Programming in C and C++ - Supervision 3

Nandor Licker

October 2019

1 Classes

Q1 Implement a `template <typename T, size_t N> class SmallVector` which stores N elements inline in the class, placing the rest into a heap-allocated block which is resized as elements are added and removed. The class should support the following:

- Correct copy constructors and assignment operators
- Correct move constructors and move assignment operators (C++11)
- `push_back`, `pop_back`, `size`
- Constructing from an initializer list (C++11)
- Indexing operator `operator [] (unsigned index)`
- Compatibility with range-based for (C++11)

Q2 Define a base class for intrusive list nodes and a class for intrusive lists. Should be used as:

```
class Element : public IntrusiveNode<Element> {
  ...
};
IntrusiveList<Element> List;
List.add(new Element());
List.add(new Element());
for (Element *E : List) {
    ...
}
```

Q3 Why is it important to define destructors as virtual?

Q4 Provide two examples where the compiler can devirtualize a call to a virtual method (replace the indirection with a direct call) and two where it cannot.

2 Metaprogramming

Q1 Defined a function `template <class T, int N> T pow(T arg)` which computes arg^N.

Q2 What happens if you try to define `const int x = pow<10>(2)`?

Q3 Implement a `template <typename T> constexpr T pow(T arg, int n)`. Remember that C++14 allows for loops to be evaluated at compile time in constexpr. Isn't that lovely?

Q4 Implement a method `template<typename T> bool CheckAddUB(T a, T b, T &result)` which returns true if adding `a` and `b` results in undefined behaviour.

Hint: you will need a different implementation for signed and unsigned integers. Use `std::enable_if` and `std::is_unsigned`. You could also consider using `if constexpr` introduced in C++17.