

What Makes Program Reasoning Harder

19CSE205 : PROGRAM REASONING

Dr. Swaminathan J

Assistant Professor

Department of Computer Science and Engineering



Jul - Dec 2020

- 1 Mistakes that make reasoning harder
- 2 Spaghetti code
- 3 Side effects
- 4 Duplicate code
- 5 Unconditional jumps
- 6 Programming by permutation
- 7 Too many variables
- 8 Redundant code
- 9 Lack of modularity
- 10 Lack of indentation

Beginners often commit some common mistakes that make reasoning of programs harder. These mistakes later turn into deeply ingrained habits that become doubly harder to unlearn.

- Spaghetti code
- Side-effects
- Duplicate code
- Unconditional jumps
- Programming by permutation
- Too many variables
- Redundant code
- Lack of modularity
- Lack of indentation

Note:

- This list is not exhaustive. You may refer to various resources in the web for tips for healthy programming.
- The scope of this list is based on the fact that you have learnt only C language.

Beginners often commit some common mistakes that make reasoning of programs harder. These mistakes later turn into deeply ingrained habits that become doubly harder to unlearn.

- Spaghetti code
- Side-effects
- Duplicate code
- Unconditional jumps
- Programming by permutation
- Too many variables
- Redundant code
- Lack of modularity
- Lack of indentation

Note:

- This list is not exhaustive. You may refer to various resources in the web for tips for healthy programming.
- The scope of this list is based on the fact that you have learnt only C language.

Bottomline: Badly written code \Rightarrow Difficult to reason

Refers to **long code** with no or **little software structure**. They are difficult to comprehend and maintain. This kind of coding style is more common among beginners.

- **Spaghetti code**
- Side-effects
- Duplicate code
- Unconditional jumps
- Programming by permutation
- Too many variables
- Redundant code
- Lack of modularity
- Lack of indentation

Recommendation

- Decompose code into smaller meaningful units (functions).
- Each function must implement exactly one functionality.
- Functions can delegate part of their job to other functions thus forming a hierarchical structure.

Refers to modifying state of a variable outside its local environment.
i.e. an observable effect besides the main effect. In the presence of side effects, a program's behaviour may depend on history.

- Spaghetti code
- Side-effects
- Duplicate code
- Unconditional jumps
- Programming by permutation
- Too many variables
- Redundant code
- Lack of modularity
- Lack of indentation

```
int y = x++;
```

assign + increment

```
int a = 0;  
int foo(int b) {  
    return b*b + a;  
}  
void bar() {  
    scanf("%d", &a);  
}
```



foo's return value depends on global variable **a**. For same **b**, **foo** returns different values influenced by **bar**.

Also known as **copy-paste** code, refers to a statement or a block of statements that is replicated at multiple sections in the code. Any change or correction in one must be carried out in each replica.

- Spaghetti code
- Side-effects
- **Duplicate code**
- Unconditional jumps
- Programming by permutation
- Too many variables
- Redundant code
- Lack of modularity
- Lack of indentation

Recommendation

- Mantra: Define once and reuse as many times necessary.
- Replace occurrences of an oft used value by a variable.
- Define a function for oft performed computation and call it whenever required.

Use of **goto** makes the control jump to random points in the code. This severely makes the reasoning difficult.

- Spaghetti code
- Side-effects
- Duplicate code
- **Unconditional jumps**
- Programming by permutation
- Too many variables
- Redundant code
- Lack of modularity
- Lack of indentation

Program using goto

```
int a, b;  
a = 1;  
L1: a++;  
b = 2 * a;  
if (b > 100)  
    goto L2;  
goto L1;  
L2: printf("%d",b)
```


Refers to the approach of writing program with poor understanding and making incremental changes (try different permutations) to test for desired behavior. It gives no assurance of program quality.

- Spaghetti code
- Side-effects
- Duplicate code
- Unconditional jumps
- Programming by permutation
- Too many variables
- Redundant code
- Lack of modularity
- Lack of indentation

Known by other names

- Programming by accident
- By-try programming
- Shotgunning
- Trial-and-error
- Poke-and-hope
- Bird-shot method
- Million monkeys style

Use of too many variables usually means increased interdependencies between them. As variables change their states often, they affect each other making analysis complex.

- Spaghetti code
- Side-effects
- Duplicate code
- Unconditional jumps
- Programming by permutation
- **Too many variables**
- Redundant code
- Lack of modularity
- Lack of indentation

Recommendation

- Avoid using global variables.
- Decompose program into several smaller functions.
- Each function with its local variables implements a single functionality.

Refers to code that will not be executed (unreachable), no more in use (dead) or repeat computation (recomputation). These are more of a distraction.

- Spaghetti code
- Side-effects
- Duplicate code
- Unconditional jumps
- Programming by permutation
- Too many variables
- **Redundant code**
- Lack of modularity
- Lack of indentation

- `#define` causes repeat code
`#define min(x,y) x<y ? x : y`

Modularity refers to code that is partitioned into logical units where each unit implements a set of closely interrelated functionalities and interact with other units by well-defined interfaces.

- Spaghetti code
 - Side-effects
 - Duplicate code
 - Unconditional jumps
 - Programming by permutation
 - Too many variables
 - Redundant code
 - Lack of modularity
 - Lack of indentation
- Units can be objects, modules, files, packages or libraries.
 - Units are hierarchially arranged to build layers of abstractions.
 - Care should be taken to ensure a unit does not implement unrelated functionality.
 - Note, for I/O operations you included `stdio.h`, for Math operations, `math.h`.

Indentation helps better convey the structure of a program to human readers and therefore easy to comprehend and analyze. However, automated reasoning does not benefit from indentation.

- Spaghetti code
- Side-effects
- Duplicate code
- Unconditional jumps
- Programming by permutation
- Too many variables
- Redundant code
- Lack of modularity
- Lack of indentation

For a comprehensive discussion on indentation of C programs you may please refer to

https://www2.cs.arizona.edu/mc-cann/indent_c.html.