Jason Egbert
CS354-002
Fall 2018
12/03/18

Textbook Assignment 3: Control Flow

Question 6.1

Whether or not these statements are contradictory depends on what is allowed within the language. If one element of the arithmetic operands is a function call, which can affect the value of another operand, then the two statements contradict each other because two compilers written to evaluate in different orders may compute differing results. However, in a language which outlaws side-effects and prevents changing of a value once it has been set will suffer no problems with this regard. Likewise, if parentheses are added to force evaluations to occur in a specific order, any properly written compiler should comply and produce the same output.

Question 6.2

No, it should not result in non-intuitive evaluations, because if there is ever a question of which operator is more important, left to right evaluation occurs. i.e., the operations should always be performed in an order that will yield the mathematically correct results.

Question 6.12

If a programmer has established has set up a loop to evaluate two expressions whose values are relevant to a table of data, and the conditions for exiting the loop are when the expressions evaluate to specific values, and we want to fill all of the table, we want the side effect of filling the table to occur at every step, so we would want a long-circuit evaluation to ensure that all of the entries of the table are filled, and that we don't stop evaluating as soon as one of the expressions reaches its maximal value.

Question 6.25

```
    boolean isBlank = false;
    while(isBlank) {
        line = read_line();
        if(all_blanks(line))
            isBlank = true;
        consume_line(line);
    }
```

or

```
    do {
        if(lines != null)
            consume_line(line);
        line = read_line();

    } while (all_blanks(line));
```

The While loop alternative requires establishing extra variables, and will evaluate every statement, making it theoretically less efficient than the mid-test loop, but it accomplishes the same task. The do loop requires one extra line of logic to ensure that we're not trying to consume null (unless the function conume_line auto-checks for null, then that line is unnecessary), and is effectively no more inefficient, just arranged differently than the mid-test loop

Jason Egbert
CS354-002
Fall 2018
12/03/18

Question 6.26

The goto function in C is in some situations quite useful. However, it makes the code very difficult to follow, no matter how efficient it may run. In theory, one could simply lock the first_zero_row = i and break inside an if statement, but this would require a boolean statement inside the second for-loop, and totally eliminate the need for a goto. The restructured code would look similar to this:

```
int isZero;
int first_zero_row = -1;
int i, j;
for (i = 0; i < n; i++) {
        isZero = 1;
        for (j = 0; j < n; j++) {
                if(A[i][j]) {
                        isZero = 0;
                        break;
                }
        }

        if(isZero) {
                first_zero_row = i;
                break;
        }
}
```