Turning a Problem into a Program

Let's elaborate a little on the mechanics of turning a problem into a program, using the first inclass exercise as an example. The mathematical statement of the problem is to compute the sum

$$A_n = \sum_{j=1}^{n} j$$

for some specified value of n. Let's rewrite this as a recurrence relation that tells us how to compute the sum A_j from A_{j-1}:

$$A_j = A_{j-1} + j\,.$$

Now the algorithmic procedure is simple to describe. We maintain a running sum, S say, initialized to zero, and successively add to it 1, 2, 3, etc., up to n. The quantity S is sometimes called an *accumulator*, and the iterative procedure of successively increasing it by the next term is normally implemented as a *loop* in a program.

Now let's write the problem in a more schematic way, as a *flow chart.* Such diagrams are a bit outdated, but still represent a helpful way of organizing a calculation or a program. In this case the chart is very simple, but in more complex projects it can be a very important tool, as it allows the programmer to see the overall structure of the calculation, and how the various pieces relate to one another. It also translates more or less directly into a program that actually does the calculation:

The chart starts by *initializing* (colored blue) the quantities used in the calculation—in this case S and j. Then we embark on a loop (in red) consisting of a *loop test*, which controls whether or not we will perform one more iteration, and a *loop body*, which is the actual iterative calculation to be performed. Finally, once the loop is done, we have a *termination* stage (shown here in purple), in this case just printing out the results. Follow the logic for (say) $N = 10$ and verify that the result is $\frac{1}{2}N(N+1) = 55$, as expected.

Finally, we can take the flow chart and convert it into our chosen programming language, in this case C++, almost element by element.

```
        #include <iostream>                 // necessary
        using namespace std;

        const int n = 10;                   // define n

        main()                              // program starting point
        {
            int S = 0;                      // declare and initialize the
            int j = 1;                      // accumulator and loop counter

            while (j <= n) {                // loop test

                S = S + j;                  // loop body: do if true
                j = j + 1;

            }
                                            // termination:       print and end

            cout << "n = " << n << ", sum = " << S << endl;

        }
```

Note the use of comments (//) to identify and explain the various steps.

The corresponding code in python is very similar:

```
n = 10 # Define N

S = 0 # initialize the accumulator
j = 1 # and loop counter

while j <= n: # loop test
    S = S + j # loop body: do if true
    j = j + 1

        print "n =", n, " sum =", S # termination: print and end
```

This is a very simple example, but the basic steps and structural components of this program will recur throughout this course.

For example, the following flowchart outlines the logic of most of the calculations we will encounter during the next few weeks: