EECS 211 – Top-down Development

An important method for algorithm and program development is the so-called top-down approach. In this approach the programmer considers first the outermost, or top, level of the problem. Is the outermost aspect of the problem a sequence of calculations, a set of cases, or a repetition? In the latter case, is it a test-first, test-last or counting repetition? The details of this level are settled first without consideration of what the actual processes at lower levels are. For example, if the outer level is a repetition, how will loop initialization and termination be handled? This can usually be answered without considering what the body of the loop actually does. Similarly, if the outer level were, say, a two-way branch, we would determine the exact form of the test without considering the specific computations in each one of the branches.

One effective way to plan out the solution of a programming problem is through the use of flow charts to represent the control structures. More experienced programmers often write C/C++ code directly because there is a one-to-one relation between these flow charts and C/C++ if statements and iteration statements. Less experienced programmers can benefit from the visual impact of a picture without worrying about C/C++ forms or syntax.

We illustrate the concepts and methods by considering program 1 assignment. In this assignment you are to write a program that computes averages of sets of scores. After each score is an indicator values that tells whether or not there are most sets. See the assignment for all the details.

At the outermost level, this is a repetition. Your program is supposed to do something (calculate an average, but for now that doesn't matter) to one or more sets of data. Moreover, the indicator about when we are at the end of the data is at the end of each data set. Thus, the requirements are that the program perform at least one average. Therefore, not only is this a repetition, it is a test-last repetition. In C/C++ terms, this is a do-while loop. This kind of repetition is represented by the following flow chart:

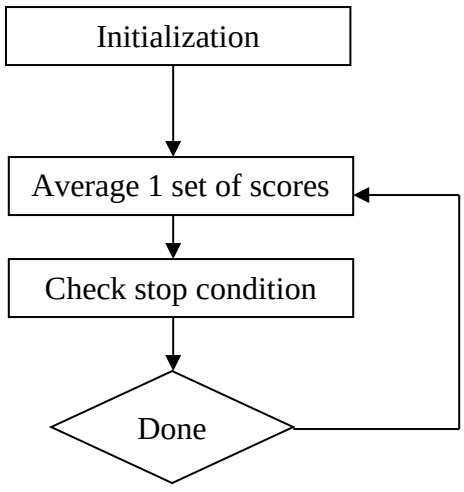

Figure 1: A test-last loop.

One interesting and important aspect of top-down design is that the programmer can also implement the partial solutions developed along the way. For example, the C/C++ statement type that implements the test-last loop is the do-while statement. In our assignment, there is no initialization step necessary for processing sets of scores: we don't have to do any preparation, just start processing the individual sets. To check for the stop condition for Lab 1, we read in an indicator value from cin and compare this with -2 (the sentinel value). So, C/C++ code for our program could look like this.

```
// No process initialization need for processing different sets of data.

do
{
    cout << "Processing one set would go here.\n\n";
    cin >> more;
} while (more != -2);
```

We can actually compile and run this code and ensure that the loop continues until -2 is entered for the termination test. Of course, there is no actual computation of averages yet, but we could be sure the outer-level mechanism worked the way we wanted it to.

Once we were happy with this level, we could consider the process of calculating average for one set of scores, an easier problem than we started out with. This involves two steps to be performed in sequence – add up the scores and compute and display the result. In flow chart terms a sequence of processes is just two rectangles in order.

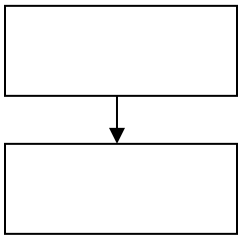

Figure 2: A generic sequence flow chart

One very useful feature of the particular flow charts corresponding to C/C++ control statements is that they are structured. Each diagram has one arrow in and one arrow out. This allows the programmer to simply paste the flow chart (or C/C++ code) inside a box in another flow chart (or inside the brackets in another C/C++ program). So, for example, we could past Figure 2 inside the "One inner process" box of Figure 1 and thereby get a slightly more detailed flow chart, just a bit closer to the ultimate solution.

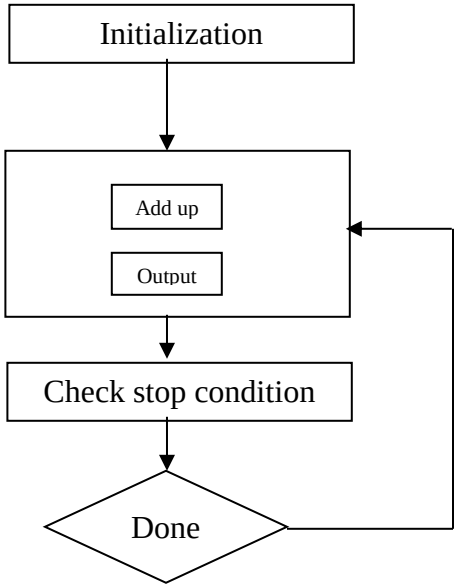

Figure 3: Flow chart showing top two levels.

Continuing in this way we consider what is involved to compute the sum and what is involved to compute and display the result. Computing the sum is a repetition – for as many scores as there are, read one score and add. Also, for this repetition there is both process and loop initialization. For the process we know we have to initialize both the sum and the number of bad scores to 0. For the loop, according to the assignment the data file contains an integer that tells how many scores are in the set. Therefore, setting up the loop means getting that number. Because we know how many scores there will be, the loop can be a counting loop. The for statement in C/C++ contains all the rest of the loop mechanism except, of course, the body part. Showing the result involves two cases – there were valid scores or not. Therefore, that step is a two-way branch.

In the top-down process, once the requirements for a level are simple enough (which depends on the skill and experience of the programmer) the remaining code is just entered and the final system is tested.

The reader is encouraged to use the top-down approach for all the assignments.