1-gsl-pointers

June 3, 2024

1 About pointers

Many of the problems with pointers come from their multiple uses. They point indifferently to: * memory areas of the stack or of the heap, * single objects or tables, * heap zones whose release is managed elsewhere, * heap zones to be released at the end of the current instruction block.

Let's sort it out and express these uses more clearly.

1.1 Raw pointers and references: `T*` and `T&`

A "raw" pointer (e.g. `T *`) is supposed to have the most ordinary meaning: **it points to an object, but does not "own" it**. In what follows, the portion of code that uses this pointer is not supposed to do a `delete` at the end of the use.

If the pointer itself is not meant to change value, and is never meant to be zero, **we will prefer a reference** whenever possible.

In addition, using a raw pointer to designate an array is to be avoided. Many other solutions exist now.

1.2 Non-zero pointers: `gsl::not_null<T>`

When dereferencing a pointer (via `*` or `->`), a common practice is to first check that it is not null as a precaution, which obscures and slows down the code.

Whenever possible, **replace this pointer with a reference**.

Otherwise, use `gsl::not_null<T>`, which ensures that you never assign a null value to the pointer.

If you try to assign a null value, you will get an error at compile time (`d2` in the example below), or at run time (`d4` in the example below).

[1]:
```
%%file tmp.gsl-pointer.cpp

#include <iostream>
#include <gsl/gsl>

struct Demo {
    Demo() { std::cout<<"Constructor"<<std::endl ; }
    ~Demo() { std::cout<<"Destructor"<<std::endl ; }
} ;
```

```
int main() {
    gsl::not_null<Demo *> d1 { new Demo() } ;
    delete d1 ;
    // ...
    gsl::not_null<Demo *> d2 { nullptr } ;
    // ...
    Demo * d3 { nullptr } ;
    // ...
    gsl::not_null<Demo *> d4 { d3 } ;
}
```

```
Writing tmp.gsl-pointer.cpp
```

[2]:
```
!rm -f tmp.gsl-pointer.exe && g++ -std=c++17 -I./ tmp.gsl-pointer.cpp -o tmp.
↪gsl-pointer.exe
```

```
tmp.gsl-pointer.cpp: In function ‘int

main()’:
tmp.gsl-pointer.cpp:14:40: error: use of deleted
function ‘gsl::not_null<T>::not_null(std::nullptr_t)
[with T = Demo*; std::nullptr_t = std::nullptr_t]’
   14 |     gsl::not_null<Demo *> d2 { nullptr } ;
      |                                        ^
In file included from ./gsl/gsl:25,
                 from tmp.gsl-pointer.cpp:3:
./gsl/pointers:106:5: note: declared here
  106 |     not_null(std::nullptr_t) = delete;
      |     ^~~~~~~~
```

[3]:
```
!./tmp.gsl-pointer.exe
```

```
/usr/bin/sh: 1: ./tmp.gsl-pointer.exe: not found
```

1.3 Ownership pointers `gsl::owner<T>`

The type `gsl::owner<T>` is used to mark that a pointer owns the pointed object. `T` is assumed to be a pointer type, such as `int *`.

The type `gsl::owner<T>` does nothing in itself: you are still in charge of making the call to `delete`, but it clarifies your intention and can allow static checking tools to detect a forgotten delete.

[4]:
```
%%file tmp.gsl-pointer.cpp

#include <iostream>
#include <gsl/gsl>

struct Demo {
    Demo() { std::cout<<"Constructor"<<std::endl ; }
    ~Demo() { std::cout<<"Destructor"<<std::endl ; }
```

```
} ;

int main() {
    gsl::owner<Demo *> d { new Demo() } ;
    delete d ;
}
```

```
Overwriting tmp.gsl-pointer.cpp
```

[5]:
```
!rm -f tmp.gsl-pointer.exe && g++ -std=c++17 -I./ tmp.gsl-pointer.cpp -o tmp.
 ↪gsl-pointer.exe
```

[6]:
```
!./tmp.gsl-pointer.exe
```

```
Constructor
Destructor
```

1.4 Smart pointers `std::unique_ptr<T>` and `std::shared_ptr<T>`

In the case of owning pointers, the GSL encourages the use of pointers from the standard libraries `std::unique_ptr<T>` and `std::shared_ptr<T>`.

`std::unique_ptr<T>` * costs nothing, * but is not copiable.

`std::shared_ptr<T>` * easy to copy, * but expensive.

Whenever you can, entrust your raw pointer, created by `new`, directly to one of these smart pointers, and only use `gsl::owner<T>` as a last resort. Even better, use `std::make_unique` and `std::make_shared`.

2 Take away

Non-owning pointers and references, by order of preference

- `T&`: non-owner, cannot be null (always attached to an element).
- `gsl::not_null<T>`: non-owner, T is a pointer, cannot be zero.
- `T*`: non-owner, can be null, assumed to point to a unique element.

Owning pointers, by order of preference

- `unique_ptr<T>`: unique owner, not copyable, movable, automatic delete, efficient.
- `shared_ptr<T>`: shared ownership, automatic delete, simple but less efficient.
- `gsl::owner<T>`: owner, T is a pointer, can be zero, assumed to point to a dynamically allocated element (on the heap)‘

3 Questions ?

4 Exercise

In the code below: * write `my_owner`, which must emulate `gsl::owner` (trivial) ; * write `my_not_null`, which must emulate `gsl::not_null` (easy) ; * make sure that your types only accept pointers as parameters (difficult).

