setjmp() & longjmp() — Notes on Program Flow

1. Objective of the Program

This program demonstrates **non-local jumps** in C using `setjmp()` and `longjmp()`. They allow control to jump from one function back to another execution point.

- `setjmp()` saves the current program state (stack + registers)
- `longjmp()` restores that saved state and forces execution to resume at `setjmp()`

This mechanism is mainly used for **error handling / recovery paths**.

---

2. Role of setjmp()

```
int v = setjmp(buff);
```

What happens:

- When called the first time → returns **0**
- When control returns via `longjmp()` → returns the value passed to `longjmp()`

So the variable `v` helps us identify **how** execution reached that line.

---

3. Role of longjmp()

Inside `abc()`:

```
longjmp(buff, 2);
```

Effects:

- Restores the saved context stored in `buff`
- Jumps back to the line containing `setjmp()` in `main()`
- Makes `setjmp()` return **2** instead of 0

Statements after `longjmp()` in `abc()` do **NOT execute**.

---

4. Step-by-Step Execution Flow

1. Program enters `main()`
2. `setjmp(buff)` executes
3. First call → returns 0
4. Value of `v` is printed
5. `abc()` is called
6. Inside `abc()` → `longjmp(buff, 2)` executes
7. Control jumps back to `setjmp()` in `main()`
8. `setjmp()` now returns **2**
9. Program continues from that point
10. `end of the program` is printed

The remaining lines in `abc()` are skipped.

---

5. Conceptual Output Sequence

```
inside the main program
the value returned by setjmp is 0
inside the function abc
the value returned by setjmp is 2
end of the program
```

---

6. Important Notes

- `longjmp()` bypasses normal function return flow
- It does **not** return to `abc()`
- It directly resumes execution at `setjmp()`
- Must be used carefully because it skips stack unwinding

These notes summarize how `setjmp()` and `longjmp()` work together to perform a controlled jump between functions.