Computing with functions.

Computation?

How much do we need to perform some computation?
For example, some algorithm from your C++ homework assignment.

- conditional branching
- loops
- good to have some data structures
- variables and code abstraction (objects, functions)

Can we do it by using only functions?

For convenience we assume that we have natural numbers and the operator + for adding them.

Simple functions

Constant function

$$x \mapsto 23$$

Identity function

$$x \mapsto x$$

Successor function

$$x \mapsto x + 1$$

Function application

If f is a function, the usual notation

$$f(x)$$

denotes a function application to the argument x.

We are going to use a shorter notation for application:

$$f \ \ x$$

Application is left-associative (just like $+$, $-$, $\times$):

$$f \ \ x \ \ y \equiv (f \ \ x) \ \ y$$

$$f \ \ x \ \ y \ \ z \equiv ((f \ \ x) \ \ y) \ \ z$$

Function application

Applying our functions to the argument 7:

$$
\begin{aligned}
(x \mapsto 23)\ 7 &\implies 23 \\
(x \mapsto x)\ 7 &\implies 7 \\
(x \mapsto x + 1)\ 7 &\implies 7 + 1 \implies 8
\end{aligned}
$$

This is really boring! The computations are trivial.

Functions that return functions

Consider a function that takes an argument x and returns another function that always returns x

$$x \mapsto (y \mapsto x)$$

Applying it to the argument 5:

$$(x \mapsto (y \mapsto x))\ 5 \quad \Longrightarrow \quad y \mapsto 5$$

So the result of the application is a constant function $y \mapsto 5$.

Functions that return functions

Another example:

$$x \mapsto (y \mapsto y)$$

Applying it to the argument 12:

$$(x \mapsto (y \mapsto y))\ 12 \quad \Longrightarrow \quad y \mapsto y$$

It drops the argument and returns an identity function.

Two or more arguments

Addition function (using operator $+$ internally):

$$x \mapsto (y \mapsto x + y)$$

Applying it to the arguments 5 and 7:

$$\begin{aligned} (x \mapsto (y \mapsto x + y))\ 5\ 7 &\implies (y \mapsto 5 + y)\ 7 \\ &\implies 5 + 7 \\ &\implies 12 \end{aligned}$$

It also resembles sequential composition and variable binding:

$x = 5;$
$y = 7;$
return $x + y;$

Observation

Consider two previously mentioned functions:

$$x \mapsto (y \mapsto x)\ 1\ 2 \quad \Longrightarrow \quad 1$$

$$x \mapsto (y \mapsto y)\ 1\ 2 \quad \Longrightarrow \quad 2$$

Can we use this behavior for doing something useful?

Ifthenelse

Function *Ifthenelse*:

$$c \mapsto (a \mapsto (b \mapsto c \;\; a \;\; b))$$

Function *True*:

$$x \mapsto (y \mapsto x)$$

Function *False*:

$$x \mapsto (y \mapsto y)$$

$$\begin{aligned}
&\mathit{Ifthenelse} \;\; \mathit{True} \;\; 1 \;\; 2 \\
&\quad\Longrightarrow \quad \big(c \mapsto (a \mapsto (b \mapsto c \;\; a \;\; b))\big) \;\; \mathit{True} \;\; 1 \;\; 2 \\
&\quad\Longrightarrow \quad (a \mapsto (b \mapsto \mathit{True} \;\; a \;\; b)) \;\; 1 \;\; 2 \\
&\quad\Longrightarrow \quad (b \mapsto \mathit{True} \;\; 1 \;\; b) \;\; 2 \\
&\quad\Longrightarrow \quad \mathit{True} \;\; 1 \;\; 2 \\
&\quad\Longrightarrow \quad (x \mapsto (y \mapsto x)) \;\; 1 \;\; 2 \quad\Longrightarrow\quad (y \mapsto 1) \;\; 2 \quad\Longrightarrow\quad 1
\end{aligned}$$

Ordered Pair

How to construct ordered pairs?

