**Chapter 8 Homework Solutions**

MTH 135

*Review Questions:*

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

3. A block is a sequence of code, delimited by either braces or reserved words.

4. The design issue for selection and iterative control statements is “should the control structure have multiple entries?”

5. The design issues for selection structures are:

* What is the form and type of the expression that controls the selection?
* How are the then and else clauses specified?
* How should the meaning of nested selectors be specified?

6. The unusual thing about Python’s design of compound statements is that it uses indentation to specify them.

8. The common solutions to the nesting problems of two-way selectors are (1) specify a rule that says that the else clause is always paired with the nearest unpaired then clause (above the else clause) and (2) use special closing words to indicate when an if construct is complete.

9. The design issues for selection structures (multiple-way) are:

* What is the form and type of the expression that controls the selection?
* How are the selectable statements specified?
* Is execution flow through the structure restricted to include just a single selectable statement?
* How should unrepresented selector expression values be handled, if at all?

14. The design issue for iterative control statements are (1) How is the iteration controlled? and (2) Where should the control mechanism appear in the loop statement?

15. The design issues for counter-controlled loops are:

* What are the type and scope of the loop variable?
* What value does the loop variable have a loop termination?
* Should it be legal for the loop variable of loop parameters to be changed in the loop, and if so, does the change affect loop control?
* Should the loop parameters be evaluated only once, or once for every iteration?

22. The main reason user-located loop control statements were invented because they fulfill a common need for goto statements through a highly restricted branch statement.

*Problem Set:*

1. Three situations in which a combined counting and logical control loops are:

a. A list of values is to be added to a SUM, but the loop is to be exited if SUM exceeds some prescribed value.

b. A list of values is to be read into an array, where the reading is to terminate when either a prescribed number of values have been read or some special value is found in the list.

c. The values stored in a linked list are to be moved to an array, where values are to be moved until the end of the linked list is found or the array is filled, whichever comes first.

4. Unique closing keywords on compound statements have the advantage of readability and the disadvantage of complicating the language by increasing the number of keywords.

5. The arguments (pro and con) for Python’s use of indentation to specify compound statements in control statements are (pro) it is easy to see when nesting is happening and which else clause goes with which then clause. The cons are that it is very easy to make mistakes with the indentation and thus with the meaning of the structures.

6. Using closure reserved words for control statements that are the reverse of the corresponding initial reserved words can lead to readability problems. For one, the reverse of the words may or may not be true English words, so readers will either not understand the word or think it has its true meaning. Unless the reader knows that the reverse word is being used as a closing word, this could lead to problems. Also, if the writer accidentally types the reverse of the word wrong (e.g. reverses two adjacent characters), the compiler will not recognize it and will produce an error, while a reader/human will most likely not notice the reversal because our brains tend to know what we mean.

9. The arguments for and against the exclusive use of Boolean expressions in control statements in Java (as opposed to also allowing arithmetic expressions as in C++) are that it can aide readability since there will not be confusion generated by wondering how an arithmetic expression can be true or false and it could also hinder writability because it would not be as flexible for the programmers.

*Programming Exercises*:

1c. **for** (k = (j + 13) / 27; k <= 10; i = 3 \* (++k) - 1)

**Hard to read, but easy to write.**

3c. **switch** (k)

{

**case** 1: **case** 2:

j = 2 \* k - 1;

**break**;

**case** 3: **case** 5:

j = 3 \* k + 1;

**break**;

**case** 4:

j = 4 \* k - 1;

**break**;

**case** 6: **case** 7: **case** 8:

j = k - 2;

**break**;

**default**:

printf("Error in switch, k =%d\n", k);

}

Having to put the breaks in each one is not good for writability; also not as clear that we are looking for what k is equal to in each of the cases.