GOOGLE SUMMER OF CODE 25’

IMPLEMENTING DEBUGGING SUPPORT FOR XEUS-CPP

Author: Abhinav Kumar
Mentors: Anutosh Bhat, Vipul Cariappa, Aaron Jomy, Vassil Vassilev

ABOUT
ME

Academic Background

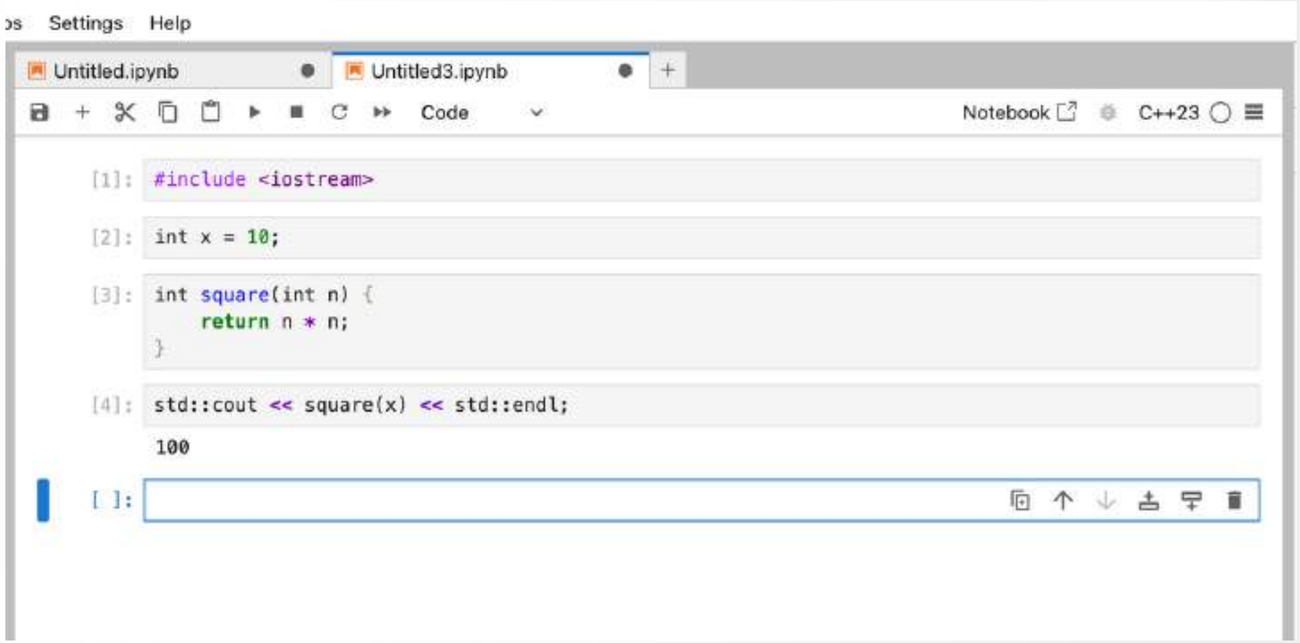
- 4th year undergraduate student at Indian Institute of Technology(IIT), Indore.
- Major in Computer Science & Engineering

Interests

- Low level programming and C++.
- Binary Exploitation.
- System Design and Software Development.
- Recently got into AI/ML.

WHAT IS XEUS-CPP?

- **Xeus-Cpp** is a Jupyter kernel that enables interactive C++ programming within the Jupyter environment.
- It is built on the Xeus library—a C++ implementation of the Jupyter kernel protocol.
- Powered by the ***Clang-Repl*** interpreter from the ***CppInterOp*** library, Xeus-Cpp allows you to write, execute in real-time, much like you would with Python.



DEBUGGING SUPPORT **FOR XEUS-CPP**

Can’t have debugger just like in xeus-python or ipykernel.

Why?

- Because, Python is an interpreted language while C++ is a compiled language.

Can’t directly use LLDB(Debugger for C++) because LLDB attaches to C++ compiled code.

So how can we debug Just-In-Time(JIT) Compiled Code??

DEBUGGING
JIT-ed CODE

Using CppInterOp library and enabling JIT loader in LLDB will resolve symbols and debug the JIT-compiled code.

```
❯ lldb ./test
(lldb) target create "./test"
Current executable set to '/Users/abhinavkumar/Desktop/Coding/Testing/test' (arm64).
(lldb) settings set plugin.jit-loader.gdb.enable on
(lldb) breakpoint set --name f1
Breakpoint 1: no locations (pending).
WARNING:  Unable to resolve breakpoint to any actual locations.
(lldb) r
Process 51881 launched: '/Users/abhinavkumar/Desktop/Coding/Testing/test' (arm64)
1 location added to breakpoint 1
In codeblock 1
Process 51881 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = breakpoint 1.1
    frame #0: 0x000000010057c008 JIT(0x10055c218)`f1() at input_line_1:4:13
(lldb)
```

