C++ 1

Week 4 – Smart Pointers

Smart Pointers

- `#include <memory>`
- `unique_ptr`: exclusief eigendom
- `shared_ptr`: gedeeld eigendom
- `weak_ptr`: doorbreek lussen in circulaire verwijzingen bij gedeeld eigendom

`unique_ptr`

- exclusief eigendom
- geen copy c'tor of –assignment
- wel move c'tor en –assignment
- bijvoorbeeld handig voor:
  - exception-safe gebruik van heap memory
  - doorgeven van eigendom van heap memory aan een functie
  - `return`en van heap memory uit een functie
  - pointers opslaan in containers

voorbeeld van unique_ptr

```
class Ding; // elders gedefinieerd
using namespace std;

void func() {
   unique_ptr<Ding> ding { new Ding };
   ding->doe_iets(); // ding lijkt Ding*
   // hier impliciet: delete ding;
}
```

unique_ptr doet ook arrays

```
unique_ptr<int[]> make_sequence(int n) {
   unique_ptr<int[]> p { new int[n] };
   for (int i = 0; i < n; ++i)
      p[i] = i;
   return p;
}
```

- bij de return gebruikt unique_ptr een move, zodat er geen data wordt gekopieerd; nice!

`shared_ptr`

- gebruikt "reference counting" om bij te houden hoeveel anderen naar hem wijzen
- als de laatste verwijzing "loslaat", roept hij `delete` tegen de geëncapsuleerde pointer
- is "duurder" dan `unique_ptr`
- niet gebruiken om eigendom over te dragen, dat doet `unique_ptr` beter en efficiënter
- alleen gebruiken als er niet één eigenaar van een object is aan te wijzen

voorbeeld van shared_ptr

```
class C {
   int i;
   string s;
   double d;
   // ...
};

auto p = make_shared<C>(1, "Avans", 3.14);
auto q { p }; // copy
```

- p en q zijn nu shared_ptr<C>, wijzend naar één instantie van C met de gegeven waarden

weak_ptr

- doorbreek circulaire referenties van shared_ptrs

```
void doe_iets(weak_ptr<Ding> ding) {
   if (shared_ptr<Ding> q = ding.lock()) {
      // doe iets met q
   } else { // oeps: Ding was al deleted
      ding.reset();
   }
}
```