**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
Example

Suppose we want to write a method to draw the following ramp of size 3 on the screen:

```plaintext
***
** *
***
```

We can define this problem in terms of smaller ones as follows:

```plaintext
***    **    *
**     **    *
***    **    *
```

Example cont'd

The following method uses the recursive definition on the previous slide to draw a triangle of a certain size:

```java
public void drawRamp( int size ) {
    if( size == 1 )
        drawRow( size );
    else {
        drawRow( size );
        drawRamp( size - 1 );
        drawRow( size );
    }
}
```
Recursive Method Calls – General Form

- Our `drawRamp` method illustrates the general form of a recursive method call:

```c
type recursiveMethod( type param1, type param2,... )
{
    if( base case )
        // handle base case (code omitted)
    else
    {
        // operations before recursive call
        // (code omitted)
        recursiveMethod( ... ); // recursive call
        // operations after recursive call
        // (code omitted)
    }
}
```

Recursion and Stacks

- There is a strong connection between recursion and stacks.
- Recall that the compiler uses a stack, called the run-time stack, in which to store data that is local to each method that is called.
- When a method is called, a new stack frame is generated and pushed onto the run-time stack. This frame stores, among other things, parameters and local variables for the method.
- When the method ends, its stack frame is popped off the run-time stack.
Recursion and Stacks cont'd

- The following is a trace of the run-time stack corresponding to the method call `drawRamp(3)`. Note that we show only the parameter `size` in each stack frame.

Recursion and Stacks cont'd

- The trace of the run-time stack gives us an insight into the space complexity of the method call. We observe that the method call `drawRamp(N)` will result in the run-time stack holding, at any given point, a maximum of `N` stack frames corresponding to the method `drawRamp`. Hence the `drawRamp` method has O(N) space complexity.

- Compare this to an iterative solution where we use a loop to draw the ramp in a single call to the method `drawRamp`. This implementation has O(1) space complexity.
Recursion and Stacks cont'd

- Any recursive method can be converted to an iterative method if we make a stack available. (Note that the use of a stack isn't always necessary.)

- We will now rewrite our `drawRamp` method as an iterative method that uses a stack to mimic the run-time stack maintained by the compiler.

```java
public void drawRamp( int size ) {
    Stack<Integer> sizeStack = new Stack<Integer>();
    // head towards the base case
    while( size > 1 ) {
        sizeStack.push( size );
        drawRow( size );
        size--;
    }
    drawRow( 1 ); // corresponds to base case
    // unwind the stack
    while( !sizeStack.isEmpty() ) {
        size = sizeStack.peek();
        drawRow( size );
        sizeStack.pop();
    }
}
```
Computing a factorial

The factorial of a non-negative integer is defined as follows:

\[ n! = 1 \quad \text{if } n=0 \]
\[ n! = n \times (n-1)! \quad \text{if } n>0 \]

Examples:

\[ 5! = 5 \times 4 \times 3 \times 2 \times 1 = 120 \]

Can we replace the definition above with a recursive definition?

Example: computing a factorial

- The factorial of a non-negative integer can be recursively defined as follows:

\[ n! = 1 \quad \text{if } n=0 \]
\[ n! = n \times (n-1)! \quad \text{if } n>0 \]

- Corresponding recursive method:

```java
public int factorial(int n) {
    return n == 0 ? 1 : n * factorial(n-1);
}
```
Palindromes

- A palindrome is a string that reads the same backwards as forwards
  - Abba
  - Madam, I’m Adam
  - A man, a plan, a canal, Panama!

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)