



— Page references correspond to locations of Extra Examples icons in the textbook.

---

**p.194, icon at Example 1**

**#1.**

(a) Describe an algorithm that determines the location of the last even integer in a nonempty list  $a_1, a_2, \dots, a_n$  of integers. (If no integer in the list is even, the output should be that the location is 0.)

(b) Describe the algorithm, with “last” replaced by “first”.

**Solution:**

(a) We need to find the last subscript,  $i$ , such that  $a_i$  is even, that is,  $a_i \bmod 2 = 0$ . We use *location* to keep track of the subscript. Initially we set *location* to 0 (because an even integer has not yet been found), and then proceed to examine each element of the list by advancing the subscript  $i$  one step at a time, until the end of the list is reached. Here is the pseudocode:

```
location := 0           {location is initially set to 0}
for i := 1 to n         {examine, in order, each entry a_i in the list}
if a_i mod 2 = 0 then location := i {change location to i if a_i is even, otherwise keep old location}
```

(b) Suppose we seek the location of the *first* even integer in the list. In this case the loop should end once an even integer  $a_i$  is encountered or else all entries in the list have been examined and no even integer has been encountered. We can use a while-loop

```
location := 0           {location is initially set to 0}
i := 1                 {begin by examining first element in the list}
while (location = 0 and i ≤ n) {as long as no even element has been found and there are
                                more elements in the list yet to be examined}
begin
  if a_i mod 2 = 0 then location := i {examine element a_i; if it is even, update the location}
  i := i + 1                       {advance counter to examine next element}
end
```

---

**p.194, icon at Example 1**

**#2.** Describe an algorithm that takes as input a sequence of distinct integers  $a_1, a_2, \dots, a_n$  ( $n \geq 2$ ) and determines if the integers are in increasing order.

**Solution:**

One way to do this is to examine each pair of consecutive integers,  $a_{i-1}$  and  $a_i$ , to see if  $a_i < a_{i-1}$ . If this happens, the integers are not in increasing order, and we stop and output FALSE. If this never happens, then the output remains TRUE.

```
output := TRUE
i := 2
while (i ≤ n and output = TRUE)
begin
```

```

    if  $a_{i-1} > a_1$  then  $output := FALSE$ 
     $i := i + 1$ 
end

```

---

**p.194, icon at Example 1**

**#3.** Describe an algorithm that takes as input a positive integer  $n$  and gives as output the tens' digit of  $n$ . For example, if the input is the positive integer 3752, the output is 5; if the input is the positive integer 4, the output is 0 (because we can think of 4 as 04).

**Solution:**

Suppose that the number  $n$  is  $a_k a_{k-1} \dots a_2 a_1 a_0$  when written as a string of digits. We want to find  $a_1$ . For example, suppose we start with 3752. We want the output 5. We first subtract  $3752 - 3700$ , which removes all digits to the left of the tens' digit, and obtain 52. We next subtract the units' digit, 2, and divide by 10, obtaining the tens' digit, 5. These subtractions can be carried out using multiples of the floor function.

We use the floor function to compute the numbers consisting of the tens' and units' digit,  $a_1 a_0$ , and the units' digit,  $a_0$ , of  $n$ . We then subtract,  $a_1 a_0 - a_0$ , and divide by 10 to obtain the tens' digit,  $a_1$ .

$$x := n - 100 \left\lfloor \frac{n}{100} \right\rfloor \quad \{ x \text{ is the number } a_0 a_1, \text{ consisting of the tens' and units' digits of } n \}$$

$$y := x - 10 \left\lfloor \frac{x}{10} \right\rfloor \quad \{ y \text{ is the units' digit of } a_1 a_0 \}$$

$$z := (x - y)/10 \quad \{ z \text{ is the tens' digit } \}$$

(Observe that if the number consists of a single digit, such as 7, then  $x = 7$  and  $z = 0$  as it should be in a case such as this.)

---

**p.194, icon at Example 1**

**#4.** Describe an algorithm that takes as input a list of integers  $a_1, a_2, \dots, a_n$  (where  $n > 2$ ) and determines if some  $a_i$  is equal to the average of an earlier entry in the list and a later entry in the list.

**Solution:**

The algorithm must take each "inside" element  $a_i$  (where  $1 < i < n$ ) and examine the sublist to the left of  $a_i$  and the sublist to the right of  $a_i$ . The algorithm must check each  $a_j$  ( $1 \leq j < i$ ) and  $a_k$  ( $i < k \leq n$ ) to see whether  $a_i = (a_j + a_k)/2$ .

```

answer := FALSE
i := 2
while (answer = FALSE and  $i < n$ )
begin
    j := 1
    while ( $j < i$  and answer = FALSE)           { examine entries to the left of  $a_i$  }
    begin
        k := i + 1
        while ( $k \leq n$  and answer = FALSE)     { examine entries to the right of  $a_i$  }
        begin
            if  $a_i = \frac{a_j + a_k}{2}$  then answer := TRUE
        end
    end
end

```

```
         $k := k + 1$   
    end  
end  
end
```

---