Advanced Control Flow

Jiseok Chae

Department of Mathematical Sciences
KAIST

Week 6

Contents

Loops can be nested.

double_for.m

```
for i = 1 : 3
    for j = 4 : 5
        disp([i, j]);
    end
end
```

```
>> double_for

     1     4
     1     5
     2     4
     2     5
     3     4
     3     5
```

The following function `my_add(A, B)` computes the sum $A + B$, by using a double loop to iterate over all indices (i, j).

my_add.m

```
function C = my_add(A, B)
if size(A) == size(B)  % size(A) is the size of A
    [m, n] = size(A);
    C = zeros(m, n);
    for i = 1 : m
        for j = 1 : n
            C(i, j) = A(i, j) + B(i, j);
        end
    end
else
    disp('size of A and B are different.');
end
```

The following function `my_prod(A, B)` computes the matrix product `AB`, by using a triple loop.

my_prod.m

```
function C = my_prod(A, B)
[m, p] = size(A); [q, n] = size(B);
if p == q
    C = zeros(m, n);
    for i = 1 : m
        for j = 1 : n
            entry = 0;
            for k = 1 : p
                entry = entry + A(i, k) * B(k, j);
            end
            C(i, j) = entry;
        end
    end
else
    disp('input matrices have imcompatible sizes.')
end
```

Loops can be terminated in the middle of execution, by the command `break`. A typical usage of `break` is as follows.

```
for i = ⟨range of i⟩
    ⟨do something...⟩
    if ⟨P⟩
        break
    end
    ⟨if loop is not broken then do more things...⟩
end
```

During the loop, if $\langle P \rangle$ is met, then `break` command is called. In that case, the loop is terminated immediately, without executing further commands in the loop of the body nor further iterations.

Let us consider a modification of the `sqsum` function, where the loop is terminated when the partial sum exceeds 100.

broken_sqsum.m

```
function res = broken_sqsum(n)
res = 0;
for i = 1 : n
    res = res + i * i;
    if res > 100
        break
    end
end
```

Since $\sum_{i=1}^{6} i^2 = 91$ and $\sum_{i=1}^{7} i^2 = 140$, the value of `broken_sqsum(n)` with `n` ≥ 7 will always be 140.

```
>> broken_sqsum(6)

ans =

     91

>> broken_sqsum(7)

ans =

    140

>> broken_sqsum(8)

ans =

    140
```

Contents

We know that, when writing a function, we can call other functions inside the function body.

In fact, a function may also call itself inside its own function body. This is called a *recursion*.

Recursion is useful when, while solving a certain problem, we need a solution for the same problem but with different inputs.

Suppose that we are given two nonnegative integers a and b. We want to compute their greatest common divisor, which we denote by $\gcd(a, b)$.

Suppose that, for two nonnegative integers q and r, it holds that

$$a = qb + r.$$

One can show that $\gcd(a, b) = \gcd(b, r)$.

If $b > 0$, by taking r to be the remainder when a is divided by b, we can ensure that $b > r$.

Therefore, we can transform a $\gcd$ problem into another $\gcd$ problem, but with the second input strictly decreased.

So, transforming repeatedly, we will eventually reach $r = 0$.

But if $r = 0$, then $\gcd(b, r) = \gcd(b, 0) = b$, so this means we are done.

An example:

$$\begin{aligned}
\gcd(221, 289) &= \gcd(289, 221) && 221 = 0 \times 289 + 221 \\
&= \gcd(221, 68) && 289 = 1 \times 221 + 68 \\
&= \gcd(68, 17) && 221 = 3 \times 68 + 17 \\
&= \gcd(17, 0) && 68 = 4 \times 17 + 0 \\
&= 17 && r = 0, \text{ so we are done.}
\end{aligned}$$

We now translate our discussion into MATLAB code.

my_gcd.m

```
function res = my_gcd(a, b)
    if b == 0
        res = a;
    else
        r = rem(a, b);
        res = my_gcd(b, r);
    end
end
```

When you use recursion, make sure the function eventually reaches the *base case*, i.e., the case where the function does not call itself anymore and gets out of it. For `my_gcd`, this corresponds to checking if `b == 0` and returning `a` immediately in that case.

Thank you!