1) 6.5

   Assuming that it does right to left exponentiation and that expt can take multiple arguments, explicit use of parenthesis is necessary to get the expected outcome.

   ```
   (expt (expt 3 4) 5)
   ```

2) 6.12

   if (a!=0 && a.next == true)

   Rewritten to work without short-circuiting:

   ```
   if (a!=0)
      if(a.next == true)
            //"if" code
      fi
   else
      //code after "if" code
   ```

3) a/b > 0 && b/a < 0
   a. evaluation with a == 0:

      the first part of the expression will evaluate to 0 which means the boolean will short circuit and the rest of the conditional statement.

      statement = 0

      evaluation with b==0:

      Making b == 0 is going to cause an error because it will attempt to divide by 0

   b. It would make sense, because it would allow the programmer to not worry about accidentally dividing by zero

4) The argument isn't convincing because it gives a programmer too much brevity to make code that cannot be optimized. I C, the best way to do what he is talking about would be to use the continue statement. Many other languages have similar control structures

5) MIPS for do-while
   a. The Language uses short-circuit evaluation

```
DOWHILE:
        #loop body
        li    $t0, -20
        lw    $t1,-8(sp)
        ble   $t1,$t0, WHILE_OUT
        li    $t0,-10
        blt   $t1,$t0, WHILE
        li    $t0,20
        bgt   $t,$t0,while
WHILE_OUT:
```

   b. The language doesn't use short-circuit evaluation

```
DOWHILE:
        #loop body
        li    $t0,-20
        lw    $t1,-8(sp)
        sgt   $t2,$t1,$t0
        li    $t0,-10
        slt   $t3,$t1,$t0
        li    $t0,20
        sgt   $t4,$t1,$t0
        or    $t3,$t3,$t4
        and   $t2,$t2,$t3
        bnez  $t2,DOWHILE
        #after loop
```

6) MIPS for C switch

```
      T        .word       0
               .word       .L1
               .word       .L2
               .word       .L4
               .word       .L3
               .word       .L3

               lw          $t0,-8(sp)
               subi        $t0,$t0,199
               blez        $t0 .L4
               li          $t1,5
               bgt         $t0,$t1,.L4
               la          $t1,T
               mul         $t0,$t0,4
               add         $t1,$t2,$t0
               lw          $t2,($t1)
               jr          $t2
   .L1
         #code for foo1();
   .L2
         #code for foo2();
         j     .L5
   .L3
         #code for foo3();
         j     .L5
   .L4
         #code for foo4();
   .L5         nop
```