

## Paradygmaty programowania - ćwiczenia

### Lista 7

Na wykładzie moduły były wykorzystywane do implementowania stosu jako abstrakcyjnego typu danych. W zadaniach z tej listy trzeba zaimplementować kolejki jako abstrakcyjne typy danych za pomocą modułów.

Moduły są też używane do grupowania użytecznych funkcji (i innych bytów) w jednej przestrzeni nazw. Standardowa biblioteka języka OCaml składa się z modułów. Od pierwszego wykładu używaliśmy modułu List.

Poniżej jako przykład użytecznego modułu bibliotecznego zostanie zdefiniowany niewielki moduł Vector.

W trójwymiarowej przestrzeni euklidesowej *wektor zaczepiony*

(<https://pl.wikipedia.org/wiki/Wektor>) jest reprezentowany przez współrzędne jego punktu końcowego  $\mathbf{a} = (a_1, a_2, a_3)$  lub  $\mathbf{a} = (a_x, a_y, a_z)$ .

W nauczaniu początkowym fizyki wersory (wektory jednostkowe) osi są zwykle oznaczane przez  $\mathbf{i}, \mathbf{j}, \mathbf{k}$ . Wówczas  $\mathbf{a} = a_x \mathbf{i} + a_y \mathbf{j} + a_z \mathbf{k}$ .

Długość wektora, iloczyn skalarny oraz iloczyn wektorowy dwóch wektorów są zdefiniowane odpowiednio:

$$\|\mathbf{a}\| = \sqrt{a_1^2 + a_2^2 + a_3^2}$$

$$\mathbf{a} \cdot \mathbf{b} = a_1 b_1 + a_2 b_2 + a_3 b_3$$

$$\mathbf{a} \times \mathbf{b} = (a_2 b_3 - a_3 b_2) \mathbf{i} + (a_3 b_1 - a_1 b_3) \mathbf{j} + (a_1 b_2 - a_2 b_1) \mathbf{k}$$

Moduł Vector będzie implementował poniższą sygnaturę. Wektor będzie reprezentowany jako rekord:

```
type vect = {i: float; j: float; k: float}
```

Nie będziemy ukrywali tej reprezentacji. Funkcja Vector.toString ma zamieniać wektor na napis postaci  $[a_x, a_y, a_z]$ .

```
module type VECTOR =  
sig  
  type vect = {i: float; j: float; k: float}  
  val create: float*float*float -> vect  
  val scalarProduct: vect*vect -> float  
  val vectorProduct: vect*vect -> vect  
  val length: vect -> float  
  val toString: vect -> string  
end;;
```