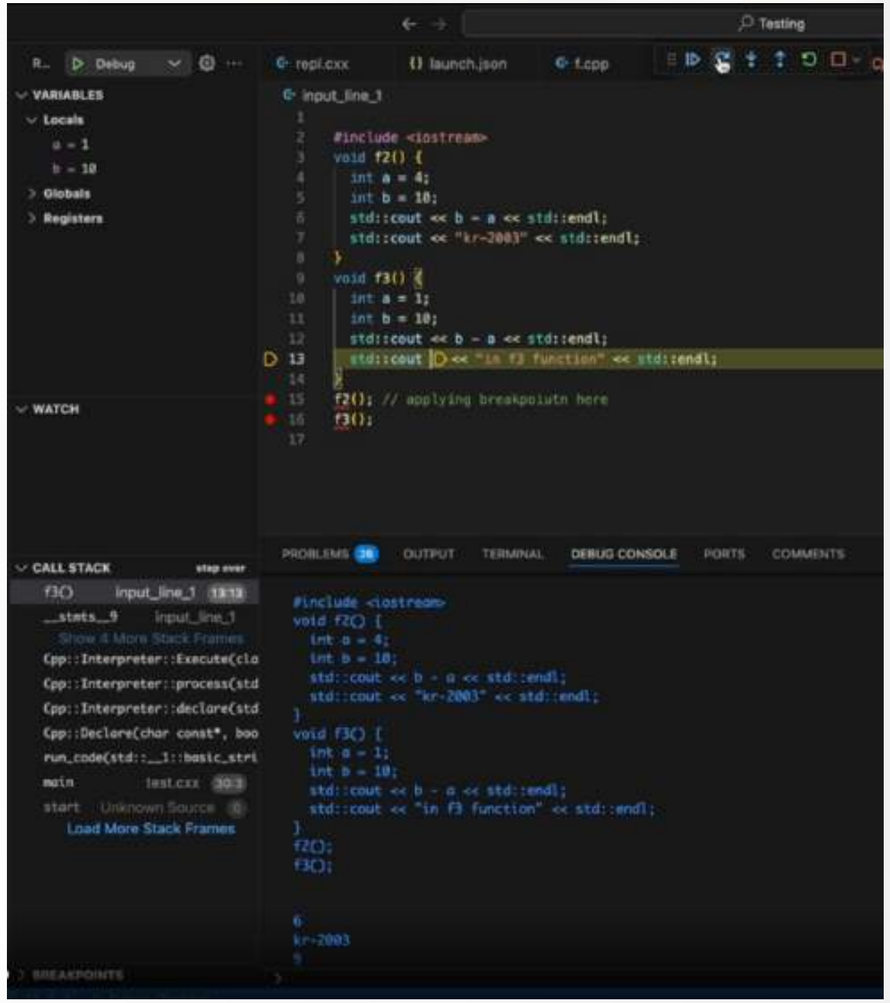
Xeus-cpp uses this structure under the hood.

```cpp
#include "clang/Interpreter/CppInterOp.h"
#include <iostream>

void run_code(std::string code) {
  Cpp::Declare(code.c_str());
}

int main(int argc, char *argv[]) {
  Cpp::CreateInterpreter({"-gdwarf-4", "-O0"});
  std::vector<Cpp::TCppScope_t> Decls;
  std::string code = R"(
#include <iostream>
void f1() {
  std::cout << "in f1 function" << std::endl;
}
std::cout << "In codeblock 1" << std::endl;
int a = 100;
int b = 1000;
  )";
  run_code(code);
  code = R"(
f1();
  )";
  run_code(code);
  return 0;
}
```

HOW CAN WE BRING THIS DEBUGGING IN **JUPYTER-LAB'S ENVIRONMENT?**

Jupyter uses **Debugger adapter protocol(DAP)**

LLDB-DAP implements the Debug Adapter Protocol (DAP) for debugging C++ with LLDB in IDEs like VS Code and JupyterLab.



Using LLDB-DAP with VS Code

IMPLEMENTATION
OVERVIEW

PHASE 1

- Extracting CppInterOp process into a standalone forked process from xeus-cpp kernel.
- Attaching lldb-dap/lldb to this CppInterOp process.
- Experimenting with kernel, lldb-dap, lldb and trying out different approaches.

PHASE 2

- JupyterLab intergration with LLDB-DAP.
- Implementing breakpoint feature.
- Implementing inspect variable feature.
- Writing unittests for above features.

IMPLEMENTATION
OVERVIEW

PHASE 3

- Implementing step-in feature and handling the multi-codeblock issue.
- Implementing step-out feature.
- Writing unittests for above features.

PHASE 4

- Overall tests for debugger.
- Refactoring, code quality review and documentation.

GOALS

- Enable interactive debugging for C++ in Jupyter notebooks with breakpoints, variable inspection, and step-through execution
- Solving the unique technical challenge of debugging JIT compiled code
- Establish a foundation for advanced debugging features

IMPACT

- This will be the first comprehensive debugging solution for C++ in Jupyter environments
- Makes C++ more accessible to students and researchers who prefer interactive development
- Enables better C++ education in academic settings where Jupyter is popular

THANK YOU