C++ London University Session 8

Tristan Brindle

Feedback and Communication

- Your feedback is vital
- Otherwise, we don’t know what you don’t know!
- Please join the #cpplondonuni channel on the cpplang Slack — Go to https://cpplang.now.sh/ for an “invitation”

Feedback and Communication

- Did you know we have a blog? cpplondonuni.com/blog
- New post from Oli: “Sanitizers — or how I learned to stop worrying and love the compiler”
- …and now we’re on Twitter too, find us @cpplondonuni

Today's Lesson Plan

- Lightning talks!
- Last week's exercise
- Pointers continued

Lightning Talks

Last week's exercise

- Solutions available at

- https://github.com/CPPLondonUni/pointers101/tree/solutions

Previously at C++ London University…

- A pointer represents the *memory address* of a variable
- Pointers are *value types*
- Form a pointer by taking the address of a variable with &x (or `std::addressof()`)
- *Declare* a pointer by saying eg `int* p = &i;`
- *Dereference* a pointer by saying `*p`
- Use `nullptr` to represent an invalid (“null”) pointer

Const Pointers

Const ~~Pointers~~ References

- Let’s go back to references for a moment
- Recall that we have both ordinary (mutable) references and const references (aka “reference-to-const”)
- We write a reference-to-const as “`const T&`” or “`T const&`”.
- A reference-to-const means that we cannot change the value of the variable via that reference.
- We can bind a const reference to a mutable variable, but not the other way around: *we can only add const-ness*

```
int i = 0;
int& r = i;
const int& cr = i; // okay

const int c = 1;
const int& cr = c; // okay
int& r = c; // ERROR
```

Const Pointers

~~Const Pointers~~ Pointers-to-const

~~Const Pointers~~ Pointers-to-const

- We can have a type that is a "*pointer to const T*"
- A pointer to const means that the value of the "pointee" cannot be changed by dereferencing that pointer.
- We write a pointer-to-const as "`const T*`" or "`T const*`"
- A *pointer to const T* can hold the address of a ***non**-const T*
- However, a *pointer to (non-const) T* cannot hold the address of a *const T*. Remember: *we can only add const-ness*.
- This works in exactly the same way as for references

```
int i = 0;
int* p = &i;
const int* pc = &i; // okay

const int c = 1;
const int* pc = &c; // okay
int* p = &c; // ERROR
```

Const Pointers

- But remember, a pointer isn't special: it's just a value type, like an `int`.

- This means we can change the *value* of the pointer

```
int i = 0;
int* p = &i;
int j = 1;
p = &j;
```

Const Pointers

- But remember, a pointer isn't special: it's just a value type, like an `int`

- This means that we can make the **pointer itself** const!

- So we have two types of "const-ness" to think about: the const-ness of the pointer itself, and the const-ness of the thing it's pointing to.

- These can be mixed and matched, giving four possibilities!

Const Pointers

- How do we declare a const pointer?
- We cannot say “`const int* p`” — that declares a pointer-to-`const int`!
- Instead, we need to say “`int* const p`” — the const goes *after* the “star”

Const Pointer Q&A

- Q: How do we write the type “pointer to int”?
- A: `int*`
- Q: How do we write the type “const pointer to int”?
- A: `int* const`
- Q: How do we write the type “pointer to const int”?
- A: `const int*` *or* `int const*`
- Q: How do we write the type “const pointer to const int”?
- A: `const int* const` *or* `int const* const`

Const Pointers

- Remember that we can have a pointer which holds the address of another pointer — “pointers to pointers”

- For example, the type “int**” is read as “pointer to pointer to int”

- This can make things quite hard to decipher when we have multiple consts involved!

- The trick is to read types *backwards*, from right to left

- Remember: `const` applies to the thing on its *left* (except when there is nothing on its left)

Const Pointer Q&A (2)

- Q: How do we write the type “const pointer to pointer to int”?
- A: `int** const`
- Q: How do we write the type “pointer to pointer to const int”?
- A: `int const**` *or* `const int**`
- Q: How do we say the type “`int* const *`”?
- A: “Pointer to const pointer to int”
- Q: How do we say the type “`int const* const* const&`”?
- A: “Reference to const pointer to const pointer to const int” (!)

Const Pointers

- Exercise!

- https://github.com/CPPLondonUni/pointers101

The “this” pointer

- Inside member functions, the special variable `this` is a pointer to the current class instance

- Within *const* member functions, `this` is a pointer-to-const: you may not change the value of any member variables

- (…except those explicitly declared *mutable*, but that’s a topic for another time.)

- `this` is always a const pointer (why?)

The arrow operator

- In C++ (and most other languages), we access the members of a class using a `.` (dot)

- For example, we can say

```
struct example {
    int i;
};


example e{};
e.i = 4;
```

The arrow operator

- If we have a pointer to an class instance, to access its members we need to *dereference* the pointer

- But (unfortunately) we cannot say `*p.i` — this means “dereference the member `i` of `p`”

- To get the meaning we want, we need to use brackets: `(*p).i` means “dereference `p`, then give me the member `i`”

The arrow operator

- Dereferencing a pointer and accessing members is a common operation, and having to use brackets all the time is a bit of a pain

- For this reason, C invented (and C++ inherited) the *arrow operator*

- With this, we can dereference a pointer `p` and access the member `i` of the pointed-to instance in one go, by saying `p->i`.

The arrow operator

- For built-in pointers, `p->i` means exactly the same as `(*p).i`

- Like other operators in C++, the arrow operator -> and dereference operator * can be *overloaded* for different types

- This is the basis for “smart pointers” — custom types which behave like pointers, but have special properties like lifetime management.

Next Time

- Oli will teach us all about Git 🙂

Online Resources

- https://isocpp.org/get-started
- cppreference.com — The bible, but aimed at experts
- cplusplus.com — Another reference site, also has a tutorial section
- learncpp.com — Free online tutorial, very up-to-date
- https://www.pluralsight.com/authors/kate-gregory - Comprehensive set of courses from an experienced C++ trainer (free trial)
- reddit.com/r/cpp_questions
- Cpplang Slack channel — https://cpplang.now.sh/ for an “invite”
- StackOverflow (but…)

Thanks for coming!

C++ London University:

- Website: cpplondonuni.com
- Twitter: @cpplondonuni
- Github: github.com/CPPLondonUni

Where to find Tom Breza:

- On Slack: cpplang.slack.com #learn #cpplondon
- E-mail: tom@PCServiceGroup.co.uk
- Mobile: 07947451167

My stuff:

- Website: tristanbrindle.com
- Twitter: @tristanbrindle
- Github: github.com/tcbrindle

See you next time! 🙂