一、核心结论（一句话版）

> **凡是会出现在语法产生式中的词法单元，都是“公用核心 Token”**；
> 注释、换行、调试用 Token 不属于核心。

二、类 C 语言的【公用核心 Token】完整清单

1. ① 控制与结束（必须）

Token	作用
ENDFILE	输入结束标志
ERROR	词法错误（全模块统一处理）

Table 1

📌 Parser、主控流程、错误系统都依赖

2. ② 标识符与字面量（必须）

Token	说明
ID	变量名 / 函数名
NUM	整型常量（可扩展 FLOAT）

Table 2

📌 语义分析、符号表、中间代码都依赖

3. ③ 关键字（控制流 + 类型，必须）

3.1 控制流关键字

Token	语义
IF	条件
ELSE	分支
WHILE	循环
RETURN	返回

Table 3

3.2 类型关键字（最小子集）

Token	语义
INT	整型
VOID	无返回值

Table 4

📌 **关键字必须在 Lexer 阶段区分，不能当 ID**

---

4. ④ 运算符 Token（表达式核心，必须）

4.1 算术运算符

Token	符号
PLUS	`+`
MINUS	`-`
MULT	`*`
DIV	`/`

Table 5

4.2 关系 / 比较运算符

Token	符号
GT	`>`
LT	`<`
GTE	`>=`
LTE	`<=`
EQ	`==`
NEQ	`!=`

Table 6

4.3 赋值运算符

Token	符号
ASSIGN	`=`

Table 7

📌 这些 Token **直接决定四元式 / 三地址码中的 op**

---

5. ⑤ 界符与结构符号（语法结构核心，必须）

Token	符号	作用
LPAREN	`(`	表达式 / 参数
RPAREN	`)`	表达式 / 参数
LBRACE	`{`	语句块
RBRACE	`}`	语句块
SEMI	`;`	语句结束
COMMA	`,`	参数 / 变量分隔

Table 8

📌 没有这些，Parser 根本跑不起来

三、【不是】公用核心 Token（明确排除）

你**可以有，但不要列入"核心规范"**：

Token	原因
注释（//, /* */)	Parser 可忽略
NEXTLINE	仅用于行号
空白符	Lexer 内部处理
调试 Token	不参与语法

Table 9

👉 **这些不写进语言规格，只写进 Lexer 实现说明**

四、最小“类 C 编译器核心 Token 集”（推荐你直接采用）

如果你现在要写一个**干净、标准、好讲的类 C 编译器**，
我建议你对外声明的核心 Token 只有下面这些 👇

```
【Core Tokens for C-like Compiler】

控制:
  ENDFILE, ERROR

标识与常量:
  ID, NUM

关键字:
  IF, ELSE, WHILE, RETURN, INT, VOID

算术运算符:
  PLUS, MINUS, MULT, DIV

关系运算符:
  GT, LT, GTE, LTE, EQ, NEQ

赋值:
  ASSIGN

界符:
  LPAREN, RPAREN, LBRACE, RBRACE, SEMI, COMMA
```

Fence 1

📌 **这就是“类 C 语言最小可编译子集”的 Token 集**

五、你作为语言设计者必须立的 3 条铁律

5.1 铁律 1

> **Lexer 只返回上述 Token，任何新 Token 必须全员讨论**

5.2 铁律 2

> **Parser 只能依赖 TokenType，不解析字符串**

5.3 铁律 3

> **中间代码的 op 必须与运算符 Token 一一对应**

六、你可以在设计文档中这样写（直接可用）

> 本编译器采用类 C 语言的最小核心 Token 集合作为词法与语法分析的公共规范，该 Token 集涵盖了控制流、类型、表达式和语句结构所需的全部基本元素，确保了各模块之间接口的一致性和可扩展性。