We need to implement three function:

- Pair construction

$$\text{“}Pair\ \ a\ \ b = (a, b)\text{”}$$

- Projection function that returns the first element

$$\text{“}First\ \ (a, b) = a\text{”}$$

- Projection function that returns the second element

$$\text{“}Second\ \ (a, b) = b\text{”}$$

Ordered Pair

Function *Pair*:

$$a \mapsto (b \mapsto (c \mapsto c \ \ a \ \ b))$$

Function *First*:

$$x \mapsto x \ \ \mathit{True}$$

Function *Second*:

$$x \mapsto x \ \ \mathit{False}$$

$$\begin{aligned}
&\mathit{Second} \ \ (\mathit{Pair} \ \ 5 \ \ 7) \\
&\quad\Longrightarrow \quad (x \mapsto x \ \ \mathit{False}) \ \ (\mathit{Pair} \ \ 5 \ \ 7) \\
&\quad\Longrightarrow \quad (\mathit{Pair} \ \ 5 \ \ 7) \ \ \mathit{False} \\
&\quad\Longrightarrow \quad \big(a \mapsto (b \mapsto (c \mapsto c \ \ a \ \ b))\big) \ \ 5 \ \ 7 \ \ \mathit{False} \\
&\quad\Longrightarrow \quad (b \mapsto (c \mapsto c \ \ 5 \ \ b)) \ \ 7 \ \ \mathit{False} \\
&\quad\Longrightarrow \quad (c \mapsto c \ \ 5 \ \ 7) \ \ \mathit{False} \\
&\quad\Longrightarrow \quad \mathit{False} \ \ 5 \ \ 7 \ \ \Longrightarrow \ \ \cdots \ \ \Longrightarrow \ \ 7
\end{aligned}$$

Function M

Function M:

$$x \mapsto x \;\; x$$

It returns its argument applied to itself.

Let's apply this function to something. Any suggestions?

$$\begin{aligned}(x \mapsto x \;\; x)\ (y \mapsto y) &\implies (y \mapsto y)\ (y \mapsto y) \\ &\implies y \mapsto y\end{aligned}$$

Better suggestions?

Function M

Function M:

$$x \mapsto x \;\; x$$

Apply it to itself:

$$(x \mapsto x \;\; x) \;\; (x \mapsto x \;\; x) \quad \Longrightarrow$$

Function M

Function M:

$$x \mapsto x \ \ x$$

Apply it to itself:

$$\begin{aligned} (x \mapsto x \ \ x) \ (x \mapsto x \ \ x) &\implies (x \mapsto x \ \ x) \ (x \mapsto x \ \ x) \\ &\implies (x \mapsto x \ \ x) \ (x \mapsto x \ \ x) \\ &\implies \ldots \end{aligned}$$

This is an infinite loop, something like

while(true) { ; }

Based on this principle, we can implement real recursion and loops that actually do something.

Lambda calculus

This computational formalism is called lambda calculus. This is a universal model of computation, in the sense that your laptop cannot compute anything what cannot be computed in lambda calculus.

It was introduced by Alonzo Church in 1930s.

We need to fix the notation.

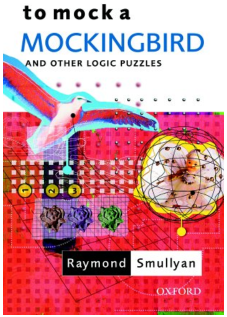
$$\text{Instead of} \quad (x \mapsto x \;\; y \;\; z),$$

$$\text{we write} \quad \lambda x\,.\,xyz$$

Also, you need to be careful with the names of the variables, to make substitutions correctly.

The order of evaluation (which function application gets reduced first?) is important, and it has to be defined precisely.

Further reading on the topic



"To Mock a Mockingbird and Other Logic Puzzles" (chapter 3)

by Raymond Smullyan

$$Mx \implies xx$$

Also, you can try learning functional programming languages like Scheme, Erlang, ML, or Haskell.

Almost every modern programming language (say, developed after 2000) has some functional features. Even JavaScript has Scheme-like functional core.