[10]:
```
%%file tmp.gsl-pointers.h

#include <iostream>

class Demo {
  public:
    Demo() { std::cout<<"Constructor"<<std::endl ; }
    void print() { std::cout<<"Printing"<<std::endl ; }
    ~Demo() { std::cout<<"Destructor"<<std::endl ; }
} ;
```

```
Writing tmp.gsl-pointer.h
```

[11]:
```
%%file tmp.gsl-pointer.cpp

#include "tmp.gsl-pointers.h"
#include <type_traits>

//... PUT HERE YOUR IMPLEMENTATION OF my_owner ...

int main() {
  //my_owner<Demo> p1 ;              // COMPILATION ERROR: Demo is not a pointer
  my_owner<Demo *> p2 {new Demo()} ;
  p2->print() ;
  delete p2 ;
}
```

```
Overwriting tmp.gsl-pointer.cpp
```

[12]:
```
!rm -f tmp.gsl-pointers-owner.exe && g++ -std=c++17 -I./ tmp.gsl-pointers-owner.
 ↪cpp -o tmp.gsl-pointers-owner.exe
```

```
tmp.gsl-pointer.cpp: In function ‘int

main()’:
tmp.gsl-pointer.cpp:9:3: error:

‘my_owner’ was not declared in this scope
    9 |   my_owner<Demo *> p2 {new Demo()} ;
      |   ^~~~~~~~
tmp.gsl-pointer.cpp:9:17: error: expected
primary-expression before ‘*’ token
```

```
    9 |   my_owner<Demo *> p2 {new Demo()} ;
      |                  ^
tmp.gsl-pointer.cpp:9:18: error: expected
primary-expression before ‘>’ token
    9 |   my_owner<Demo *> p2 {new Demo()} ;
      |                   ^
tmp.gsl-pointer.cpp:9:20: error:

‘p2’ was not declared in this scope
    9 |   my_owner<Demo *> p2 {new Demo()} ;
      |                    ^~
tmp.gsl-pointer.cpp:11:3: error: type
‘<type error>’ argument given to ‘delete’, expected
pointer
   11 |   delete p2 ;
      |   ^~~~~~~~~
```

[9]:
```
!./tmp.gsl-pointers-owner.exe
```

```
/usr/bin/sh: 1: ./tmp.gsl-pointer.exe: not found
```

[13]:
```
%%file tmp.gsl-pointers-not-null.cpp

#include "tmp.gsl-pointers.h"
#include <type_traits>

class Demo
 {
  public:
    Demo() { std::cout<<"Constructor"<<std::endl ; }
    void print() { std::cout<<"Printing"<<std::endl ; }
    ~Demo() { std::cout<<"Destructor"<<std::endl ; }
 } ;

//... PUT HERE YOUR IMPLEMENTATION OF my_not_null ...

int main() {
  //my_not_null<Demo*> p1 ;          // COMPILATION ERROR: p1 is not initialized
  //my_not_null<Demo*> p2(nullptr) ; // COMPILATION ERROR: p2 cannot be null
  my_not_null<Demo *> p3 = new Demo() ;
  //p3 = nullptr ;                   // EXECUTION ERROR: p3 cannot be null
  p3->print() ;
  delete p3 ;
}
```

```
Writing tmp.gsl-pointers-not-null.cpp
```

[8]:
```
!rm -f tmp.gsl-pointers-not-null.exe && g++ -std=c++17 -I./ tmp.
 ↪gsl-pointers-not-null.cpp -o tmp.gsl-pointers-not-null.exe
```

```
tmp.gsl-pointer.cpp: In function ‘int
main()’:
tmp.gsl-pointer.cpp:23:3: error:
‘my_owner’ was not declared in this scope
   23 |   my_owner<Demo *> d2 = new Demo() ;
      |   ^~~~~~~~
tmp.gsl-pointer.cpp:23:17: error: expected
primary-expression before ‘*’ token
   23 |   my_owner<Demo *> d2 = new Demo() ;
      |                 ^
tmp.gsl-pointer.cpp:23:18: error: expected
primary-expression before ‘>’ token
   23 |   my_owner<Demo *> d2 = new Demo() ;
      |                  ^
tmp.gsl-pointer.cpp:23:20: error:
‘d2’ was not declared in this scope
   23 |   my_owner<Demo *> d2 = new Demo() ;
      |                    ^~
tmp.gsl-pointer.cpp:24:3: error:
‘my_not_null’ was not declared in this scope
   24 |   my_not_null<Demo *> p3 = d2 ;
      |   ^~~~~~~~~~~
tmp.gsl-pointer.cpp:24:20: error: expected
primary-expression before ‘*’ token
   24 |   my_not_null<Demo *> p3 = d2 ;
      |                    ^
tmp.gsl-pointer.cpp:24:21: error: expected
primary-expression before ‘>’ token
   24 |   my_not_null<Demo *> p3 = d2 ;
      |                     ^
tmp.gsl-pointer.cpp:24:23: error:
‘p3’ was not declared in this scope
   24 |   my_not_null<Demo *> p3 = d2 ;
      |                       ^~
tmp.gsl-pointer.cpp:27:3: error: type
‘<type error>’ argument given to ‘delete’, expected
pointer
   27 |   delete d2 ;
      |   ^~~~~~~~~
```

[9]:
```
!./tmp.gsl-pointers-not-null.exe
```

```
/usr/bin/sh: 1: ./tmp.gsl-pointer.exe: not found
```

4.1 Sources

- http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#gsl-guidelines-support-library
- http://modernescpp.com/index.php/c-core-guideline-the-guidelines-support-library
- http://nullptr.nl/2018/08/refurbish-legacy-code/