CSCI 136: Data Structures and Advanced Programming
Lecture 9
Recursion
Instructor: Dan Barowy
Williams

Topics

- Pre/post conditions
- Recursion

Your to-dos

1. Lab 3, due Tuesday 3/1 by 10pm ([random] partner lab!)
2. Read before Fri: Bailey, Ch 7.1–7.2.

Announcements

- Colloquium today: Senior thesis proposals #2, 2:35pm in Wege Auditorium with cookies.
- Survey: masking…
Pre/post conditions

Example

Example

What does this operation do? (i.e., what is our desired post-condition?)

Are you sure?

(code)
Example

These examples may seem contrived but trust me, they are not.

What should have been true about x?

1. x is an int
2. x < Integer.MAX_VALUE

Pre-condition

A pre-condition is a true/false statement (a “predicate”) that must always be true prior to a code segment (e.g., a function) being called. If a pre-condition is false, the result of executing the code is undefined.

Post-condition

A post-condition is a true/false statement (a “predicate”) that must always be true after a code segment (e.g., a function) is called assuming that the pre-condition was true.

Post-condition implications

If a pre-condition is false, there is no guarantee that the post-condition will be true.

Conversely, if a post-condition is false, then if the pre-condition is valid, the pre-condition must have been false.
Example, with conditions

```java
public static int addOne(int x) {
    Assert.pre(
        x < Integer.MAX_VALUE,
        "x must be an integer less than MAX_VALUE.");
    int z = x + 1;
    Assert.post(z > x, "z must be greater than x.");
    return z;
}
```

Pre/post in comments

- It's a good idea to put pre- and post-conditions in comments before your methods

```
/* @pre 0 ≤ index < length
 * @post returns char at position index */
```

```java
public char charAt(int index) { … }
```

Pre/post conditions

- Pre and post conditions form a contract.
- *Principle: post-condition is satisfied if pre-condition is satisfied.*
- Examples:
  - `s.charAt(s.length() - 1): index < length, so valid`
  - `s.charAt(s.length() + 1): index > length, so not valid`
- These conditions document requirements that user of method should satisfy.
- As comments, they are not enforced.

Assert class

- Pre- and post-condition comments are *useful to a human*, but it would be really helpful to know as soon as a pre-condition is violated (and return an error)
- The Assert class (in structure5 package) allows us to *programmatically check* for pre- and post-conditions
  
Remember: “Assume your code will fail.”
Assert class

The Assert class contains the static methods
public static void pre(boolean test, String message);
public static void post(boolean test, String message);
public static void condition(boolean test, String message);
public static void fail(String message);

If the boolean test is NOT satisfied, an exception is raised, the message is printed and the program halts.

General guidelines

1. State pre/post conditions in comments
2. Check conditions in code using Assert
3. Use Fail in unexpected cases (such as the default block of a switch statement)

• Any questions?
• You should use Assert in Lab 3

Recursion

General problem solving strategy:
• Split big problem into smaller sub-problems.
• Sub-problems may look a lot like original; are often smaller versions of same problem!
**Recursion**

Recursion is when a thing is defined in terms of itself. The most concrete application of recursion in computer science is when a function is called within its own definition.

```java
public static int fibonacci(int n){
    if (n == 0){
        return 0;
    }
    if (n == 1){
        return 1;
    }
    return fibonacci(n - 1) +
          fibonacci(n - 2);
}
```

**Recursion**

- Many algorithms are recursive
- Often easier to understand (and prove correctness/state efficiency of) than non-recursive versions!

**Recursion: formal structure**

- Recursion is a good solution when a problem fits a basic pattern:
- It has at least one “terminating” rule that does not use recursion, called the base case.
- It has at least one rule that does use recursion, called the recursive case. The recursive case should reduce the problem toward the base case.

We will talk about formal (i.e., “inductive”) proofs for recursion this week.

**Passing a note (recursively)**

What are our base/recursive cases?
Recursion

• \( n! = n \times (n-1) \times (n-2) \times \ldots \times 1 \)
• How can we implement this?
  • We could use a for loop...
    ```java
    int product = 1;
    for(int i = 1;i <= n; i++)
        product *= i;
    ```
  • But we could also write it recursively....

Graphically...

Activity: Factorial

• \( n! = n \times (n-1) \times (n-2) \times \ldots \times 1 \)
• Work with a partner and see if you can come up with a recursive solution.

Recap & Next Class

Today:

• Pre/post conditions
• Recursion

Next class:

• Mathematical induction