Mistakes That Make Reasoning Harder

19CSE205 : PROGRAM REASONING

Dr. Swaminathan J

Assistant Professor

Department of Computer Science and Engineering

Jul - Dec 2020

Contents

Mistakes that make reasoning harder

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.

Lets take a closer look at these mistakes and how to deal with them.

Spaghetti code

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.

Side effects

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

Result depends on prior value of global variable x

```
int x;
int incr() {
    return ++x;
}
```

⇒

For an x, incr will return exactly same value always

```
// x is in local env now
int incr(int x) {
    return ++x;
}
```

Global variable s is changed by sqr

```
int s;
void sqr(int v) {
    s = v*v;
}
```

⇒

Global variable s assigned the result of sqr

```
int s = sqr(v);
int sqr(int v) {
    return v*v;
}
```

Duplicate code

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.

Unconditional jumps

Use of goto makes the control jup to random points in the coded. 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

```
goto label;
...
...
label: statement;
```

Programming by permutation

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

Too many variables

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.

Redundant code

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

Lack of modularity

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.

Lack of indentation

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 compehensive discussion on indentation of C programs you may please refer to https://www2.cs.arizona.edu/ mc-cann/indent_c.html.