🧩 一、先了解：Linux 中的输入输出流（I/O Stream）

每个程序运行时，系统默认会为它打开三个“数据通道”（称为文件描述符）：

文件描述符	名称	默认连接对象	作用
`0`	标准输入（**stdin**）	键盘	读取输入
`1`	标准输出（**stdout**）	屏幕	输出正常信息
`2`	标准错误（**stderr**）	屏幕	输出错误信息

🧭 二、`<`：输入重定向

`<` 的作用是**把文件内容作为程序的输入**，即把文件“接”到标准输入流（stdin）上。

🔹 语法

```
命令 < 文件名
```

🔹 示例 1：

```
cat < example.c
```

等价于：

```
cat example.c
```

意思是：

> 把 `example.c` 的内容“输入”给 `cat` 程序，`cat` 再把内容输出到屏幕。

🔹 示例 2：

在你的 Flex 分析器中：

```
./lexer < example.c
```

意思是：

> 不用键盘输入，而是让程序从 `example.c` 读取内容作为输入流。
>
> 对 Flex 而言，这就像你在键盘上输入 `example.c` 的所有内容一样。

⚙️ 三、`>`：输出重定向

`>` 的作用是**把程序的输出结果保存到文件中**，而不是显示在屏幕上。

🔹 语法

```
命令 > 文件名
```

如果文件存在，会被**覆盖**；如果文件不存在，会**新建**。

🔹 示例 1：

```
ls > list.txt
```

意思是：

> 把 `ls` 命令的输出内容写入文件 `list.txt`，屏幕上不会显示。

🔹 示例 2：

在你的词法分析器中：

```
./lexer < example.c > result.txt
```

意思是：

> 从 `example.c` 读入内容（通过 `<`）
> 把分析输出写入到 `result.txt`（通过 `>`）

这样 `result.txt` 中就会保存所有分析结果。

🧠 四、扩展：常见变体

符号	含义	示例
`>`	输出重定向（覆盖）	`ls > out.txt`
`>>`	输出重定向（追加）	`echo hello >> log.txt`
`<`	输入重定向	`./lexer < example.c`
`2>`	错误输出重定向	`gcc test.c 2> errors.log`
`&>`	标准输出+错误输出一起重定向	`make &> build.log`

📘 五、类比理解（形象比喻）

可以把 `<` 和 `>` 想象成"数据流管道"的方向箭头：

```
文件 ----<---- 程序 ----->---- 文件
```

- `<` 表示 **"从文件流入程序"**
- `>` 表示 **"从程序流出到文件"**

✅ 六、总结表

符号	方向	作用	默认连接对象
`<`	文件 → 程序	输入重定向	stdin
`>`	程序 → 文件	输出重定向（覆盖）	stdout
`>>`	程序 → 文件	输出重定向（追加）	stdout
`2>`	程序 → 文件	错误重定向	stderr

💡 **你在执行：**

```
./lexer < example.c
```

时，实际上等价于：

```
cat example.c | ./lexer
```

意思是：

> "把 example.c 的内容通过管道（或输入流）送进 lexer 的标准输入"。