Port C and C++ Course to xeus-cpp for interactive learning: Final Report

Georgi Runtolev

Mentor: Vassil Vassilev

Project Summary

> **The primary focus of this project was to fully port a classic, compiler-driven C and C++ educational curriculum into interactive Jupyter Notebooks powered by xeus-cpp.**

Unlike traditional programming environments that depend heavily on static build cycles, this implementation shifts the focus toward continuous REPL evaluation. By managing execution states dynamically, the new framework significantly lowers structural friction for students while surfacing novel challenges related to global state preservation, compilation side-effects, and runtime redefinitions.

Traditional Compilation vs. Notebook Interpretation

Traditional Execution Model

In standard software development or foundational courses, source code is evaluated as isolated monoliths. Every single execution requires:

- A global entry-point function explicitly defined as int main()
- Re-compilation of all declaration header and imports sequentially
- A completely blank stack and heap allocation state on startup

Xeus-cpp Interpretation

Xeus-cpp relies on the Clang-repl ecosystem to compile statements dynamically. This fundamentally modifies behavioral properties:

- State lifecycle: Instantiated variables and declared entities persist across cells indefinitely
- Immediate code execution

Safe Coding Patterns

- **Assignment over re-declaration:** use x = 10, not int x = 10, once a variable exists
- **Lambdas for function iteration:** test revised functions in block scope to avoid global conflicts
- **Namespace versioning:** wrap versions in namespace v1, namespace v2 to let both coexist
- **#ifndef include guards:** wrap definitions to prevent redefinition errors on re-run

```
#ifndef STUDENT
#define STUDENT

struct Student {
    std::string name;
    int grade;
};

#endif
```

Modular Notebook Structure

- A consistent section template was designed and applied across all notebooks
- 1. Title and learning objectives
- 2. Setup cell (all #includes, using declarations)
- 3. Concept introduction
- 4. Minimal working example
- 5. Extended example
- 6. Edge cases and common mistakes
- 7. xeus-cpp specific notes
- This structure enforces a correct top-to-bottom execution order, preventing state-related bugs

Lessons Learned

What Worked Well

- Building examples one cell at a time made it easy to isolate bugs
- Consistent structure across all notebooks made them predictable for students
- Visual guides to execution model and memory layout added significant pedagogical value

Challenges

- Re-running cells out of order produces different results - unintuitive for students from compiled languages
- int main() is mandatory in traditional workflow but forbidden in xeus-cpp
- Template error messages from Clang span many lines and can be overwhelming
- C and C++ together span a very large curriculum and prioritizing depth over breadth was essential

Timeline Summary

- Weeks 1–2: Environment setup and execution model documentation
- Week 3: Redefinition conflicts in xeus-cpp
- Weeks 4–5: C fundamentals and memory management notebooks
- Weeks 6–7: C++ OOP and STL/modern C++ notebooks
- Weeks 8–9: File I/O, exceptions, advanced templates
- Week 10: Modular notebook design
- Weeks 11–12: Bug fixes, syntax cleanup, notebook refinement
- Week 13: Visual guides - execution model and memory layout diagrams
- Weeks 14–15: Final report and submission