CSC 180 lab 10: Recursion

Thurs Nov 15 or Mon Nov 19, 2001

Recursion is tricky. For some reason, it is not a good match for how people think. However, it is a very good way to solve certain kinds of problems, so it is a technique all computer programmers need to know about.

1. Write the recursive factorial function (without consulting your notes in which you already have it written out): The result of your factorial(x) for x==1 is 1, and for x>1 it is x*f(x−1).

Write a short test main() function which does a scanf of an integer and then printfs the value of factorial(x), so that you can verify that your factorial function is working.

2. Make your factorial function work for zero, i.e. when called as factorial(0). (Zero factorial is 1.)

3. In general, a loop can be turned into recursion. Write a recursive function which contains no loop instructions and prints ‘‘Hello’’ the same number of times as the following function:

```
void hello(int n)
{
    int i;
    for (i = 0; i < n; i++)
       printf("Hello\n");
}
```

4. Turning recursion into a loop is harder, in general, but it is usually possible to produce a straightforward non-recursive version of most simple recursive functions.

Write a non-recursive function (using loops, but not calling itself) which prints ‘‘Hello’’ the same number of times as the following function:

```
void hello2(int n)
{
    int i;
    for (i = 0; i < n; i++)
       printf("Hello\n");
    if (n > 0)
       hello2(n-1);
}
```

5. Write a recursive function which adds the elements of an array (without using ‘for’ or ‘while’ or any loop constructs other than recursion). It takes two parameters, a pointer-to-int which is a pointer to the beginning of the array, and an int which is the size of the array.

The sum of a zero-sized array is zero (this is the ‘‘base case’’). The sum of a larger array is a[0] plus a recursive call which increments the pointer by one (thus skipping a[0]) and decrements the size by one (so that we don’t exceed the bounds of the array).

6. If you have time after the above, you might want to spend the rest of the lab time looking at the sorting and searching algorithms. Copy the programs from the example directory (http://www.dgp.toronto.edu/~ajr/180/example/7/), put printf statements in them to figure out what’s going on, and try them with different values.

7. (harder) Write a recursive version of the binary search, which does not assign to ‘low’ and ‘high’ but takes them as parameters, and does not contain any loop constructs except for recursion.