A. Core C fundamentals

You need to be solid in:

1. Pointers

- **pointer arithmetic**
- **pointer to structs**
- **pointer to functions**
- **double pointers**

Source:
• ***"Pointers in C" – CodeWithHarry (YouTube)***
• ***Learn-C.org – Pointer section***

---

2. Memory management

- **malloc, calloc, realloc, free**
- **dynamic arrays**
- **manual memory lifetime**

Source:
• ***freeCodeCamp – C Dynamic Memory tutorial (YouTube)***

---

3. Strings & character buffers

- **null-terminated strings**
- **reading/writing character-by-character**
- **char arrays**
- **token buffering**

Source:
• ***Beej's Guide to C Programming – Strings***

---

4. Structs (extremely important)

- **structs with multiple fields**
- **nested structs**

- **arrays of structs**

Your AST, tokens, and symbol tables will all be structs.

Source:
• *Learn-C.org – Structs*

---

5. Linked lists and dynamic data structures

Your token list, AST nodes, and symbol tables often use dynamic structures.

Source:
• *K&R C Book – Linked Lists chapter*
• *freeCodeCamp – C linked lists video*

---

6. Recursion

Your parser will use recursion heavily (recursive descent parser).

Source:
• *All above sources cover recursion well*

---

7. File I/O

Your compiler needs to:

- **open source file**
- **read characters**
- **write assembly output**

Source:
• *Programiz – "C File Handling"*

---

8. Modular programming (headers + .c files)

Your compiler will have many .h/.c files
(lexer.c, parser.c, ast.c, emit_x86.c, main.c)

Source:
• *Beej's Guide to C – header files section*

---

9. Hash tables (optional but recommended)

For symbol table (variables, functions, scopes).

Source:
- ***"How to implement Hash Table in C" – Jacob Sorber (YouTube)***

---

10. Basic math

- **operator precedence**
- **integer operations**
- **binary vs decimal**
- **how CPUs handle registers**

Source:
- **MIPS or x86 tutorials on YouTube**

x86-64 topics you must learn (with sources)

Since you chose **Option 1: x86-64**, you'll generate real assembly.

Learn these:

1. x86-64 registers

- RAX, RBX, RCX, RDX
- RDI, RSI
- RSP, RBP

Source:
• *"x86-64 Assembly Crash Course" – Jacob Sorber*

---

2. Stack + stack frames

Used for:

- variables
- function calls
- return values

Source:
• *"Programming with Mosh – Assembly for Beginners"*
• *CS 61 – Calling conventions lecture*

---

3. Syscalls or calling the C library

Your print() can call printf.

Source:
• *GNU ABI x86-64 calling convention docs*

---

4. NASM assembler basics

You will produce .asm output, so learn:

- mov
- add
- sub
- imul
- idiv
- push/pop
- call/ret

Source:
• *Official NASM documentation*

---

5. Linking object files

Use:

nasm -f elf64 code.asm

ld code.o -o output

Source:
• Tutorials on “Writing x86-64 in NASM”

Compiler theory topics you must learn (with sources)

You don’t need everything — just these:

1. Lexing (Tokenization)

Break source → tokens.

Source:
• *"Write a lexer in C" – Tsoding (YouTube)*

---

2. Parsing (Recursive Descent)

Turn tokens into AST.

Source:
• *"Write a parser in C" – Tsoding*
• *"Writing a Recursive Descent Parser" – YouTube*

---

3. AST (Abstract Syntax Tree)

Nodes like:

- NumberNode
- BinaryOpNode
- AssignmentNode

Source:
• AST tutorials (any language)
• *Crafting Interpreters (AST chapter)*

---

4. Code generation

Turn AST → assembly.

Source:
• *"Write a compiler from scratch – Jack Crenshaw"*
• *Tsoding's compiler series (C + x86)*

---

4. FIRST step you should take

Here's the exact first step — don't jump ahead.

✓ Step 1: Build a character reader + basic tokenizer (lexer)

Start extremely small.

Create a C program that:

1. Opens a file
2. Reads characters
3. Groups them into tokens:
    - identifiers
    - numbers
    - operators (+ - * / =)
    - parentheses
    - semicolons
4. Prints each token to the screen.