John Pennig

Homework 9

1. Chapter 7 Review Questions:
    1. Define *operator precedence* and *operator associativity*. Operator precedence is the order of which the operators are evaluated in. Such as multiplication before addition. Operator associativity defines which operator is evaluated first when it is adjacent with another occurrence of an operator of the same precedence.

    2.What is a ternary operator? A ternary operator is one that has three operands.

    3. What is a prefix operator? A prefix operator is one that precedes its operand.

    4. What operator usually has right associativity? The exponentiation operator usually has right associativity.

    5. What is a nonassociative operator? A nonassociative operator is where the order of operations matters when multiple operators of the same type are chained together.

    8. Define *functional side effect*. A functional side effect is when a function changes one of its parameters or a global variable.

    9. What is a coercion? Coercion is defined as an implicit type conversion that is initiated by the compiler of run-time system. Such as when adding a integer and a float.

    12. Define *narrowing* and *widening conversions*. A narrowing conversion is when a type is converted into another type that cannot store even approximations of all the values of the original. Such as double to float. A widening conversion is when a type is converted into another type that can include at least approximations of all the values of the original. Such as int to float.

    15. What is referential transparency? Referential transparency is if any two expressions in the program that have the same value can be substituted for one another anywhere in the program without affecting the action of the program.

    16. What are the advantages of referential transparency? The semantics of referential transparent programs are much easier to understand than those without. Functions are equivalent to a mathematical function in terms of ease of understanding.

17. How does operand evaluation order interact with functional side effects? Operand evaluation order determines if a value is changed in a function before evaluation or after evaluation. Such an example is the expression “a + fun(a)” where fun(a) can change the value of a if evaluated before the addition. If a = 10 and fun(a) = 20, then the expression if fun(a) is evaluated first is 30. Otherwise, if fun(a) is not evaluated and instead takes the current value of a, then a + fun(a) = 20.

2. Chapter 7 Problem Set:

9. a. $a * b - 1 + c \rightarrow (((a * b)^1 - 1)^2 + c)^3$

b. $a * (b - 1) / c \bmod d \rightarrow (((a * (b - 1)^1)^2 c)^3 \bmod d)^4$

c. $(a - b) / c \,\&\, (d * e / a - 3) \rightarrow (((a - b)^1 / c)^5 \,\&\, (((d * e)^2 / a)^3 - 3)^4)^5$

d. $-a \text{ or } c = d \text{ and } e \rightarrow (((-a)^1 \text{ or } (c = d)^2)^3 \text{ and } e)^4$

e. $a > b \text{ xor } c \text{ or } d <= 17 \rightarrow (((a > b)^1 \text{ xor } c)^3 \text{ or } (d <= 17)^2)^4$

f. $-a + b \rightarrow (-(a + b)^1)^2$

13. a. sum1: (10 / 2) + fun (10) = 5 + 41 = 46

sum2: fun(14) + (18 / 2) = 53 + 9 = 62

b. sum1: (14 / 2) + fun(10) = 41 + 7 = 48

sum2: fun(14) + (14 / 2) = 53 + 7 = 60

19. a. x = 3 + fun(3) = 3 + 4 = 7

b. x = 8 + fun(3) = 8 + 3 = 11

3. Consider the precedence chart for C-based languages in Section 7.5.2 of the text. Explain the rationale for this order of precedence. Specifically, why do unary operators have the highest precedence?

This order of precedence follows mathematical rules of arithmetic. The unary operators have the highest precedence because they only belong to that one variable. Also typically incremented or decrementing a variable is done before any mathematical

expression due to the expression often being contingent on that variable being the value after incrementing or decrementing.

Why are arithmetic ops done in the specified order? Why are all arithmetic ops done before relational ops, and logical ops at the end? (It may be helpful to search these on the internet. A good starting point is https://en.wikipedia.org/wiki/Order_of_operations)

Arithmetic ops are done in the specified order to avoid any notational ambiguity. Such as what would happen if you negate (!) an entire solution to an expression rather than just the one variable it was attached to. Arithmetic ops are done before relational ops because the expression 1 + 2 < 1 + 3 would compare the expression 2 < 1 before the addition saying the expression is false, when after arithmetic operation it would be true. Therefore, relational ops are done after arithmetic ops. The same reason is why logical ops are at the end to let the full arithmetic operation to be performed to eliminate ambiguity and to ensure correctness of the program.