Introduction to computational programming

Introductory Exercise

Loops and flow control *R*

Solutions

David M. Rosenberg
University of Chicago
Committee on Neurobiology

Version control information:
Last changed date: 2009-10-02 14:22:47 -0500 (Fri, 02 Oct 2009)
Last changes revision:72
Version: Revision 72
Last changed by: David M. Rosenberg

October 6, 2009

1. (a) Starting with the fibonacci sequence examples from the tutorial, write a code chunk which takes as input a number `k` and finds the largest fibonacci number less than or equal to `k`

The intent of this exercise was for you to see how more complex computations can be built up from simpler ones.

```
> x0 <- 0;
> x1 <- 1;
> while(x1 < k){
+   next_value <- x0 + x1;
+   x0 <- x1;
+   x1 <- next_value;
+ }
> if (x1 == k) {
+   cat(k, ' is a fibonnaci number')
+ } else {
+   cat(x0, ' is the largest fibonnaci number <= ', k)
+ }
```

   (b) Write a code chunk which takes as input three numbers (say, `k`, `x`, and `y`) and prints either the second number (`x`) or the third number (`y`) depending on which is closer to `k`. (*Hint:* the functions `min()`, `max()`, and `abs()` may be helpful. See the online documentation for their usage.)

```
> if(abs(x-k) < abs(y-k)) {
+   cat(x, ' is closer to ', k, '\n');
+ } else if (abs(x-k) == abs(y-k)) {
+   cat(x, ' and ', y, ' are equidistant from ', k, '\n')
+ } else {
+   cat(y, ' is closer to ', k, '\n')
+ }
```

(c) Write a code chunk which takes as input a number `k` and returns the fibonacci number with is closest to `k`.

Combining the previous parts

```
> x0 <- 0;
> x1 <- 1;
> while(x1 < k){
+   next_value <- x0 + x1;
+   x0 <- x1;
+   x1 <- next_value;
+ }
> if (x1 == k) {
+   cat(k, ' is a fibonnaci number')
+ } else if(abs(x0-k) < abs(x1-k)){
+   cat(x0, ' is the fibonnaci number closest to ', k)
+ } else {
+   cat(x1, ' is the fibonnaci number closest to ', k)
+ }
```

2. (a) Write a code chunk which takes a “sorted” numeric vector of length 2 and another numeric vector of length 1 and prints a single “sorted” vector of length 3.

```
> ## Given these two 'inputs'
> sorted <- c(a, b) ## a < b
> to_add <- d       ## new value
> if(d < a) {
+   c(d, a, b)
+ } else if (d < b) {
+   c(a, d, b)
+ } else {
+   c(a, b, d)
+ }
```

(b) Write a code chunk which takes an “unsorted” numeric vector of length 2 and prints the values of that vector, “sorted.”

```
> ## unsorted input
> unsorted <- c(a, b)
> if (a <= b) {
+   c(a, b)
+ } else {
+   c(b, a)
+ }
```

(c) Write a code chunk that takes an "unsorted" numeric vector of length 10 and prints the sorted values to the screen.

Don't worry about the first line of code. I just use it here to generate an arbitrary unsorted vector of length 10 to demonstrate the validity of this code.

```
> ## input
> unsorted <- floor(runif(n=10, min=1, max=100))
> unsorted

 [1]  5 19 36 62 88 28 85 67 76 25

> sorted <- numeric(length=length(unsorted))
> for (ii in 1:9) {
+   smallest <- unsorted[1]
+   unsorted <- unsorted[-1]
+   for (jj in 1:length(unsorted)) {
+     if (unsorted[jj] < smallest) {
+       temp <- unsorted[jj]
+       unsorted[jj] <- smallest
+       smallest <- temp
+     }
+   }
+   sorted[ii] <- smallest;
+ }
> sorted[10] <- unsorted
> sorted

 [1]  5 19 25 28 36 62 67 76 85 88
```

3. Consider the equation

$$x^k - 1 = 0, \quad x \in \mathbb{C}, k \in \mathbb{N} \tag{1}$$

Write a code chunk that takes in integer (k) as input and prints all values x which satisfy the above equation.

$$
\begin{aligned}
(a_k) &= a_1, a_2, \ldots \\
\text{and } a_k &= b_k + c_k\, i, \quad b_k, c_k \in \mathbb{Q} \\
&= \sqrt{b_k^2 + c_k^2} e^{i\left(\text{arctan}\left(\frac{b_k}{c_l}\right)\right)} \\
\text{Let } r_k &= \sqrt{b_k^2 + c_k^2} \text{ and } \psi_k = \arctan\left(\frac{b_i}{c_i}\right) \\
\text{Then } a_k &= r_k e^{\psi_k i} \\
a_k^n &= \left(r_k e^{\psi_k i}\right)^n \\
&= {r_k}^n e^{ni\psi_k}
\end{aligned}
$$

Since $x^n - 1, {r_k}^n = 1$ and $e^{ni\psi_k} = 0 \pm 2c\pi$, where $c \in \mathbb{Z}$, we know that ${r_k}^n = 1$ and further, $r_k = 1$. We know that $0 \leq \psi_k \leq 2\pi$. A first two solutions are found by solving

$$
\begin{aligned}
n\psi_k &= 2\pi & n\psi_k &= 0 \\
\psi_k &= \frac{2\pi}{n} & \psi_k &= 0
\end{aligned}
$$

We can then begin to generate solutions by sequentially adding the second solution to the first until we no longer get unique solutions modulo 2π.

$$\psi_0 = 0$$
$$\psi_1 = \frac{2\pi}{n}$$
$$\psi_2 = 2\left(\frac{2\pi}{n}\right)$$
$$\vdots$$
$$\psi_n = n\left(\frac{2\pi}{n}\right) = 2\pi \equiv 0$$

Therefore $\forall n \in \mathbf{Z} : 0 \leq n \leq k$, the complex number $e^{\frac{2ni\pi}{n}}$ is a solution for $x^k - 1 = 0$. In code

```
> ## As an example
> complexRootsOfUnity <- function(k) {
+   args <- seq(from=0, to=k-1, by=1) * 2 * pi / k
+   mods <- 1;
+   solutions <- complex(modulus=mods, argument=args)
+ }
> roots_15 <- complexRootsOfUnity(15)
> roots_15

 [1]  1.0000000+0.0000000i  0.9135455+0.4067366i
      0.6691306+0.7431448i
 [4]  0.3090170+0.9510565i -0.1045285+0.9945219i
      -0.5000000+0.8660254i
 [7] -0.8090170+0.5877853i -0.9781476+0.2079117i
      -0.9781476-0.2079117i
[10] -0.8090170-0.5877853i -0.5000000-0.8660254i
     -0.1045285-0.9945219i
[13]  0.3090170-0.9510565i  0.6691306-0.7431448i
     0.9135455-0.4067366i

> roots_15^15

 [1] 1+0i 1-0i 1-0i 1-0i 1-0i 1-0i 1-0i 1-0i 1-0i 1-0i 1-0i 1-0
      i 1-0i 1+0i 1+0i

> roots_3 <- complexRootsOfUnity(3)
> roots_3

[1]  1.0+0.0000000i -0.5+0.8660254i -0.5-0.8660254i

> roots_3^3

[1] 1+0i 1-0i 1-0i
```