ECE 811 – SOFTWARE ENGINEERING

STRUCTURED PROGRAMMING – BEGINNERS'S STUDY GUIDE/REVISION

---

1. INTRODUCTION TO STRUCTURED PROGRAMMING

- **Definition**

Structured programming is a programming paradigm that emphasizes organizing code into logical blocks or modules, using control flow constructs like sequencing, selection (if/else), and iteration (loops), to improve code clarity, reliability, and maintainability.

Structured programming discourages the use of GO TO statements and promotes the use of functions or subroutines for modularity.

- **Historical Context:**
  - Developed in the 1960s (Dijkstra, Böhm, Jacopini).
  - Response to "spaghetti code" in early programming (GOTO statements).

- **Core Principle:**

Top-down design - break problems into smaller, manageable modules/functions.

- **Structured Programming Languages**

While any language can be used in a structured manner, some languages are designed with features that support and encourage structured programming practices.

Examples include Pascal, Ada, C, C++, Java, and Python.

2. CORE CONTROL STRUCTURES IN STRUCTURED PROGRAMMING

Three fundamental building blocks:

1. **Sequence**
   - Linear execution of statements in order.
     Example:

     python

     ```
     a = 5
     b = 10
     sum = a + b  # Executes line-by-line
     ```

2. **Selection (Decision-Making)**
   - Choose paths with if, else, switch.
     Example:

c

```
if (score >= 90) {
  grade = 'A';
} else if (score >= 80) {
  grade = 'B';
} else {
  grade = 'F';
}
```

3. **Iteration (Loops)**
    - Repeat actions with for, while, do-while.
      Example:

```
java
for (int i = 0; i < 5; i++) {
  System.out.println(i);   // Prints 0 to 4
}
```

3. KEY PRINCIPLES OF STRUCTURED PROGRAMMING

1. **Single Entry/Single Exit**: Each control structure has one entry and one exit point (no goto).
2. **Modularity**: Divide programs into functions/procedures.
    - Each module should:
        - Perform one specific task
        - Be reusable
        - Be independently testable
3. **Hierarchy**: Organize modules in layers (high-level → low-level details).
4. **Local Variables**: Limit variable scope to where they're used.

4. BENEFITS OF STRUCTURED PROGRAMMING

- **Readability**: Code is easier to understand and debug.
- **Maintainability**: Changes affect isolated modules.
- **Reduced Errors**: 50-90% fewer bugs vs. unstructured code (historical studies).
- **Reusability**: Functions can be repurposed.
- **Verifiability**: Easier to prove correctness mathematically.

5. STRUCTURED VS. UNSTRUCTURED PROGRAMMING

FEATURE	STRUCTURED	UNSTRUCTURED
Control Flow	if, loops, functions	GOTO jumps
Readability	High (linear flow)	Low (jumps create tangles)
Debugging	Easier (predictable paths)	Harder (unpredictable paths)
Modularity	Enforced	Ad-hoc
Example	C, Java, Python	Early BASIC, Early FORTRAN, Assembly

6. STEP-BY-STEP PROBLEM SOLVING

1. **Understand the problem**: Define inputs/outputs.
2. **Top-down design**: Break into sub-problems.
3. **Pseudocode**: Outline logic in plain English.
4. **Implement modules**: Write functions for each sub-problem.
5. **Test incrementally**: Validate each module before integration.

Example: Calculate Factorial
Pseudocode:

```
function factorial(n):
  if n <= 1 return 1
  else return n * factorial(n-1)
```

Python Implementation:

```
def factorial(n):
    if n <= 1:
        return 1
    else:
        return n * factorial(n-1)
```

7. BEST PRACTICES

- **Avoid Deep Nesting**: Max 3-4 levels of if/loop nesting.
- **Function Length**: Keep functions short (< 30 lines).
- **Meaningful Names**: Use calculateTax() instead of func1().
- **Comments**: Explain *why*, not *what* (code should be self-documenting).
- **Error Handling**: Validate inputs, handle edge cases.

8. COMMON PITFALLS TO AVOID

1. **Global Variables**: Cause unintended side effects.
2. **Long Functions**: Hard to debug/reuse.
3. **Nested Loops**: Can often be split into functions.
4. **Ignoring Edge Cases**: Test with 0, negative numbers, empty inputs.

9. STUDY TIPS

- **Flashcards**:
  Front: "What are the 3 control structures?"
  Back: Sequence, Selection, Iteration
- **Diagram Flowcharts**: Map out program logic visually.
- **Code Review**: Analyse open-source projects (e.g., GitHub) for structure.
- **Practice**: Solve problems on LeetCode/HackerRank using structured design.

10. Sample Exam Questions

1. *Convert this unstructured code to structured*:

   Unstructured - BASIC

   ```
   10 INPUT X
   20 IF X > 50 GOTO 50
   30 PRINT "FAIL"
   40 GOTO 60
   50 PRINT "PASS"
   60 END
   ```

   Structured Solution: Python

   ```
   x = int(input())
   if x > 50:
       print("PASS")
   else:
       print("FAIL")
   ```

2. *Why is modularity important?*
   → Isolates errors, enables reuse, simplifies collaboration.

3. *Write a structured function to find max in a list*:

   Using C language:

   ```
   int findMax(int arr[], int size) {
     int max = arr[0];
   ```

```
  for (int i = 1; i < size; i++) {
    if (arr[i] > max) {
      max = arr[i];
    }
  }
  return max;
}
```