

Problem 7.1

7. Describe a situation in which the add operator in a programming language would not be commutative.

This might be the case if one of the operands had a side effect that affected the value of the other operand. Consider, for example, the following C program.

```
int scaleIt(int *k, int s)
{
    *k *= s;
    return *k;
}

int main(void)
{
    int m;

    m=10;
    printf("m = %i\n", scaleIt(&m, 2) + scaleIt(&m, 3));

    m=10;
    printf("m = %i\n", scaleIt(&m, 3) + scaleIt(&m, 2));

    return 0;
}
```

Problem 7.8

8. Describe a situation in which the add operator in a programming language would not be associative.

If overflow/underflow were to happen when operators were performed in a different order or, when working with floating point values, if roundoff error were to accumulate differently. The classic example is when small floating point values are added to a large value as opposed to being added together first before being added to the large value.

Problem 7.9

9. Assume the following rules of associativity and precedence for expressions:

<i>Precedence</i>	<i>Highest</i>	* , / , not
		+ , - , & , mod
		- (unary)
		= , /= , < , <= , >= , >
		and
	<i>Lowest</i>	or , xor
<i>Associativity</i>	<i>Left to right</i>	

Show the order of evaluation of the following expressions by parenthesizing all subexpressions and placing a superscript on the right parenthesis to indicate order. For example, for the expression

a + b * c + d

the order of evaluation would be represented as

$$((a + (b * c)^1)^2 + d)^3$$

- a. a * b - 1 + c
- b. a * (b - 1) / c mod d
- c. (a - b) / c & (d * e / a - 3)
- d. -a or c = d and e
- e. a > b xor c or d <= 17
- f. -a + b

NOTE: Assuming () has highest precedence – was not specified.

- a) (((a * b)¹ - 1)² + c)³
- b) ((a * (b - 1)¹)² / c)³ mod d)⁴
- c) ((((a - b)¹ / c)³ & ((d * e)² / a)⁴ - 3)⁵)⁶
- d) ((-a)¹ or ((c = d)² and e)³)⁴
- e) (((a > b)¹ xor c)³ or (d <= 17)²)⁴
- f) (- (a + b)¹)²

Problem 7.10

10. Show the order of evaluation of the expressions of Problem 9, assuming that there are no precedence rules and all operators associate right to left.

- a) $(a * (b - (1 + c)^1)^2)^3$
- b) $(a * ((b - 1)^1 / (c \bmod d)^2)^3)^4$
- c) $((a - b)^4 / (c \& (d * (e / (a - 3)^1)^2)^3)^5)^6$
- d) $(-(a \text{ or } (c = (d \text{ and } e)^1)^2)^3)^4$
- e) $(a > (b \text{ xor } (c \text{ or } (d \leq 17)^1)^2)^3)^4$
- f) $(-(a + b)^1)^2$

Problem 7.11

11. Write a BNF description of the precedence and associativity rules defined for the expressions in Problem 9. Assume the only operands are the names *a*, *b*, *c*, *d*, and *e*.

```

<or-expr> : <and-expr>
           | <or-expr> or <and-expr>
           | <or-expr> xor <and-expr>
<and-expr> : <rel-expr>
           | <and-expr> and <rel-expr>
<rel-expr> : <unary-exp>
           | <relational-exp> = <unary-exp>
           | <relational-exp> /= <unary-exp>
           | <relational-exp> < <unary-exp>
           | <relational-exp> <= <unary-exp>
           | <relational-exp> >= <unary-exp>
           | <relational-exp> > <unary-exp>
<unary-exp> : <add-expr>
           | - <unary-exp>
<add-expr> : <mult-expr>
           | <add-expr> + <mult-expr>
           | <add-expr> - <mult-expr>
           | <add-expr> & <mult-expr>
           | <add-expr> mod <mult-expr>
<mult-expr> : <factor>
           | <mult-expr> * <factor>
           | <mult-expr> / <factor>
           | not <mult-expr>
<factor> : (<or-expr>)
           | a | b | c | d | e | constant

```

Problem 7.12

12. Using the grammar of Problem 11, draw parse trees for the expressions of Problem 9.

NOTE: The trees below do not enclose nonterminals in angle brackets due to the limitations of the online tree generator used.

(a) $a * b - 1 + c$	(b) $a * (b - 1) / c \bmod d$
(c) $(a - b) / c \& (d * e / a - 3)$	(d) $-a \text{ or } c = d \text{ and } e$
(e) $a > b \text{ xor } c \text{ or } d \leq 17$	(f) $-a + b$

Problem 7.13

13. Let the function fun be defined as

```
int fun(int *k) {
    *k += 4;
    return 3 * (*k) - 1;
}
```

Suppose fun is used in a program as follows:

```
void main() {
    int i = 10, j = 10, sum1, sum2;
    sum1 = (i / 2) + fun(&i);
    sum2 = fun(&j) + (j / 2);
}
```

What are the values of sum1 and sum2

- a. if the operands in the expressions are evaluated left to right?
- b. if the operands in the expressions are evaluated right to left?

Part a) Left to Right					Part b) Right to Left				
eval	i	j	sum1	sum2	eval	i	j	sum1	sum2
initialize	10	10	?	?	initialize	10	10	?	?
(i/2)=5					fun(&i)=41	14			
fun(&i)=41	14				(i/2)=7				
sum1=46			46		sum1=48			48	
fun(&j)=41		14			(j/2)=5				
(j/2)=7					fun(&j)=41		14		
sum2=48				48	sum2=46				46
Final	14	14	46	48		14	14	48	46

Problem 7.18

18. Should an optimizing compiler for C or C++ be allowed to change the order of subexpressions in a Boolean expression? Why or why not?

No, it should not. The language specification for C/C++ includes guaranteed short circuiting, meaning that Boolean expressions will be evaluated in a specific order and only evaluated as far as required to determine the overall result of the expression. Rearranging the order will violate this guarantee.

Problem 7.20

20. Consider the following C program:

```
int fun(int *i) {  
    *i += 5;  
    return 4;  
}  
  
void main() {  
    int x = 3;  
    x = x + fun(&x);  
}
```

What is the value of x after the assignment statement in main, assuming

- a. operands are evaluated left to right.
- b. operands are evaluated right to left.

Part a) Left to Right	Part b) Right to Left		
eval	x	eval	x
initialize	3	initialize	3
x=3		fun(&x)=4	8
fun(&x)=4	8	x=8	
x=x+fun(&x)	7	x=x+fun(&x)	12
Final	7		12