Type definition exercise

- **Given this new type**

```
type PersonalInfo = Address of int * string
                  | Phone of string | Age of int
```

define function `street: PersonalInfo` $\rightarrow$ `string` **that returns the street name for an address, and the empty string for any other kind of value:**

Recursive type definitions

- **In this type definition:**

$$\texttt{type}\ t\ \texttt{=}\ C_1\ \texttt{[of}\ te_1\texttt{]}\ \mid \ldots \mid C_n\ \texttt{[of}\ te_n\texttt{]}$$

the type expressions te_i can contain t, making the type declaration recursive. This allows for the definition of infinite data types, such as lists and trees, e.g.

```
type mylist = Empty | Cons of int * mylist
let list1 = Cons (3, Cons (4, Empty))
```

- **Ex: write the function** `sum` **:** `mylist` $\rightarrow$ `int`**.**

Exercises: Functions on binary trees

```
type bintree = Empty
             | Node of int * bintree * bintree
```

- **Define isLeaf: bintree $\rightarrow$ bool**

- **Define sum: bintree $\rightarrow$ int**