John Pennig

Homework 10

1. Chapter 8 Review Questions:

   1. What is the definition of *control structure?* A control structure is a control statement and the collection of statements whose execution it controls.

   2. What did Böhm and Jocopini prove about flowcharts? All algorithms that can be represented by flowcharts can be coded in a programming language with only two control statements. One for choosing between two control flow paths, and one for logical iterations.

   3. What is the definition of *block?* A block is group of one or more statements enclosed by that programming languages specific syntax. Typically brackets { }.

   8. What are the common solutions to the nesting problem for two-way selectors? The common solutions to the nesting problem are indentation to signify which if statement goes with which else clause. Another solution is placing the if statement in a compound statement to differentiate. Also, sometimes there is an end special word to signify the end of the current selection statement.

   9. What are the design issues for multiple-selection statements? A design issue is the question of the type of expression on which the selector is based. Another design issue for multiple-selection statements is whether single statements, compound statements, or statement sequences may be selected. Other design issues include the form of the case value specifications, and what should result from the selector expression evaluating to a value that does not select one of the segments.

2. What are the design issues for iterative statements. How do they relate to the different kinds of iterative statements provided in programming languages? The design issues of counter-controlled loops are what is the type and scope of the loop variable, should it be legal for the loop variable or loop parameters to be changed in the loop, and how does the change affect loop control, should the loop parameters be evaluated only once, or once per iteration, and what is the value of the loop variable after loop termination.

The design issues of logically controlled loops are should the control be pretest or posttest and should the logically controlled loop be a special form of a counting loop or a separate statement.

3. What are the design issues for user-located loop control mechanisms? Using the criteria from Chapter 1, how do such mechanisms measure up against the standard pretest or posttest loop structures? The design issues for user-located loop control mechanisms are should the conditional mechanism be the integral part of the exit, and should only one loop body be exited or can enclosing loops also be exited. Readability can be heavily affected when you go against the standard pretest or posttest loop structures. It can be harder to find out where a program may exit a loop at if you check the condition in the middle. The writability of a function can be improved by allowed more flexibility to the programmer, allowing the programmer to exit the loop at any step they want. The reliability can be affected by causing unwanted exits if the condition is constantly checked during the loop structure where you may want it to finish.

4. Who introduced Guarded Commands and why? How do the guarded conditional and the guarded loop structure compare, respectively, with the multi-way selection and the pretest loop? Guarded Commands were introduced by Dijkstra, and he introduced them to provide control statements that would support a program design methodology that ensured correctness during development rather than when verifying or testing completed programs. Guarded Commands semantics are different from multi-way selection and the pretest loop. In guarded conditional and loops, all the Boolean expressions are evaluated every time, and the program can technically choose any of the true Boolean expressions. Guarded Commands must also have at least one true Boolean or else it throws an error so the programmer must take into account all possibilities.

5. Short circuit evaluation. Consider the following problem. We have to read a sequence of int values from a file into an array in C++, while simultaneously verifying that the numbers are in non-decreasing order. Let infile denote the file stream, size denote the size of the array A[0…(size-1)]. There are three possible ways the loop can exit: end of file, item out of order, and end of array. Here is the incomplete “pseudocode”: index = 0; // assume array size is >= 1 if (infile >> previous_num) // read an item from file + verify success of read { A[index] = previous_num; index++}//store in array and increase index

While (______________ && ______________ && _________________) { A[index] = next_num; previous_num = next_num; index++;} The if statement reads the first item from the file to prime the loop. The loop runs as long as none of the exit conditions occur.
(a) What are the Boolean expressions for each of the three end conditions?

1. End of File: !(infile >> nextnum)

2. Item out of Order: nextnum < previous_num

3. End of Array: index >= array_size

(b) List them in order of "priority" for checking? Justify your order.

End of array check is first to ensure that there is enough room in the array to continue reading int values from the file. Next up is the end of file check to see if more values need to be read into the array or not. Finally, check if the item is out of order only after you ensure there is space in the array, and that there are more values in the file to read.

(c) How would this code be written if C++ did not support short-circuit evaluation?

We would have to have independent tests that check for all three end conditions that would break out of the loop if they equated to false. Such as if (index >= array_size): break;

(d) From the point of view of the basic evaluation criteria for programming languages in Chapter 1 of Sebesta's text, how do these two options compare? How does this comparison relate to issues discussed in Sections 7.6 (short circuit evaluation) and 8.3.3 (User-located loop control) (~400 words)

The readability of short-circuit evaluation is very good because all exit conditions are in one place. You can immediately see where and why a loop stops. However, sometimes these short-circuit evaluation statements can have a lot of moving parts that may make it hard to read. Without short-circuit evaluation, all Boolean expressions are evaluated in a statement. This can cause unwanted lack of readability especially if the statement increments a value during a Boolean expression check such as i++ < range. Also, without short-circuit evaluation, you may need separate if statements with breaks afterwards to properly implement your code. This can cause clutter in your code making it harder to read, but it may help with understanding where exactly the loop may exit, and why.

The writability of short-circuit evaluation is also very good because you can chain the Boolean expressions together into one, and it typically requires less code and clutter than user-located loop control. It is also a simple concept of combining the Boolean expressions together using “ands”. The writability of user-located loop controls can be easily implemented in most languages using if statements with breaks. It may however take more writing since you need an if statement and break.

The reliability of short-circuit evaluation is also good because most languages ensure that the short-circuit statement is evaluated left-to-right and that it stops at the first false. This is very safe for the execution of the program to ensure that you do not check any Boolean expression that you do not have too, that could cause errors to be thrown. However, you must remember what order to perform the tests in to ensure the proper checks are being done before others to not cause unintended side effects. The reliability of user-located loop control is very good as well because you can perform each check exactly where you want to in the loop without side-effects. You can see exactly when a statement is called and manipulate it easily.

The readability, writability, and reliability of programming functions are extremely important to take into account. When it comes to short-circuit evaluation and user-located loop control, these are the things that define what to use and where. There are advantages and disadvantages of each and you must understand them to fully understand when to use them to ensure readability, writability, and reliability of your program.