

AuxFlow — Review 1 Speaker Guide

How to present the deck · Upayan Mazumder (24BDS0367) · Compiler Design Lab

Delivery rules

- **Total ~8–9 min**, then questions. If time is tight, trim Slides 3 and 7 first.
 - Point at diagrams; **do not read slides word-for-word** — the slides are cues, you supply the sentences.
 - Repeat the word **“compiler”** often — the evaluator should never doubt this is a CD project.
 - Use **“may not converge”** and **“conservative”**. Avoid “detects all”, “guarantees”, or “replaces latexmk”.
 - If unsure of an answer, restate scope: “within the restricted subset...” and stay conservative.
 - Bridge between slides with one line, e.g. after Slide 3: “...so how does this map to compiler concepts?”
-

Slide 1 — Title

~30 sec

Say:

- "Good morning. I'm Upayan; my project is AuxFlow — a compiler design project."
- "In one line: a compiler front-end for a restricted document language that analyses, statically, how a document behaves when it is compiled in multiple passes."
- "It reads source and studies dependencies — it does not typeset anything, and it is not an AI project."

Point at / emphasize:

- Say **“compiler design project”** clearly — set the frame for the phases to come.
- Keep it to ~2 sentences; don't linger on the cover.

Slide 2 — Problem

~1 min 15

Say:

- "Some documents compile in several passes, because of feedback."
- "A pass writes generated information — a section number, a contents entry — and the next pass reads it back."
- "A cross-reference can't be resolved until an earlier pass recorded it, so the passes repeat until the state stops changing."
- "The hard part is that, from the source alone, this behaviour is difficult to predict — and that is the gap my project studies."

Point at / emphasize:

- Trace the loop with your hand as you describe it.
- Do **not** say existing tools are bad — frame it as “discovered by running” vs “analysed from source”.

Slide 3 — Objectives & Scope

~1 min

Say:

- "My objectives are laid out as compiler stages, so each one is something I can build and test on its own."
- "Build the front-end, extract the cross-reference dependencies, represent them as a dependency graph, analyse that graph statically, and produce readable reports."
- "Scope is deliberately narrow: a restricted subset, dependency-only analysis, and conservative results — anything outside the subset is marked opaque and handled safely."

Point at / emphasize:

- Gesture left for objectives, right for scope.
- Stress **“restricted subset”** and **“conservative”** — they pre-empt viva questions.

Slide 4 — Why Compiler Design (MARKS SLIDE — slow down)

~1 min 30

Say:

- "This slide maps the project onto standard compiler concepts."
- "Lexical analysis is a category-code scanner that makes tokens; syntax analysis is a recursive-descent parser that builds an AST."
- "Semantic analysis uses symbol tables to extract the dependencies; those become the intermediate representation — a dependency graph."
- "Static analysis evaluates that graph with dataflow and a fixpoint, and I use Tarjan's algorithm to find cycles."
- "So every stage is a textbook compiler phase — the only difference is the target: an analysis report instead of machine code."

Point at / emphasize:

- Read each row left → right; this is where most marks are — take your time.
- Pause on **Intermediate Representation** and **Static Analysis**.

Slide 5 — Proposed Architecture

~1 min 15

Say:

- "The architecture is a single-direction pipeline — the shape of a compiler."
- "Source goes through the lexer, parser, AST, and semantic analyzer — that is the front-end."
- "It lowers into the Dependency IR, highlighted here, and the static analyzer works only on that IR to produce the report."
- "Each stage is a separate command I can demonstrate on its own, and the whole thing runs without any typesetting engine."

Point at / emphasize:

- Run a finger down the pipeline as you speak.
- Point at the two highlighted boxes (IR, Static Analyzer).

Slide 6 — Methodology

~1 min 15

Say:

- "Here is the analysis flow, left to right."
- "Extract the facts — labels and references. Link each written value to the later read that depends on it. Those links form a graph."
- "Analyse the graph — traversal, strongly connected components for cycles, and a fixpoint — then explain the result with source locations."
- "The key idea is a conservative separation: ordinary feedback fixed by structure settles; feedback where a value changes how much content exists is flagged as a risk — I never claim certainty."

Point at / emphasize:

- Trace the five stages; land on the benign-vs-risky distinction.
- Gesture at the "Techniques" strip to name the algorithms.

Slide 7 — Example

~1 min

Say:

- "A quick concrete example, read left to right."
- "A section defines a label; later the text refers to it. AuxFlow extracts two facts and one dependency: the reference can only be filled in one pass after the label is written."
- "There is no cycle here, so the output is a plain sentence naming the dependency and where it comes from."
- "The point is that the tool explains the dependency — it does not just say 'cycle detected'."

Point at / emphasize:

- Point at the LABEL → REF_USE arrow and the "one pass later" label.
- Keep it short — it is an illustration, not a proof.

Slide 8 — Conclusion

~45 sec

Say:

- "To summarise: AuxFlow is a compiler project end to end — lexer, parser, AST, symbol tables, an IR, and dataflow and graph analysis."
- "Its contribution is a dependency-based static analysis of multi-pass compilation, with explainable, conservative output."
- "It is feasible for one student because it is built phase by phase, each independently testable. Thank you — I am happy to take questions."

Point at / emphasize:

- End on the pipeline strip; invite questions confidently.
- Do not trail off — finish on "happy to take questions".