CSc 28
Discrete Structures

Chapter 13
Side Effects

Herbert G. Mayer, CSU CSC

Status 4/1/2021

Syllabus

- Side Effects
- Languages and Side Effect
- Sample Side Effect
- References

Side Effects

Side Effects **of an executing imperative program are:**

- **Actions performed by a function**
- **Caused by changes of the program state**
- **In a way that is at times surprising**
- **Due to their unnatural location or time! – careful: loaded term!**
- **As a consequence, side effects are hard to track, their impact difficult to account for, and programs generating them hard to maintain**

What *Side Effects* are NOT!

- **These changes are not caused by external agents, aside from programmer; instead you/the programmer put them there for a reason, sometimes good ones**
- **Side effects are not bad (poor programming) per se!**
- **Functional languages are designed to be side effect free!**

Side Effects

- **Why do Side Effects occur? Why are they practiced?**
  - **For convenience lumped together with action proper of a function**
  - **For efficiency, to avoid invoking another, separate function**
- **How to avoid Side Effects?**
  - **Either use a functional language, or**
  - **Assuming the impact of the Side Effect is needed:**
  - **Separate the Side Effect out in an imperative language, thus simulating the behaviour/constraint of a functional language**
- **Why avoid Side Effects?**
  - **A program written with Side Effects, no matter how well designed, is often more difficult to read --harder to maintain– than a program without Side Effects**
  - **But watch out for "snake-oil" propaganda ☺**
  - **The work of the Side Effect has to be accomplished somehow**

Language and Side Effect

Imperative languages (e.g. C++) encourage side effects

- Underlying compute model is Turing Machine
- Side Effect means: a change in machine/program state that is hard to see
- Enabled by references to, and changes of, global objects
- Can change global objects directly, or indirectly via reference parameters or pointers
- Can change files, state, condition codes, any machine resource
- Languages prone to side effects: Fortran, PL/I, Pascal, Ada, C, C++
- Also pointer type objects enable programming with side effects!

Language and Side Effect

Can be more obscure in languages other than C/C++

- More complex languages with nested procedural scope offer even more chances for side effects; C has no lexically nested functions
- E.g. PL/I, Ada, Pascal, Modula-2, Algol-60, Algol68, …

Goal of Functional languages: Avoid Side Effects

- E.g. Haskell, ML, Prolog

Caveat

- Functional language are not inherently better, just because they reduce chances for side effects
- Imperative languages with nested procedural scope are not inherently better, just because they enable easy data sharing

Sample Side Effect

```
#include <stdio.h>
#define MAX 10       // arbitrary array bound

int global_i = 5;    // arbitrary number within array bounds
int arr[ ] = { 0, 1, 4, 9, 16, 25, 36, 49, 64, 81 };

int one()     // bad function with side effect
{ // one
   global_i++;
   return 1;
} //end one

void print()
{ // print
   for( int i = 0; i < MAX; i++ ) {
       printf( "arr[%d] = %2d\n", i, arr[ i ] );
   } //end for
} //end print

int main()
{ // main
   arr[ one() + global_i ] = arr[ one() + global_i ];
   print();
   exit( 0 );
} //end main
```

Sample Side Effect, Output

```
arr[0] =  0
arr[1] =  1
arr[2] =  4
arr[3] =  9
arr[4] = 16
arr[5] = 25
arr[6] = 36
arr[7] = 64 ← element arr[ 7 ] overridden with arr[ 8 ]
arr[8] = 64
arr[9] = 81
```

Sample Side Effect, Output

In the following assignment, what is computed?

```
arr[ one() + global_i ]
   = arr[ one() + global_i ]
      = arr[ one() + global_i ];
```

Not necessarily visible from the source code!

Sample Side Effect, Output

```
arr[0] =  0
arr[1] =  1
arr[2] =  4
arr[3] =  9
arr[4] = 16
arr[5] = 25
arr[6] = 36
arr[7] = 81 ← element arr[ 7 ] overridden with arr[ 9 ]
arr[8] = 81 ← element arr[ 8 ] overridden with arr[ 9 ]
arr[9] = 81
```

Sample Side Effect

- Was the correct element overridden?
- Was the right element referenced?
- Which language rule should apply?
- Must left-hand side of assignment operator = be evaluated first? Or the right-hand side? And every part of it?
- Should the function `main()` be allowed to change `global_i` or any other global object?
- Did the programmer commit a SWE sin?

References

1. Side effect: http://en.wikipedia.org/wiki/Side_effect_(computer_science)
2. Functional language: http://en.wikipedia.org/wiki/List_of_programming_languages_by_category#Functional_languages
3. Turing Machine: http://plato.stanford.edu/entries/turing-machine/
4. Functional vs Imperative: http://msdn.microsoft.com/en-us/library/bb669144.aspx