Chapter 7: (Problem Set: Pages 306 – 308)

3) Do you think the elimination of overloaded operators in your favorite language would be beneficial? Why or why not?
Answer:

- Yes, I think that the elimination of overloaded operators in my favorite language would be beneficial.
- Elimination of overloaded operators will result in:
    - Enhance and increase readability.
    - Minimize the compiler overhead (choosing the correct operator meaning).

---

4) Would it be a good idea to eliminate all operator precedence rules and require parentheses to show the desired precedence in expressions? Why or why not?
Answer:

- No, it is not a good idea to eliminate all operator precedence rules.
- Although this idea sounds good, but in fact it affects the flexibility provided by the language since only one way is used to show the precedence in expressions. Moreover, using parentheses will future more affects writability and in some cases readability.

---

10) Assume the following rules of associativity and precedence for expressions:

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

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

Expression	evaluated	*Order of evaluation*
a. **a * b - 1 + c**	→	$(((a * b)^{1} - 1)^{2} + c)^{3}$
b. **a * (b -1) / c mod d**	→	$(((a * (b - 1)^{1})^{2} / c)^{3} \bmod d)^{4}$
c. **(a - b) / c & (d * e / a – 3)**	→	$(((a - b)^{1} / c)^{2} \,\&\, ((d * e)^{3} / a)^{4} - 3)^{5})^{6}$
d. **–a or c = d and e**	→	$((-a)^{1}$ or $((c = d)^{2}$ and $e)^{3})^{4}$
e. **a > b xor c or d <= 17**	→	$(((a > b)^{1}$ xor $c)^{3}$ or $(d <= 17)^{2})^{4}$
f. **–a + b**	→	$(- (a + b)^{1})^{2}$

12) Write a BNF description of the precedence and associativity rules defined for the expressions in Problem 10. Assume the only operands are the names a, b, c, d, and e.
Answer:

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

<expression> → <expression> or <or_exp>
| < expression > xor <or_exp>
| <or_exp>

<or_exp> → <or_exp> and <and_exp>
| <and_exp>

< and_exp > → <and_exp> = <expr>
| <and_exp /= <expr>
| <and_exp> < <expr>
| <and_exp> <= <expr>
| <and_exp> >= <expr>
| <and_exp> > <expr>
| <expr>

<expr> → - <unary_expr>
| <unary_expr>

<unary_expr> → <unary_expr> + <term>
| <unary_expr> - <term>
| <unary_expr> & <term>
| <unary_expr> mod < term>
| <term>

< term > → < term > * < factor >
| < term > / < factor >
| not < factor >
| < factor >

< factor > → (< expr >)
| <operand>

<operand> → a | b | c | d | e

Chapter 8: (Problem Set: Pages 345 – 347)

8) What are the pros and cons of using unique closing reserved words on compound statements?

Answer:

Using reserved word on compound statements will slightly affect writability, but in the other hand, it will enhance and increase language readability and flexibility and those two issues are significant.

10.d) Rewrite the following code segment using a loop structure in the following languages C, C+ + , or Java:

```
k := (j + 13) / 27
loop:
        if k > 10 then goto out
        k := k + 1
        i := 3 * k - 1
        goto loop
out: ...
```

Assume all variables are integer type. Discuss which language, for this code, has the best writability, the best readability, and the best combination of the two.

Answer:

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

12.d) Rewrite the following code segment using a multiple selection statement in the following languages C, C++, or Java:

```
if (k = 1) or (k = 2) then j := 2 * k - 1
if (k = 3) or (k = 5) then j := 3 * k + 1
if (k = 4) then j := 4 * k - 1
if (k = 6) or (k = 7) or (k = 8) then j := k - 2
```

Assume all variables are integer type. Discuss the relative merits of the use of these languages for this particular code.

Answer:

```
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: System.out.println("invalid input, "+k+" case is not handled!!!");
} //end switch
```

The merits of the use of JAVA language for this particular code are that JAVA language provides sequential execution to cases that are not separated by break special word and also JAVA provides a default value if the input is not included in the implemented cases.

16) Consider the following Pascal case statement. Rewrite it using only two-way selection.

```
case  index - 1 of
        2, 4: even :=  even + 1;
        1, 3: odd :=  odd + 1;
        0: zero :=  zero + 1;
        else error := true
end
```

Answer:

```
input = index – 1;
if (input == 2 || input == 4)
        even += 1;
else {
        if (input == 1 || input == 3)
                odd += 1;
        else
                if (input == 0)
                        zero += 1;
                else
                        error = true;
} // end else block
```

---

17) Consider the following C program segment. Rewrite it using no gotos or breaks.

```
j = -3;
for (i = 0; i < 3; i++) {
        switch (j + 2) {
                case 3:
                case 2: j--; break;
                case 0: j += 2; break;
                default: j = 0;
        } //end switch
        if (j > 0) break;
        j = 3 – i
} //end for-loop block
```

Answer:

The above given code can be re-written using no **goto**s or **break**s as follows:

```
int j = -3; int i;
for (i = 0 ; i < 3 ; i++) {
        if ( (j+2) == 3 || (j+2) == 2)
                j--;
        if ( (j+2) == 0 )
                j += 2;
        else
                j = 0;
        if (j > 0)
                i = 2;
        else
                j = 3 - i;
}
```

18) In a letter to the editor of CACM, Rubin (1987) uses the following code segment as evidence that the readability of some code with gotos is better than the equivalent code without gotos. This code finds the first row of an n x n integer matrix named x that has nothing but zero values.

```
for  i  :=  1 to n do
        begin
        for  j  :=  1 to n do
                if x[i, j] <> 0
                        then goto reject;
                writeln('First all-zero row is:', i);
                break;
reject:
        end;
```

Rewrite this code without *goto*s in one of the following languages: C, C++, Pascal, Java, or Ada. Compare the readability of your code to that of the code above.
Answer:

```
for ( int i = 0, int j = 0 ; i < n && j < n ; i++ )
        for ( j = 0; j < n && x[i][j] != 0; j++);
                if ( i < n )
                        System.out.println("First all-zero row is: "+ i );
```

I have the feeling that this code doesn't look readable as the one that is given and this is maybe because the nature of for-loop I am using (compact and many expressions to be evaluated grouped in one line).

End homework 4