Ahram Kim

Jim Buffenbarger

CS354 - 001

December 5, 2017

Textbook Assignment 3

1. Question 6.1. We noted in Section 6.1.1 that most binary arithmetic operators are left associative in most programming languages. In section 6.1.4, however, we also noted that most compilers are free to evaluate the operands of a binary operator in either order. Are these statements contradictory? Why or why not?

   : Those statements are not contradictory. Those statements compiler evaluation of statements which doesn't directly affect the precedence that a language specifies arithmetic should be evaluated in. The only thing in which it would affect the mathematical result would be when the terms in an expression that is being evaluated are linked with each other like through a side effect function.

2. Question 6.2. As noted in Firgure 6.1, Fortran and Pascal give unary and binary minus the same level of precedence. Is this likely to lead to nonintuitive evaluations of certain expressions? Why or why not?

   : No. The same level of precedence to unary and binary minus is not leading to nonintuitive evaluations of certain expressions because of the left-to-right order of evaluation of expressions. The left-to-right order of evaluation of expressions causes the unary minus to group more tightly with it operand than other operators at the same level, thus eliminating the chances of nonintuitive evaluation.

3. Question 6.12. Describe a plausible scenario in which a programmer might wish to avoid short-circuit evaluation of a Boolean expression.

: The short circuit evaluation is a Boolean expression. It has the readability process to be increased. Short circuit doesn't compile time code and it works is done in the minimal amount in the process. In the operation perform && and || operator uses for both false and null values. Pascal code avoid the short circuit. However, & is more faster than && because && is used in bits operation.

4. Question 6.25. Consider a mid-test loop, here written in C, that looks for blank lines in its input:

```
for (;;) {
    line = read_line();
    if (all_blanks(line)) break;
    consume_line(line);
}
```

Show how you might accomplish the same task using a while or do (repeat) loop, if mid-test loops were not available. (Hint: One alternative duplicates part of the code; another introduces a Boolean flag variable.) How do these alternatives compare to the mid-test version?

: //using do loop
boolean new = false;
do {
    line = read_line();
    if(all_blacks(line)) {
        new = true;
        consume_line(line);
    }
} while(!new);
// using while loop
line = read_line();
while(!all_blanks(line)) {

```
        consume_line(line);
        line = read_line();
    }
```

I think the midtest loop is a better alternative than while and do loop. They are repeated because of lots of memory which is consumer which make the code ugly. Midtest loop is easy for the programmer to understand and run the program.

5. Question 6.26. Rubin[Rub87] used the following example (rewritten here in C) to argue in favor of a goto statement:

```
int first_zero_row = -1;        /* none */
int i, j;
for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
        if (A[i][j]) goto next;
    }
    first_zero_row = i;
    break;
next: ;
}
```

The intent of the code is to find the first all-zero row, if any, of an n*n matrix. Do you find the example convincing? Is there a good structured alternative in C? In any langu age?

```
: int first_zero_row = -1;
for(int i=0; i<n; i++)
{
        boolean non_zero = false;
        for(int j=0; j<n; j++)
        {
                if(A[i][j])
                {
                        non_zero = true;
                }
                if(first_zero_row == -1 && !non_zero)
                {
                        first_zero_row = i;
                }
        }
}
```

I think that it is not convincing and C is a good structured alternative.