A.A. 2000-01

Linguaggi di Programmazione I: Paradigmi
Prova scritta del 19 giugno 2001: soluzioni

Esercizio 4.

```
abstype NAT = ZERO | SUCC of NAT

   with val zero = ZERO
        and uno = SUCC ZERO

        fun sum ZERO n = n
          | sum (SUCC m) n = SUCC (sum m n)

        fun equals ZERO ZERO = true
          | equals (SUCC m) (SUCC n) = equals m n
          | equals m n = false
   end;
```

L'interprete restituisce:

```
type NAT
val zero = - : NAT
val uno = - : NAT
val sum = fn : NAT -> NAT -> NAT
val equals = fn : NAT -> NAT -> bool
```

Notate il tipo di `sum`. Come fa SML a sapere che il *secondo* argomento deve essere di tipo `NAT`? Notate inoltre che la definizione di `equals` sfrutta l'ordinamento delle clausole: la terza clausola (applicabile per qualunque argomento) vale a dire: "e in tutti gli altri casi ... ".

Altra soluzione: implementare i naturali usando gli interi:

```
abstype NAT = N of int

   with val zero = N 0
        and uno = N ~1
        fun sum (N n) (N m) = N (n+m)
        fun equals (N n) (N m) = (n=m)
   end;
```

In SML/NJ "-1" si scrive `~1`. Nella soluzione proposta il "meno" è una complicazione tanto voluta quanto inutile: `1` sarebbe andato bene lo stesso.

Terza soluzione: usare una *abstraction*. Non tutte le implementazioni di SML supportano questo costrutto.

Esercizio 5. Nell'esecuzione del metodo `m()` il sistema runtime di Java risolve l'invocazione `super.x()` cercando il metodo `x()` a partire dalla superclasse di `B`, ovvero della classe in cui è definito il metodo che contiene l'invocazione. Dunque in questo caso si invoca la `x()` della classe `A`. Per quanto riguarda l'invocazione `this.x()`, la pseudovariabile `this` denota un oggetto della classe `C`, e dunque si esegue il metodo `x()` ivi definito. Il risultato è dunque: 10.