6.4 Functions

Functions and procedures are very similar. They are both types of subprograms. They can both have parameters or not. They can both have local variables or not. The difference is that a function produces a single answer. The result of that function needs to be used in some way by the program. This means that a function is called in a different way than a procedure.

Consider the built in Turing function `strint`. It takes a string as a parameter and turns it into a `int`. We have to do something with that `int`. So we don't called `strint` like a procedure:

```
var s : string := "1234"
strint(s)      <--- This would cause an error.
```

We have to do one of three things:

```
var i : int
i := strint(s)     <--- (1) store the result of the function in a variable
put strint(s)      <--- (2) print the result of the function
if strint(s) > 1000 then   <--- (3) compare the result of the function with something
```

Now let's consider making our own function. We will make a function that will find the mean of 3 numbers:

```
function mean (a, b, c : real) : real
   result (a + b + c) / 3
end mean
```

Notice some differences with procedures. In the first line of the definition we use the word `function` instead of procedure. After we list the parameters we have to say what type of result this function is going to produce. We do this as if we are declaring a variable. Somewhere before the function is over we need a `result` statement to produce the result of the function that we will return to where we called the function. Here are some examples of how we could call the function:

```
put mean(1, 2, 3)      ---> prints 2
var x: real := 2.2
var y: real := -3.153
var answer : real
answer := mean(x, y, x + 5.7)
put answer         ---> prints 2.315667      (2.2 - 3.152 + 7.7) / 3
if mean(10, x, 35.1) > 10 then
   put "Whatever"
end if
mean(3.8, 6.2, 9.1)      X mean is not a procedure, so you need to do something with
                           the result
```

If we wanted to find the mean of any number of numbers, we would need to use an array as a parameter. Then we could rewrite the mean function like this:

```
function mean (a: array 1 .. * of int) : real
   var sum : int := 0
   for i : 1 .. upper(a)
      sum := sum + a(i)
   end for
   result sum / upper(a)
end mean
```

Suppose we wanted to find the largest of 3 numbers. There are two main approaches. One is to use a local variable and one `result` statement. It has the advantage of exiting the function at the end of the function:

```
function largest (a, b, c : int) : int
   var big : int
   if a > b and a > c then
      big := a
   elsif b > c then
      big := b
   else
      big := c
   end if
   result big
end largest
```

Once we know what the answer is we could return the result right away if we want to. It is possible to have multiple `result` statements. The drawback is that the function is a bit harder to understand when it has multiple possible exits. We could do it like this with no local variable needed and 3 `result` statements:

```
function largest (a, b, c : int) : int
   if a > b and a > c then
      result a
   elsif b > c then
      result b
   else
      result c
   end if
end largest
```

As soon as a function hits a `result` statement it returns to where you called it from and the rest of the function is not executed.

Read section 6.4 in the textbook. Do exercise 6.4 #1-8 (I'll post solutions on Thursday).

Keep checking the web site for the summative, it will be posted soon..