perplexity

# Make the same roadmap for making a compiler for cpp to showcase my understanding of theory of computations and c++.

Here's a **detailed 7-day roadmap** for building a simple C++ compiler project to showcase your understanding of Theory of Computation and C++. This plan leverages high-quality tutorials, YouTube videos, and GitHub repos so you can follow along and complete the project efficiently.

### 7-Day Roadmap: Building a Simple C++ Compiler

### Day 1: Understand Compiler Structure & Set Up Environment

- **Goal:** Grasp compiler phases and set up your C++ development environment.
- **Resources:**
  - Tutorialspoint: Compiler Design Tutorial[1]
  - How to Build a Compiler from Scratch | Full Guide (YouTube)[2]
  - ssloy/tinycompiler GitHub[3]
- **Tasks:**
  - Read about compiler phases: lexical analysis, parsing, semantic analysis, code generation[1] [4].
  - Install a C++ IDE (VSCode, CLion, or Visual Studio).
  - Set up a GitHub repo for your project.

### Day 2: Implement Lexical Analysis (Lexer/Scanner)

- **Goal:** Convert source code into tokens (keywords, identifiers, literals).
- **Resources:**
  - Compiler Design Tutorial: Lexical Analysis[1]
  - How to Build a Compiler from Scratch | Full Guide (YouTube) — Lexer section[2]
  - ssloy/tinycompiler GitHub — Lexer implementation[3]
- **Tasks:**
  - Write a C++ function to read source code and break it into tokens.
  - Handle whitespace, comments, and different token types.
  - Print tokens for a sample input to verify correctness.

### Day 3: Implement Syntax Analysis (Parser)

- **Goal:** Build a parser to convert tokens into a parse tree (AST).
- **Resources:**
  - Compiler Design Tutorial: Syntax Analysis[1]
  - How to Build a Compiler from Scratch | Full Guide — Parser section[2]
  - ssloy/tinycompiler GitHub — Parser implementation[3]
- **Tasks:**
  - Define grammar rules for your subset of C++ (e.g., variable declarations, assignments, arithmetic, print).
  - Implement a recursive descent parser in C++.
  - Output the parse tree or AST for test programs.

### Day 4: Implement Semantic Analysis

- **Goal:** Check for semantic errors (type checking, undeclared variables).
- **Resources:**
  - Compiler Design Tutorial: Semantic Analysis[1]
  - How to Build a Compiler from Scratch | Full Guide — Semantic Analysis[2]
- **Tasks:**
  - Traverse the AST to check for type errors and variable declarations.
  - Report semantic errors with clear messages.

### Day 5: Intermediate Code Generation

- **Goal:** Generate an intermediate representation (IR) or simple assembly code.
- **Resources:**
  - How to Build a Compiler from Scratch | Full Guide — Code Generation[2]
  - ssloy/tinycompiler GitHub — Code Generation[3]
  - How to Build a C++ Compiler: A Modern Guide[4]
- **Tasks:**
  - Translate AST nodes to IR or assembly instructions.
  - Output the generated code to a file.

### Day 6: Assemble, Link, and Run

- **Goal:** Convert the generated code to an executable and test it.
- **Resources:**
  - A Compiler For Our Own Programming Language // Full Guide (YouTube)[5]
  - Let's Create a Compiler (YouTube)[6]
- **Tasks:**
  - Use an assembler (e.g., NASM for x86) to assemble your code.
  - Link the object file using a linker (e.g., GCC).
  - Run the executable and verify output for sample programs.

### Day 7: Polish, Document, and Prepare Resume Entry

- **Goal:** Clean up code, write documentation, upload to GitHub, and craft a resume bullet.
- **Resources:**
  - How should I list a compiler project in my resume? (Reddit)[7]
  - Compiler Resume Template[8]
- **Tasks:**
  - Refactor and comment your code.
  - Write a detailed README: explain the language subset, design choices, and compilation phases.
  - Upload to GitHub with sample code and outputs.
  - Prepare a concise, quantifiable resume bullet (see below).

### Best YouTube Video Tutorials

- How to Build a Compiler from Scratch | Full Guide — covers lexer, parser, semantic analysis, and code generation in detail[2].
- Let's Create a Compiler (Pt.1) — practical C++ project walkthrough[6].
- A Compiler For Our Own Programming Language // Full Guide — step-by-step compiler build, including assembly and linking[5].

### Example Resume Entry

> **Built a simple C++-like language compiler in C++**: Implemented all major phases (lexical analysis, parsing, semantic analysis, code generation). Used recursive descent parsing and generated x86 assembly, assembled and linked to produce executables. Demonstrated strong grasp of Theory of Computation and compiler construction. [GitHub link]

**Tips:**

- Quantify features (e.g., “Supports variables, arithmetic, conditionals, and print statements”).
- Mention tools (C++, NASM, GCC).
- Link your GitHub repo.

**Follow this roadmap and you’ll have a robust, well-documented compiler project that highlights both your theoretical and practical skills in C++ and computation theory.**

⁂


1. https://www.tutorialspoint.com/compiler_design/index.htm
2. https://www.youtube.com/watch?v=HOe2YFnzO2I
3. https://github.com/ssloy/tinycompiler
4. https://hub.paper-checker.com/blog/building-a-c-compiler-insights-and-techniques-from-concept-to-implementation/
5. https://www.youtube.com/watch?v=GsCWivTeFpY
6. https://www.youtube.com/watch?v=vcSijrRsrY0
7. https://www.reddit.com/r/csMajors/comments/rjbyhw/how_should_i_list_a_compiler_project_in_my_resume/
8. https://guide.resumegemini.com/resumesamples/compiler-resume-template/