**Semantyka i weryfikacja programów 2022/23.**
**Egzamin 6/02/2023, zadanie 3 (weryfikacja)**

Pracujemy w języku $\textsc{Tiny}_{\mathcal{A}}$ nad typem danych rozszerzonym o rodzaj *Array* i operacje:

$$\begin{aligned}
&\mathit{newarr}\colon \mathit{Array}\\
&\mathit{put}\colon \mathit{Array}\times \mathit{Int}\times \mathit{Int}\to \mathit{Array}\\
&\mathit{get}\colon \mathit{Array}\times \mathit{Int}\to \mathit{Int}\\
&\mathit{swap}\colon \mathit{Array}\times \mathit{Int}\times \mathit{Int}\to \mathit{Array}
\end{aligned}$$

Nośnik rodzaju *Array* to zbiór funkcji (całkowitych) z liczb całkowitych w liczby całkowite,

$$|\mathcal{A}|_{\mathit{Array}} = \mathbf{Int}\to\mathbf{Int},$$

a operacje interpretowane są jako funkcje

$$\begin{aligned}
&\mathit{newarr}_{\mathcal{A}}\colon |\mathcal{A}|_{\mathit{Array}}\\
&\mathit{put}_{\mathcal{A}}\colon |\mathcal{A}|_{\mathit{Array}}\times|\mathcal{A}|_{\mathit{Int}}\times|\mathcal{A}|_{\mathit{Int}}\to|\mathcal{A}|_{\mathit{Array}}\\
&\mathit{get}_{\mathcal{A}}\colon |\mathcal{A}|_{\mathit{Array}}\times|\mathcal{A}|_{\mathit{Int}}\to|\mathcal{A}|_{\mathit{Int}}\\
&\mathit{swap}_{\mathcal{A}}\colon |\mathcal{A}|_{\mathit{Array}}\times|\mathcal{A}|_{\mathit{Int}}\times|\mathcal{A}|_{\mathit{Int}}\to|\mathcal{A}|_{\mathit{Array}}
\end{aligned}$$

zdefiniowane następująco:

$$\begin{aligned}
&\mathit{newarr}_{\mathcal{A}}(i) = 0\\
&\mathit{put}_{\mathcal{A}}(A,i,n) = A[i\mapsto n]\\
&\mathit{get}_{\mathcal{A}}(A,i) = A(i)\\
&\mathit{swap}_{\mathcal{A}}(A,i,j) = A[i\mapsto A(j), j\mapsto A(i)]
\end{aligned}$$

dla wszystkich $i,j,n\in\mathbf{Int}$ i $A\colon \mathbf{Int}\to\mathbf{Int}$. Wyrażenie $\mathit{get}(A,e)$ będziemy jak zwykle zapisywać jako $A[e]$. Ponadto, w asercjach wykorzystujemy „predykat” $A{:}\mathit{Array}$ stwierdzajacy, że zmienna $A$ jest rodzaju *Array* (pozostałe zmienne są rodzaju *Int*). Wyrażenia logiczne języka rozszerzamy też o oczywiste nierówności $e < e'$ (wyrażalne jako $e\le e'\wedge\neg(e=e')$) oraz $e > e'$ (wyrażalne jako $e' < e$). Dla czytelności, w formułach ciągi nierówności będziemy zapisywać w skróconej postaci, np. $j\le i\wedge i<k$ zapisując jako $j\le i<k$.

Dla wygody specyfikacji i dowodzenia poprawności wprowadzamy pomocniczy schemat formuł:

$$\mathit{max}(A,j,k,p) \equiv A{:}\mathit{Array}\wedge\forall i.(j\le i\le k \implies A[i]\le A[p])$$

Należy udowodnić całkowitą poprawność następującego programu względem podanych warunków:

$[A{:}Array \wedge n > 0\ ]$

```
m := 1;
while m < n
  do
  (if A[m] > n*n then
      (j := 0;
       while m+j < n-j
         do (if A[m+j] > A[n-j] then A := swap(A,m+j,n-j) else skip;
             j := j+1
            );
       m := m+j
      )
    else
      (x := A[n];
       j := n; i := n;
       while j > m
          do (j := j-1;
              if A[j] > x then (A := swap(A,i,j); i := i-1) else skip
             );
       if i = n then m := n else m := i+1
      )
  )
```

$[A{:}Array \wedge max(A, 1, n, n)]$

Na następnej stronie podana jest wersja tego programu z miejscem na wpisanie odpowiednich adnotacji, którą można wykorzystać dla przedstawienia rozwiązania (przy braku miejsca można też np. wpisać tam tylko nazwy adnotacji zdefiniowanych osobno).

