# Performance Comparison

In this lesson, we will try a simple performance comparison test for various smart pointers.

WE'LL COVER THE FOLLOWING



A simple performance test should give an idea of the overall performance.

Run the code in the tabs below to see the performance of each pointer.

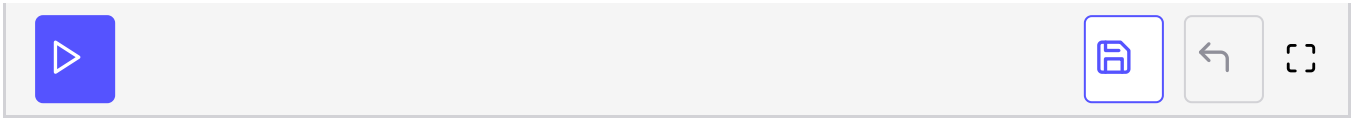
## Test Code #

The codes might take some time to execute.

native | shared_ptr | make_shared | unique_ptr | make_unique

```
// all.cpp

#include <chrono>
#include <iostream>
#include <memory>
static const long long numInt= 100000000;

int main(){

  auto start = std::chrono::system_clock::now();

  for ( long long i=0 ; i < numInt; ++i){
    int* tmp(new int(i));
    delete tmp;
    // std::shared_ptr<int> tmp(new int(i));
    // std::shared_ptr<int> tmp(std::make_shared<int>(i));
    // std::unique_ptr<int> tmp(new int(i));
    // std::unique_ptr<int> tmp(std::make_unique<int>(i));
  }

  std::chrono::duration<double> dur= std::chrono::system_clock::now() - start;
```

```
  std::cout << "time native: " << dur.count() << " seconds" << std::endl;

}
```



## Explanation #

- In this test, we compare the explicit calls of `new` and `delete` (line 13 and 14) with the usage of `std::shared_ptr` (line 15), `std::make_shared` (line 16), `std::unique_ptr` (line 17), and `std::make_unique` (line 18).
- The handling of smart pointers (line 15 - 18) is now much simpler since the smart pointer automatically releases its dynamically created `int` variable if it goes out of scope.
- The two functions `::make_shared` (line 16) and `std::make_unique` (line 18) are useful, for they create the smart pointers respectively.
- There are more memory allocations necessary for the creation of an `std::shared_ptr`. Memory is necessary for the managed resource and reference counters. `std::make_shared` makes one memory allocation out of these counters.

---

In the next lesson, we will learn how to pass smart pointers.