S6.1: Imperative OCaml Programming

CSci 2041:

Advanced Programming Principles

University of Minnesota,
Prof. Van Wyk,
Spring 2022

Effects

- "effects" are computations that change, or effect, the system (memory, file system, network, etc.)
- Nearly all features of OCaml are "effect free" or "pure" computations. These are expression that evaluate to a value without any "effect."

OCaml references

- A **reference** can be seen as a box whose contents can be updated to a new value. Similarly, it can be seen as a pointer to a memory location whose value can change.
- The operations:

```
val ref : a -> a ref
val (!) : a ref -> a
val (:=) : a ref -> a -> unit
```

ref

```
val ref :  a -> a ref
```

- A reference containing an integer has type `int ref`
- A reference is created by the `ref` operation.
- `let ri = ref 9`

!

```
val (!)  :  a ref -> a
```

- The `!` extracts the value of a reference.
- References *always* contain a value.
  This make them different from C or Java pointers/references which can be “null” and thus not point to a value.
- `! ri` evaluates to 9

:=

```
val (:=) :  a ref -> a -> unit
```

- Updating the contents of a reference.
- After `ri := 10`
  `! ri` evaluates to 10
- This operation returns the value `()` of type `unit`.
  This is the only value of this type.

Ordering of operations

We can specify the order of such operations as follows:

```
let ri = ref 9  in
let ten = !ri + 1 in
let () = ri := 20 in
let twenty = !r1 in
...
```


Some examples

- Circular structures - see `circular.ml` in the `Sample-Programs` directory of the public course repository.
- A doubly-linked list
  Consider the sample functions in `dllist.ml` in the `Sample-Programs` directory of the public course repository.