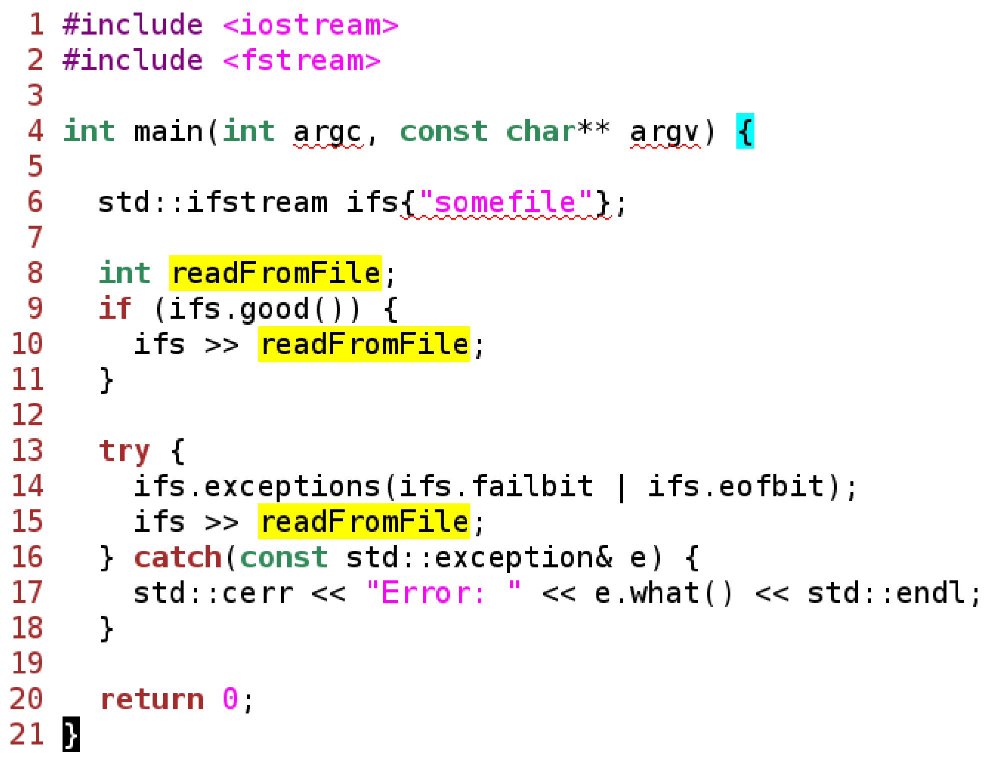
API is ART : Debate

Max 5 people discuss in front of others about an API design problem or topic.

Others look at them, and finally can propose their own ideas after the talk.

I/O in std:: C++

- Apparently the ISO comittee couldn't decide :



```cpp
#include <iostream>
#include <fstream>

int main(int argc, const char** argv) {

  std::ifstream ifs{"somefile"};

  int readFromFile;
  if (ifs.good()) {
    ifs >> readFromFile;
  }

  try {
    ifs.exceptions(ifs.failbit | ifs.eofbit);
    ifs >> readFromFile;
  } catch(const std::exception& e) {
    std::cerr << "Error: " << e.what() << std::endl;
  }

  return 0;
}
```

Error reporting/handling – 30 min

- Error Handling
  - Input/Output Errors as exceptions or as known expected state ?

- How do you return values ?
  - Out parameters
  - Returnvalue, what about error return code ?
  - Getters to check for errors ?

Recovering from errors

- Error Recovery
  - Handling cleanup of resources
  - Handling cleanup of own locks in multithreaded applications

- How would one implement graceful degradation ?
  - Nullptr checks ?
  - Non available modules ?
  - Non responding APIs (timeouts, wait for comeback...)

Object Oriented vs Functional – 15 min

- Identity based computations :
  - Typical OO, types instances have identities and states, which provide methods to operate on them
    - Java, C#, old-school C++

- Value based computations
  - Typical functional
  - Types instances are values, doesn't own identity and are operated with freestanding functions / functions objects.
    - Go, Haskell, modern C++, template metaprograms

- Could both cohabit well in a single problem domain resolution ?