Index of /~cs61b/lectures/07

Name	Last modified	Size	Description
Parent Directory	24-Sep-98 21:22	-	
BinarySearch.java	09-Sep-98 22:16	2k	

```
                     CS61b: Lecture 7
                    Wednesday, September 9
ANNOUNCEMENTS:
-------------
There are not enough students to justify the "overflow" lab from
  1-3pm on Wednesdays.  If you were planning to go, please choose another
  lab and have this week's lab checked off at the beginning of next week's.
The last day to switch sections through Telebears is Friday.  After that,
  you will be admitted into a lab only if some other students do not show up,
  i.e., there are free seats available.  Your labs will not be graded if you
  cannot attend an available lab.
Everyone on the waiting list will be assigned to a section automatically,
   probably 8-10am/pm or perhaps 11-1 on Wednesday.
My office hours: 11-12 on Monday, 11-12 on Thursday; 777 Soda.

WRITING LOOPS
-------------
Most programmers write loops in an ad hoc way.  This is a big source of bugs.
 *  Forget initialization.
 *  Forget increment (or other operations) to guarantee termination.
 *  Fence-post errors (i.e., off-by-one errors).
 *  Hard to read: e.g., wrong variables, mixture of different parts of code

STEPS FOR WRITING LOOPS:
1) Choosing the variables and their meaning
2) Sketch the loop
3) Setting the loop condition
4) Initialization
5) Guaranteeing termination
6) The loop body

EXAMPLE: Exponentiation
-----------------------
Given x >= 0, y > 0, write a loop to compute x**y (i.e., x^y, x to
  the power y) using only multiplication:  x**y = 1*x*x*x...*x, y times

A good way to write loops is often backwards from the end.  Several "tricks."
Loop trick #1: make up a variable (count) to replace a constant (y).

1) Variables: Need to keep track of number of multiplies so far (count) and
   result of more recent multiplication (result)
2) Sketch
             int count;  // number of multiplications
             int result; // result = 1*x*...*x count times
             ...init...
             while (...cond...) {
```

```
             ...body...
            }
            ...finish...
3) Condition: what do we want to be true after the loop finishes? result = x**y
   This will be true when count == y, so make the negation a loop condition.
     cond becomes:      while (count != y)
4) Initialization: count (number of multiplies) should probably be 0.
     init becomes:     count = 0;  result = 1;
5) Termination: eventually, count should be equal to y (assumed non-negative)
     body includes:    count = count + 1;
6) Body:
     body becomes:     result = result * x;
                       count = count + 1;

EXAMPLE: Integer square root
----------------------------
Given int n >= 0, write a loop to compute int result, such that
      (result**2 <= n) & (n < (result+1)**2)

Loop trick #2: Given desired A & B at the end, use negation of A (or B) as the
  condition; and B (or A) as an "invariant" (always true after each iteration)

1) Variables: int result; // result**2 <= n
3) Condition: (n >= (result+1)**2)
4) Initialization: result = 0
5) Termination: result = result + 1;
6) Body: nothing else

EXAMPLE: Binary search
----------------------
Given a sorted non-empty Vector, v, search for an element e.

Loop trick #3: divide-and-conquer

1) Variables: int left, right; // searching [left...right)
              int mid;         // between
3) Condition: while (left != right-1)
4) Initialization: left = 0; right = v.size();
5) Termination: either left increases or right decreases;
6) Body: mid = (left+right)/2;
         if (v.elementAt(mid) < e) { left = mid; }
         else if (v.elementAt(mid) > e) { right = mid; }
         else { left = mid; right = mid+1; }

EXAMPLE: User input (read-eval loop)
------------------------------------
1) Variables: String command;
3) Condition: (!command.equals("quit"))   // for example
4) Initialization: prompt user, then set command = ...readLine()
5) Termination: prompt user, then set command = ...readLine()
6) Body: process the command

FOR LOOPS
---------
The following common while loop pattern can also be written using a "for loop"

     init_expr                    for (init_expr; cond_expr; inc_expr) {
     while (cond_expr) {            loop_body
       loop_body                  }
       inc_expr
     }
```

UNUSUAL FLOW OF CONTROL

To go to the next iteration, without executing the rest of the body use "continue"

```
      while (something) {
        execute-this-code;
        if (something) continue;
        avoid-this-code-if-something-is-true;
      }
```

To get out of such a loop, you may use a "break" or "return":

```
      while (true) {
         ...
        if (something) break;   // or return
      }
```

Because of break and return in loops, it is sometimes useful to write loops that, without the break or return, would be "infinite":

```
    while (true) {                for (;;) {
         ... }                      ... }
```