Funkcje modułu są zaimplementowane zgodnie z przytoczonymi powyżej definicjami. Dodatkowe komentarze są zbędne, dzięki odpowiedniemu wyborowi identyfikatorów.

```
module Vector: VECTOR =
struct
  type vect = {i: float; j: float; k: float}

  let create(a1, a2, a3) = {i = a1; j = a2; k = a3}

  let length {i = a1; j = a2; k = a3} = sqrt(a1*.a1 +. a2*.a2 +. a3*.a3)

  let scalarProduct({i = a1; j = a2; k = a3}, {i = b1; j = b2; k = b3}) = a1*.b1 +. a2*.b2 +. a3*.b3

  let vectorProduct({i = a1; j = a2; k = a3}, {i = b1; j = b2; k = b3}) =
    {i = a2*.b3 -. a3*.b2; j = a3*.b1 -. a1*.b3; k = a1*.b2 -. a2*.b1}

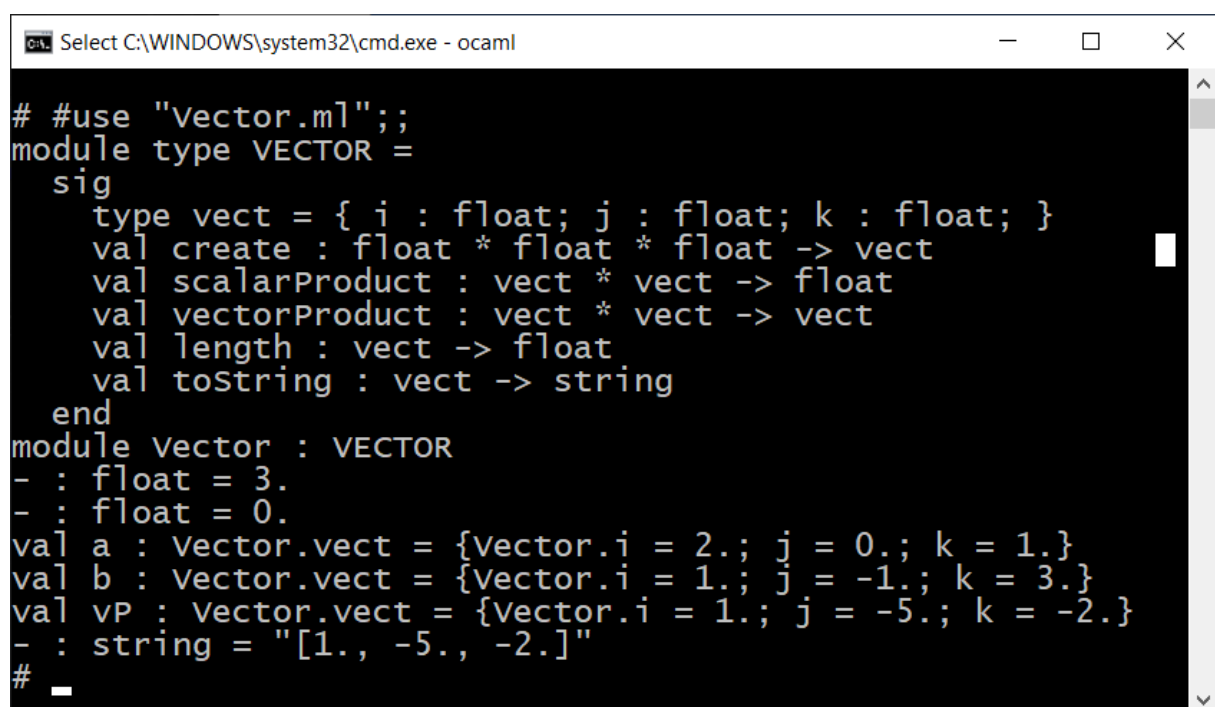
  let toString {i = a1; j = a2; k = a3} =
    "[" ^ string_of_float a1 ^ ", " ^ string_of_float a2 ^ ", " ^ string_of_float a3 ^ "]"
end;;
```

W pliku Vector.ml oprócz definicji sygnatury VECTOR i modułu Vector jest też kilka wywołań funkcji z modułu:

```
Vector.length (Vector.create(1., 2., 2.));;
```

```
Vector.scalarProduct({Vector.i = 1.; j = 3.; k = -5.}, {Vector.i = 4.; j = -3.; k = -1.});;
```

```
let a = Vector.create(2., 0., 1.);;
let b = Vector.create(1., -1., 3.);;
let axb = Vector.vectorProduct(a, b);;
Vector.toString axb;;
```



The screenshot shows a Windows command prompt window titled "Select C:\WINDOWS\system32\cmd.exe - ocaml". The window contains OCaml code and its output. The code defines a module Vector and performs several operations. The output shows the results of these operations, including the length of a vector, the scalar product of two vectors, and the vector product of two vectors, along with their string representations.

```
# #use "vector.ml";;
module type VECTOR =
  sig
    type vect = { i : float; j : float; k : float; }
    val create : float * float * float -> vect
    val scalarProduct : vect * vect -> float
    val vectorProduct : vect * vect -> vect
    val length : vect -> float
    val toString : vect -> string
  end
module Vector : VECTOR
- : float = 3.
- : float = 0.
val a : Vector.vect = {Vector.i = 2.; j = 0.; k = 1.}
val b : Vector.vect = {Vector.i = 1.; j = -1.; k = 3.}
val vP : Vector.vect = {Vector.i = 1.; j = -5.; k = -2.}
- : string = "[1., -5., -2.]"
# _
```

## Paradygmaty programowania - ćwiczenia

### Lista 7

#### Algebraiczna specyfikacja kolejki nieskończonej

##### Sygnatura

```
empty    : -> Queue
enqueue  : Elem * Queue -> Queue
first    : Queue -> Elem
dequeue  : Queue -> Queue
isEmpty  : Queue -> bool
```

##### Aksjomaty

```
For all q:Queue, e1,e2: Elem
isEmpty (enqueue (e1,q))           = false
isEmpty (empty)                     = true
dequeue (enqueue (e1,enqueue (e2,q))) =
    enqueue (e1,dequeue (enqueue (e2,q)))
dequeue (enqueue (e1,empty))        = empty
dequeue (empty)                     = empty
first (enqueue (e1,enqueue (e2,q))) = first (enqueue (e2,q))
first (enqueue (e1,empty))          = e1
first (empty)                       = ERROR
```