Wymagane jest:

- podanie niezmienników $\gamma_0$, $\gamma_1$ i $\gamma_2$ wszystkich trzech pętli programu oraz asercji $\alpha_1$ i $\alpha_2$, które „kodują” dowód poprawności (podanie pozostałych asercji jest opcjonalne, ale jeśli zostaną podane, to ewentualne błędy mogą wpłynąć na ocenę rozwiązania), oraz
- podanie adnotacji `[decr ... in ... wrt ... ]` tak, aby (w kontekście podanych niezmienników) wynikała z nich własność stopu pętli.

```
[A:Array ∧ n > 0 ]
m := 1
[                                                                    ]
while [γ0 :                                                          ]
    m < n
  do [decr              in                wrt                 ]
  (if A[m] > n*n then
     ([                                                              ]
      j := 0;
      [                                                              ]
      while [γ1 :                                                    ]
          m+j < n-j
        do [decr              in              wrt                ]
          ( [                                                        ]
            if A[m+j] > A[n-j] then A := swap(A,m+j,n-j) else skip;
            [                                                        ]
            j := j+1
            [                                                        ]
          );
          [α1 :                                                      ]
       m := m+j
       [                                                             ]
     )
    else
      ([                                                             ]
       x := A[n];
       [                                                             ]
       j := n; i := n;
       [                                                             ]
       while [γ2 :                                                   ]
           j > m
         do [decr              in              wrt                ]
         ( [                                                         ]
           j := j-1;
           [                                                         ]
           if A[j] > x then ( A := swap(A,i,j); i := i-1 ) else skip
           [                                                         ]
         );
       [α2 :                                                       ]
       if i = n then m := n else m := i+1
       [                                                           ]
      )
  [                                                                ]
  )
[A:Array ∧ max(A, 1, n, n)]
```

## Możliwe rozwiązanie

```
[A:Array ∧ n > 0 ]
m := 1
while [γ0 : A:Array ∧ 1 ≤ m ∧ ∃k.(m ≤ k ≤ n ∧ max(A, 1, m − 1, k))]
    m < n
  do [decr n − m in Nat wrt >]
  (if A[m] > n*n then
     (j := 0;
      while [γ1 : γ0 ∧ m < n ∧ 0 ≤ 2 ∗ j ≤ n − m + 1 ∧
                  ∀k.(0 ≤ k < j ⟹ A[m + k] ≤ A[n − k])]
          m+j < n-j
        do [decr n − j in Nat wrt >]
          ( if A[m+j] > A[n-j] then A := swap(A,m+j,n-j) else skip;
            j := j+1
          );
       [α1 : γ0 ∧ m < n ∧ (m + j = n − j ∨ m + j = n − j + 1) ∧ j > 0 ∧
                ∀k.(0 ≤ k < j ⟹ A[m + k] ≤ A[n − k])]
      m := m+j
     )
    else
      (x := A[n];
       j := n; i := n;
       while [γ2 : γ0 ∧ m < n ∧ m ≤ j ≤ i ≤ n ∧ (i = n ⟹ x = A[n]) ∧
                   ∀k.(i < k ≤ n ⟹ A[k] > x) ∧ ∀k.(j ≤ k ≤ i ⟹ A[k] ≤ x)]
          j > m
        do [decr j in Nat wrt >]
        ( j := j-1;
          if A[j] > x then ( A := swap(A,i,j); i := i-1 ) else skip
        );
          [α2 : γ0 ∧ m < n ∧ m ≤ i ≤ n ∧ (i = n ⟹ x = A[n]) ∧
                ∀k.(i < k ≤ n ⟹ A[k] > x) ∧ ∀k.(m ≤ k ≤ i ⟹ A[k] ≤ x)]
       if i = n then m := n else m := i+1
      )
  )
[A:Array ∧ max(A, 1, n, n)]
```