Department of Computer Science
Undergraduate Events

Events this week

Interview Skills Practice Session
Date: Mon., March 15
Time: 12 – 2 pm
Location: Rm 202, ICICS/CS

Transport Canada Info Session
Date: Tues., March 16
Time: 4 – 6 pm
Location: HENN 201

Financial Literacy 101
Date: Wed., March 17
Time: 12 – 1 pm
Location: Angus 426

CS Distinguished Lecture Series Featuring Jeff Hawkins
Date: Thurs., March 18
Time: 3:30 – 4:50 pm
Location: DMP 110

Townhall Meeting for CS Major/Honours Students
Date: Thurs., March 18
Time: 12:30 – 2 pm
Location: DMP 310
Lunch will be served!

Events next week

ICICS/KPMG Seminar: What Industry Wants
Date: Tues., Mar 23, 3:30 – 5 pm, Rm 2020, 2332 Main Mall
(Kaiser Bldg.)

UBC Science Co-op 30th Anniversary Celebration!
Date: Wed., Mar 24, 12 – 3 pm, Ladha Science Student Centre

Drop-In Resume and Cover Letter Editing
Date: Wed., Mar 24, 12 – 2 pm, Rm 255, ICICS/CS

CSSS 2009-2010 Year-End Boat Cruise
Date: Sat., Mar 27, 6:30 – 11 pm, Harbour Cruises Marina, 501 Denman St.

Administrivia

• Assignment #3 deadline has been extended (a bit)
  • Due Saturday March 20, 10:00pm
  • Lots of discussion of assignment 3 on Vista
  • A few updates posted to the home page
• In case you hadn’t noticed
  • Classes end Thursday April 15th
• Final exam
  • Friday Apr 23, 7:00pm
  • BCS section: DMP 101
Recursive Methods - Correctness

How can you check that a recursive method is correct?

check that the base case(s) is (are) correct
assuming that the recursive call(s) will return the right answer(s) for the smaller problem(s), show that the recursive step will return the right answer to the original problem
make sure that the terminating condition will eventually become true and the recursion will terminate – each recursive step should take you one step closer to reaching the base case

This is a form of mathematical induction (see CPSC 121)

Example: Fibonacci Numbers

The Fibonacci sequence is generated as follows:

- the first two numbers in the sequence are 1
- all other numbers are generated by adding the previous two numbers

1, 1, 2, 3, 5, 8, 13, 21, 34, …

The following web site contains some interesting facts about Fibonacci and his sequence of numbers:
http://plus.maths.org/issue3/fibonacci/
Example: Fibonacci Numbers

The description of the Fibonacci sequence on the previous page lends itself to the following recursive definition of the Nth Fibonacci number:

\[
Fib(N) = \begin{cases} 
1 & \text{if } n = 1 \text{ or } n = 2 \\
Fib(N-1) + Fib(N-2) & \text{if } n > 2 
\end{cases}
\]

Note that the recursive step involves two recursive calls to the method – we call this \textit{multi-branch recursion}.

Example: Fibonacci Numbers

The corresponding recursive method:

```java
public int fib( int n ) {
    if( n == 1 || n == 2 )
        return 1;
    else
        return fib( n - 1 ) + fib( n - 2 );
}
```
Example: Fibonacci Numbers

- Trace the following method call:
  ```java
  int fibNum = fib(4);
  ```

Corresponding stack trace

- Here's the stack trace corresponding to the call:
  ```java
  int fibNum = fib(4);
  ```
Example: Fibonacci Numbers

- Note that the recursive method wasn't difficult to write after we had figured out a recursive definition for the Nth Fibonacci number.
- Note that an iterative solution can be written that has O(N) time complexity, but it might take you a little longer to write…

Example: Binary Search

How do we find an element in a sorted array?

- Look at the middle element
- Then decide whether to search the first half or the second half.
- Return the position of the element in the array.

```java
public <E extends Comparable>
int bSearch(E obj, E[] array, int first, int last) {
```
Example: TreeMap

Example: Password Breaker
Recursion vs Iteration
Many recursive functions can be easily defined iteratively without using a stack:

```c
int fact(int n) {
    if (n == 0) {
        return 1;
    }
    return n * fact(n - 1);
}
```

```c
int fact(int n) {
    int result = 1;
    while (n > 0) {
        result *= n;
        n = n - 1;
    }
    return result;
}
```

Recursion vs Iteration (cont’d)
Recursion usually requires more memory than iteration
  each method call creates a new stack frame in which its parameters and local variables are stored
Most compilers can remove unnecessary recursive calls “tail recursion elimination”
Sometimes recursion is more natural so it may take more time to develop an iterative solution.
Rule of thumb:
  use iteration when it is easy and natural to do so.
  use recursion when it is easy and natural to do so.