1. (OCaml) Dana jest następująca sygnatura dla kolejek czysto funkcyjnych.

```
module type QUEUE_FUN =
sig
  type 'a t
  exception Empty of string
  val empty: unit -> 'a t
  val enqueue: 'a * 'a t -> 'a t
  val dequeue: 'a t -> 'a t
  val first: 'a t -> 'a
  val isEmpty: 'a t -> bool
end;;
```

Napisz moduł, zgodny z powyższą sygnaturą, w którym kolejka jest reprezentowana:

a) przez zwykłą listę;

b) przez parę list.

Reprezentacja z punktu a) jest mało efektywna, ponieważ operacja wstawiania do kolejki (lub usuwania z kolejki) ma złożoność liniową. W lepszej reprezentacji kolejka jest reprezentowana przez parę list.

Para list  $([x_1; x_2; \dots; x_m], [y_1; y_2; \dots; y_n])$  reprezentuje kolejkę  $x_1 x_2 \dots x_m y_n \dots y_2 y_1$ . Pierwsza lista reprezentuje początek kolejki, a druga – koniec kolejki. Elementy w drugiej liście są zapamiętane w odwrotnej kolejności, żeby wstawianie było wykonywane w czasie stałym (na początek listy).

$enqueue(y, q)$  modyfikuje kolejkę następująco:  $(xl, [y_1; y_2; \dots; y_n]) \rightarrow (xl, [y; y_1; y_2; \dots; y_n])$ . Elementy w pierwszej liście są pamiętane we właściwej kolejności, co umożliwia szybkie usuwanie pierwszego elementu.  $dequeue(q)$  modyfikuje kolejkę następująco:  $([x_1; x_2; \dots; x_m], yl) \rightarrow ([x_2; \dots; x_m], yl)$ . Kiedy pierwsza lista zostaje opróżniona, druga lista jest odwracana i wstawiana w miejsce pierwszej:

$([], [y_1; y_2; \dots; y_n]) \rightarrow ([y_n; \dots; y_2; y_1], [])$ . Reprezentacja kolejki jest w postaci normalnej, jeśli nie wygląda tak:  $([], [y_1; y_2; \dots; y_n])$  dla  $n \geq 1$ . **Wszystkie operacje kolejki mają zwracać reprezentację w postaci normalnej**, dzięki czemu pobieranie wartości pierwszego elementu nie spowoduje odwracania listy. Odwracanie drugiej listy po opróżnieniu pierwszej też może się wydawać

kosztowne. Jeśli jednak oszacujemy nie koszt pesymistyczny (oddzielnie dla każdej operacji kolejki), ale koszt zamortyzowany (uśredniony dla całego czasu istnienia kolejki), to okaże się, że koszt operacji wstawiania i usuwania z kolejki jest stały.

W ten sposób reprezentowane są kolejki w językach czysto funkcyjnych.

2. (OCaml) Dana jest następująca sygnatura dla kolejek modyfikowalnych.

```
module type QUEUE_MUT =
sig
  type 'a t
    (* The type of queues containing elements of type ['a]. *)
  exception Empty of string
    (* Raised when [first q] is applied to an empty queue [q]. *)
  exception Full of string
    (* Raised when [enqueue(x,q)] is applied to a full queue [q]. *)
  val empty: int -> 'a t
    (* [empty n] returns a new queue of length [n], initially empty. *)
  val enqueue: 'a * 'a t -> unit
    (* [enqueue (x,q)] adds the element [x] at the end of a queue [q]. *)
  val dequeue: 'a t -> unit
    (* [dequeue q] removes the first element in queue [q] *)
  val first: 'a t -> 'a
    (* [first q] returns the first element in queue [q] without removing
       it from the queue, or raises [Empty] if the queue is empty. *)
  val isEmpty: 'a t -> bool
    (* [isEmpty q] returns [true] if queue [q] is empty,
       otherwise returns [false]. *)
  val isFull: 'a t -> bool
    (* [isFull q] returns [true] if queue [q] is full,
       otherwise returns [false]. *)
end;;
```

Napisz moduł, zgodny z powyższą sygnaturą, w którym kolejka jest reprezentowana przez tablicę cykliczną (ang. circular array).

### *Kolejka reprezentowana przez tablicę cykliczną*

*kolejka pusta*

*kolejka pełna*

