Thread-Safe Initialization of Data

Read only must only to be initialized in a thread-safe way.

➡ The expensive synchronization with locks is not necessary.

- C++ offers three possibilities
  - Constant expressions
  - The function `std::call_once` in combination with the `std::once_flag`
  - Static variables with block scope

Constant Expressions

- Constant expression
  - Will be initialized during compile time.
  - Can be user-defined types if they are simple enough.

```
struct MyDouble{
    constexpr MyDouble(double v): val(v){}
    constexpr double getValue(){ return val; }
private:
    double val
};

constexpr MyDouble myDouble(10.5);
std::cout << myDouble.getValue() << std::endl;
```

std::call_once and std::once_flag

The function `std::call_once` and `the flag` `std::once_flag`.

- `std::call_once` registers a callable unit.
- `std::once_flag` guarantees that only one of the registered functions will be exactly called once.

```
void initSharedDataFunction(){ ... }
std::once_flag initSharedDataFlag;

std::call_once(initSharedDataFlag, initSharedDataFunction);
```

safeInitializationCallOnce.cpp

Static Variables

The C++11 runtime guarantees that scoped static variables will be initialized in a thread-safe way.

```
void blockScope(){
    static int mySharedDataInt= 2011;
}
```

safeInitializationStatic.cpp