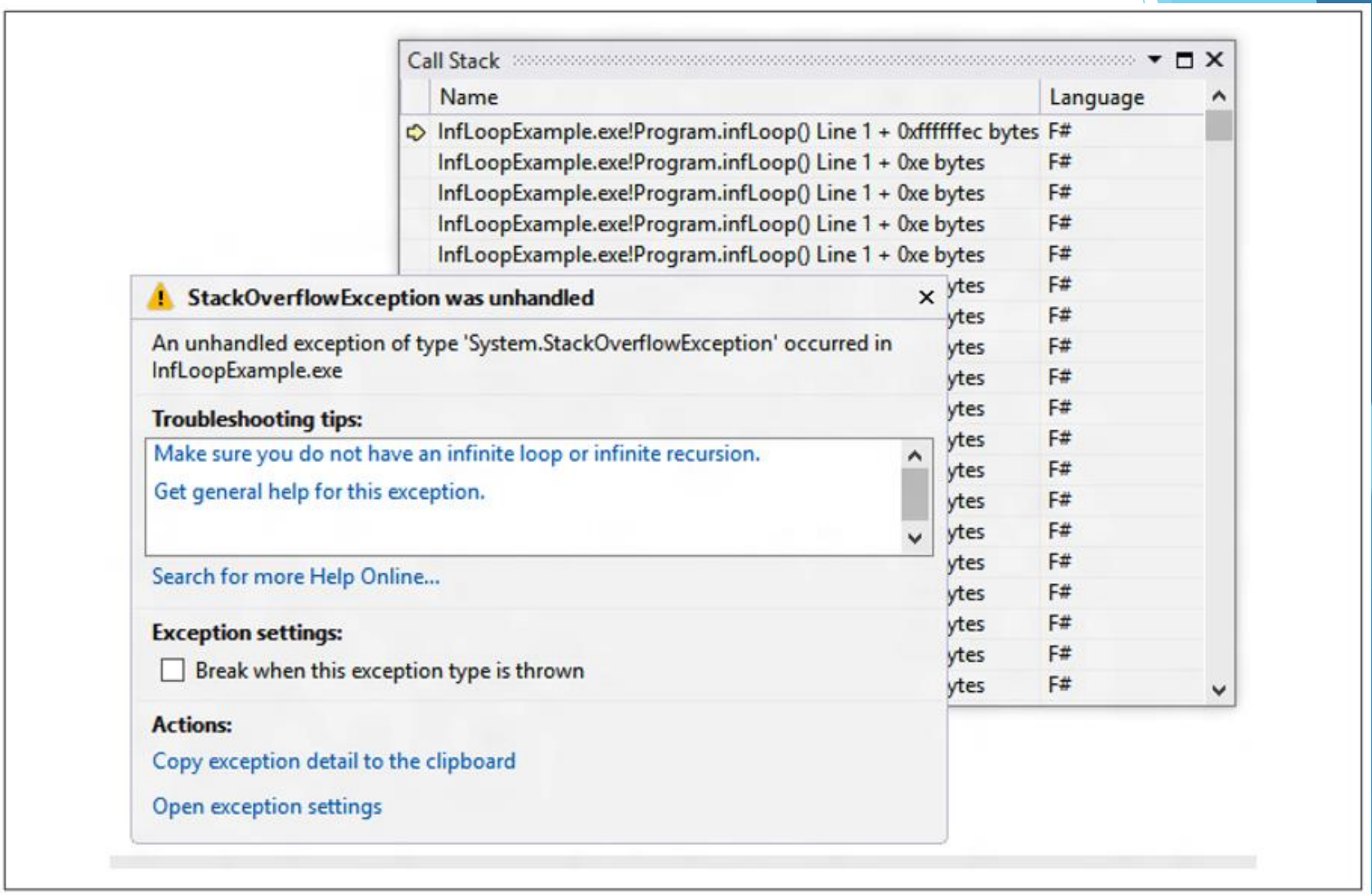
Programming Concepts and Languages

Spring 2024

Learning Objectives

- By the end of class today, you should be able to:
  - ✓ explain Tail Recursion
  - ✓ explain how to avoid stack overflows with tail recursion using accumulator and continuations
  - ✓ implement simple tail recursive F# programs
  - ✓ explain and implement simple F# programs using the following Recursive Data Types
    - ✓ Sequences
    - ✓ Sets
    - ✓ Maps
    - ✓ Arrays
- N/B Feedback to course project during exercises

Avoid Stack Overflow



Stack Frame

- For every function call, the runtime allocates a stack frame.
  - stored on a stack maintained by the system.
- A stack frame is removed when a call completes.
  - If a function calls another function, then a new frame is added on top of the stack.
- The size of the stack is limited, so too many nested function calls leave no space for another stack frame, and the next function can't be called.
- When this happens in .NET, a StackOverflowException is raised.

Stack Overflow

What can we do about it?

- The essential idea is that we only need to keep the stack frame because we need to do some work after the recursive call completes.

- Use **tail recursion**

```
let rec i_am_tail_recursive arg =
        (*check out of bound and fail with*)
        if(arg = 1000) then true
        else i_am_tail_recursive (arg + 1)
```

  - the last operation that ***i_am_tail_recursive*** function performs in the else branch is a recursive call.

- It doesn't need to do any processing with the result, it just returns it directly.
- This kind of recursive call is called tail recursion
- the result of the deepest level of recursion ***i_am_tail_recursive(1000)***, can be directly returned to the caller

Benefits of using Tail Recursion

- The function executes slightly faster, because fewer stack pushes and pops are required.
- The function can recurse indefinitely.
- No StackOverflowException is thrown.

- a function is considered tail recursive if and only if there is no work to be performed after a recursive call is executed

Tail-Recursive Patterns

Accumulator pattern

- add additional parameters
- pass an accumulator parameter to the recursive call so that the base case will return the final state of the accumulator.

Continuations – cont()

- rather than passing the current state of the accumulator "so far" as a parameter to the next function call, pass a function value representing the rest of the code to execute
- i.e., rather than storing "what's left" on the stack, you store it in a function.
- Continuations are function values that represent the rest of the code to execute when the current recursive call completes
- Conceptually, you are trading stack space (the recursive calls) with heap space (the function values).

Is this tail recursive?

```
let rec factorial x =
    if x <= 1
    then 1    // Base case
    else x * factorial (x - 1)
```