available in the repo as constexercise.cpp)

```
int main()
{
  assert(doit());
  // static_assert(doit());
}
```

```
int* incr_int(void* vp)
{
  int *p = reinterpret_cast<int*>(vp);
  if (p)
    (*p)++;
  return p;
}

bool doit()
{
  void *vp = malloc(sizeof(int)*1024);
  int *p = reinterpret_cast<int*>(vp);
  *p = 42;
  p = incr_int(p);
  bool b = (*p==43);
  free(vp);
  return b;
}
```

Problem Description

Challenge:

1. Modify to allow the `static_assert` also to pass
2. Maintain the function type signatures
3. Use a C++26 compiler (tip)
4. For extra challenge: use Clang

```
int main()
{
  assert(doit());
  // static_assert(doit());
}
```

```
int* incr_int(void* vp)
{
  int *p = reinterpret_cast<int*>(vp);
  if (p)
    (*p)++;
  return p;
}

bool doit()
{
  void *vp = malloc(sizeof(int)*1024);
  int *p = reinterpret_cast<int*>(vp);
  *p = 42;
  p = incr_int(p);
  bool b = (*p==43);
  free(vp);
  return b;
}
```

The slides that follow step through the solution; so don't read ahead if you want to tackle the problem yourself.

1. Add constexpr

```
constexpr int* incr_int(void* vp)
{
  int *p = reinterpret_cast<int*>(vp);
  if (p)
    (*p)++;
  return p;
}

constexpr bool doit()
{
  void *vp = malloc(sizeof(int)*1024);
  int *p = reinterpret_cast<int*>(vp);
  *p = 42;
  p = incr_int(p);
  bool b = (*p==43);
  free(vp);
  return b;
}
```

2. Use constexpr C++ standard allocator

```
constexpr int* incr_int(void* vp)
{
  int *p = reinterpret_cast<int*>(vp);
  if (p)
    (*p)++;
  return p;
}

constexpr bool doit()
{
  int *p = std::allocator<int>{}.allocate(1024);

  *p = 42;
  p = incr_int(p);
  bool b = (*p==43);
  std::allocator<int>{}.deallocate(p,1024);
  return b;
}
```

3. Start the lifetime of the first int

```
constexpr int* incr_int(void* vp)
{
  int *p = reinterpret_cast<int*>(vp);
  if (p)
    (*p)++;
  return p;
}

constexpr bool doit()
{
  int *p = std::allocator<int>{}.allocate(1024);
  std::construct_at(p);
  *p = 42;
  p = incr_int(p);
  bool b = (*p==43);
  std::allocator<int>{}.deallocate(p,1024);
  return b;
}
```

Only a (C++26) static_cast is required here

```
constexpr int* incr_int(void* vp)
{
  int *p = static_cast<int*>(vp);
  if (p)
    (*p)++;
  return p;
}

constexpr bool doit()
{
  int *p = std::allocator<int>{}.allocate(1024);
  std::construct_at(p);
  *p = 42;
  p = incr_int(p);
  bool b = (*p==43);
  std::allocator<int>{}.deallocate(p,1024);
  return b;
}
```

Bonus Challenge!

Lastly:

1. Add the code below before the `return` statement in `incr_int`
2. Ensure the runtime and static assert still pass

```
b = b && incr_int(nullptr) == nullptr;
```

Bonus: Test for nullptr *before* the cast

```
constexpr int* incr_int(void* vp)
{
  int *p = vp ? static_cast<int*>(vp) : nullptr;
  if (p)
    (*p)++;
  return p;
}

constexpr bool doit()
{
  int *p = std::allocator<int>{}.allocate(1024);
  std::construct_at(p);
  *p = 42;
  p = incr_int(p);
  bool b = (*p==43);
  b = b && incr_int(nullptr) == nullptr;
  std::allocator<int>{}.deallocate(p,1024);
  return b;
}
```

Acknowledgements

UWS UNIVERSITY OF THE WEST of SCOTLAND

RSE The Royal Society of Edinburgh
KNOWLEDGE MADE USEFUL