Nested Menus

Why We Avoid Nested Menus with while Loops

1. **Complexity grows quickly**
   Each nested loop adds another layer of control flow. Once you have 2–3 levels, it becomes hard to trace “which loop am I in?” and “how do I exit cleanly?”
2. **Difficult to maintain**
   If you want to add or remove menu options later, you might have to restructure multiple loops. This creates duplication and fragile code.
3. **Harder to navigate backwards**
   Menus usually require a "go back" option. With deeply nested loops, backing out gracefully can require lots of break statements or flags, making code messy.
4. **Scalability problem**
   A real application might have many menus and submenus. Nested loops don’t scale — you’d end up with deeply indented code that’s unreadable and inflexible.

Why It’s Still a Good Example

1. **Clear visualization of program flow**
   Students can *see directly* how selecting one option leads to another level of looping. It maps naturally onto the mental model of menus within menus.
2. **Structure drives solution**
   The code’s structure *is* the design: outer loop = main menu, inner loop = submenu. This helps beginners connect program control flow to user experience.
3. **Teachable stepping stone**
   It sets the stage for showing better designs (like a **menu stack** or **function dispatch system**) by first illustrating the “brute force” way.

✅ **Summary**:
We generally don’t use nested menus in production code because they become hard to maintain, debug, and extend. But as a teaching tool, they show how program structure can directly create solutions, and they make the evolution to more elegant approaches easier to appreciate.