Lab 6 - Submission Instructions

Submission Instructions

Students must submit a **single compressed archive** named:

```
lab6_<rollno>.tar.gz
```

Expected Contents

The submission must include the following:

1. **Integrated System**

A single, unified system integrating work from **Labs 1–5**.
The system must manage the **complete lifecycle of a program**, including:

- program submission and execution
- parsing and metadata generation
- IR-based execution
- debugging and execution control
- memory tracking and garbage collection
  Removing any one component should **break system functionality**.

---

2. **Shell Interface**

The shell must act as the **single entry point** and support at least:

```
submit <program>
run <pid>
debug <pid>
kill <pid>
memstat <pid>
gc <pid>
leaks <pid>
```

Shell commands must trigger coordinated behavior across subsystems.

---

3. **Parser, IR, and Execution Integration**

- Programs must be parsed into an AST and lowered to a common IR.
- Debug metadata (line numbers, variable mappings, allocation sites) must flow from parsing to execution and debugging.
- Execution must happen **only through the VM**, not via native shortcuts.

---

4. **Debugger Support**

- Debugging must operate through the integrated execution framework.
- Supported features should include:
- breakpoints
- stepping
- continue / pause
- inspection of execution state
- Debugger commands must rely on metadata generated earlier in the pipeline.

---

5. **Memory Management**

- All dynamic allocations must be tracked system-wide.
- Garbage collection must:
- identify roots from execution state
- reclaim unreachable objects
- Memory statistics must be available per program (`memstat` , `leaks`).

---

6. **Source Code**

- Clean, modular integration of all components.
- Well-defined interfaces between subsystems.
- Code must compile and run using **GCC** on lab machines.
- No hard-coded assumptions or lab-specific hacks.

---

7. **Test Programs**

- Test cases demonstrating:
- correct program lifecycle handling
- debugging across execution stages
- memory allocation and GC behavior
- interaction between shell, debugger, and GC
- Provided tests may be used, but **additional tests are encouraged**.

---

8. **Technical Report**

- High-level system architecture.
- Component interfaces and data flow.
- Key design decisions and trade-offs.
- Limitations and known issues.

Clarity and reasoning are more important than length.

---

9. **README**

- Build instructions.
- How to start the shell.
- Example command workflows.
- How to run demos and test cases.

Demo and Evaluation

- Demos will be conducted **live by the TAs**.
- Students will be asked to:
- execute end-to-end workflows
- debug running programs
- demonstrate memory statistics and GC
- explain how components interact internally
- The system may be tested on **unseen and more complex test cases**.

Use of AI tools is permitted, but **students must clearly understand and explain their code**.
Inability to explain design or implementation may result in loss of marks.