Programming Language Concepts
Control Flow - Loops

Janyl Jumadinova

20 October, 2020
Loops

while: while (condition) {
    loop body
}

The loop body is executed zero or more times (the condition might be false from the very beginning).
Loops

do: do {
    loop body
} while (condition);
Loops

do: do {
    loop body
} while (condition);

The loop body is executed one or more times (the condition is not tested until after the loop body has been executed at least once).
do...while Is Syntactic Sugar

We can achieve the same effect as a “do while” using a plain while loop, for instance:

```java
while (true) {
    loop body
    if (! condition) break;
}
```
Iterators

In Java we can do things like this:

```java
String[] words = {"cat", "dog", "bird", ...};
...
for (String s : words) {
}
```

Most compound data types in Java include an iterator feature.
Recursion represents a certain special kind of “control flow”.

**Problem:** compute the values in the Pascal’s triangle

Each internal element is the sum of the two above it.

Values are also called “binomial coefficients”
Recursion

Each internal element is the sum of the two above it.
Values are also called “binomial coefficients”
Reorganization and number rows and columns:

0: 1
1: 1 1
2: 1 2 1
3: 1 3 3 1
4: 1 4 6 4 1
5: 1 5 10 10 5 1
...

0 1 2 3 4 5 ...

Rows: n
Columns: k

\text{binom}(4, 2) = 6
Recursion

\[
\begin{array}{ccccccc}
0: & 1 \\
1: & 1 & 1 \\
2: & 1 & 2 & 1 \\
3: & 1 & 3 & 3 & 1 \\
4: & 1 & 4 & 6 & 4 & 1 \\
5: & 1 & 5 & 10 & 10 & 5 & 1 \\
\vdots \\
0 & 1 & 2 & 3 & 4 & 5 & \ldots \\
\end{array}
\]

\[
\text{binom}(n,k) = 1 \text{ if } k = 0 \text{ or } k = n
\]
Recursion

\[
\text{binom}(n,k) = \text{binom}(n-1,k-1) + \text{binom}(n-1,k)
\]

0: 1
1: 1 1
2: 1 2 1
3: 1 3 3 1
4: 1 4 6 4 1
5: 1 5 10 10 5 1
...

0 1 2 3 4 5 ...
public static int binom(int n, int k) {
  if (k == 0 || k == n) {
    return 1;
  }
  else {
    return binom(n-1, k-1) + binom(n-1, k);
  }
}
Recursion

```java
public static int binom(int n, int k) {
    int retval, temp1, temp2;
    if (k == 0 || k == n) {
        retval = 1;
    }
    else { // recursive case:
        temp1 = binom(n-1, k-1);
        temp2 = binom(n-1, k);
        retval = temp1 + temp2;
    }
    return retval;
}
```

Recursive calls always take us back to the beginning of the function.

“Returns” could take us back to a location in the function or to some external location.
Let’s eliminate explicit recursion and instead simulate the behavior of the frame stack.

We will need a “frame” to hold values of local variables $n$, $k$, temp1, temp2 (and retval, but in this example we don’t need it so we’ll skip it).

The frame must also hold a “return address”, which we will simulate with an integer value.
Recursion

private int n, k, t1, t2; // parameters and local variables
private int ra; // return address

// Constructor
public Frame (int n, int k, int ra, int t1, int t2) {
    this.n = n; this.k = k; this.ra = ra;
    this.t1 = t1; this.t2 = t2;
}

public int n() {return n;}
public int k() {return k;}
public int ra() {return ra;}
public int t1() {return t1;}
public int t2() {return t2;}

And we’ll need a stack:

```java
import java.util.Stack;
...
Stack<Frame> stack = new Stack<Frame>();
```
Recursion

And we’ll need a stack:

```java
import java.util.Stack;
...
Stack<Frame> stack = new Stack<Frame>();
```

– Each recursive call is replaced with a “push” to the stack; execution then goes back to the top of the function.
– Each “return” is replaced by a “pop” and a return to the location in the (popped) return address.
The heart of the “binom” function is an infinite loop that uses the return address variable ra to “goto” the correct section of code to simulate a return from a recursive call.

```c
while (true) {
    switch(ra) {
        case 0: // base case test: go here when first entering the function
            ...
        case 1: // First recursive call to binom.
            ...
        case 2: // Second recursive call to binom.
            ...
        case 3: // We just returned from the second recursive call.
            ...
        case 4:...
    }
}
```
To prepare to simulate a recursive call, we save values onto the stack, update to new values, and return to the top of the loop by setting ra to 0. E.g., here’s the first recursive call to binom(n−1,k−1):

```java
stack.push(new Frame(n,k,2,temp1,temp2));
n=n-1; k=k-1;ra=0;
    continue;
```
Recursion

To simulate a “return”, we see if there is anything in the stack (if not, then binom was called from an external function). Pop the stack, restore old variable values, and go to the popped return address:

```java
// Is this a top-level call? Then return:
if (stack.empty())
    return retvalue;
else {
    Frame s = stack.pop();
    n = s.n(); k = s.k(); ra = s.ra(); // go here next
    temp1 = s.t1(); temp2 = s.t2();
}
```