# Zadanie na pracownię nr 6

**Uwaga:** Termin oddania rozwiązań tego zadania to wtorek, 10 kwietnia 2018, godz 6:00.

## Podpowiadanie zadeklarowanych zmiennych

W cyklu produkcji programu kod jest nie tylko pisany i czytany przez programistę, a następnie interpretowany lub kompilowany. Poddawany jest on także najprzeróżniejszym analizom w celu wyłapania błędów czy przeprowadzenia optymalizacji. Jest on także analizowany na bieżąco przez edytor/IDE w celu np. refaktoringu, kolorowania składni i podpowiadania programiście nazw dostępnych procedur, modułów, funkcji, metod, klas itp. Ta ostatnia funkcja nazywa się *code completion*. Chociaż w DrRackecie nie jest ona szczególnie rozbudowana, jest często dość użyteczna, bo pozwala odszukać nazwę funkcji bibliotecznej bez potrzeby opuszczania edytora (przywołuje się ją kombinacją klawiszy `Ctrl-/`).

Celem tej pracowni jest napisanie fragmentu bardzo prostego podpowiadacza dla autorów `let`-wyrażeń przedstawionych na wykładzie. Jego zadaniem jest wczytanie takiego wyrażenia i poinformowanie, jakie zmienne związane są zadeklarowane w oznaczonym miejscu.

Formatem wejściowym zadania są wyrażenia rozbudowane o nową konstrukcję: **dziurę**. Dziura reprezentowana jest symbolem `'hole`[1], a predykat definiujący `let`-wyrażenia z dziurą wygląda następująco (pełen kod dostępny jest na SKOS-ie):

```
(define (hole? t)
  (eq? t 'hole))

(define (arith/let/holes? t)
  (or (hole? t)
      (const? t)
      (and (binop? t)
           (arith/let/holes? (binop-left  t))
           (arith/let/holes? (binop-right t)))
      (and (let? t)
           (arith/let/holes? (let-expr t))
```

[1] Uważny czytelnik zauważy pewien konflikt reprezentacji dziur z tym, jak reprezentujemy zmienne. Proszę tym się nie przejmować, zakładając, że nie używa się w wyrażeniach zmiennej o nazwie `hole`.

```
          (arith/let/holes? (let-def-expr (let-def t))))
      (var? t)))

(define (num-of-holes t)
  (cond [(hole? t)  1]
        [(const? t) 0]
        [(binop? t) (+ (num-of-holes (binop-left  t))
                       (num-of-holes (binop-right t)))]
        [(let? t)   (+ (num-of-holes (let-expr t))
                       (num-of-holes (let-def-expr (let-def t))))]
        [(var? t)   0]))

(define (arith/let/hole-expr? t)
  (and (arith/let/holes? t)
       (= (num-of-holes t) 1)))
```

Celem zadania jest zdefiniowanie jednoargumentowej procedury `hole-context`, która bierze jako swój argument wyrażenie z dziurą (czyli takie, które spełnia predykat `arith/let/hole-expr?`) i odpowiada *kontekstem*, czyli listą zmiennych związanych widocznych w miejscu, w którym w wyrażeniu znajduje się dziura. Proszę pamiętać, że zmienna przykryta zmienną o tej samej nazwie nie jest widoczna. W praktyce oznacza to, że na liście wynikowej zmienne nie powinny się powtarzać. Kolejność zmiennych na liście wynikowej nie jest istotna. Kilka przykładów:

```
> (hole-context '(+ 3 hole))
'()
> (hole-context '(let (x 3) (let (y 7) (+ x hole))))
'(y x)
> (hole-context '(let (x 3) (let (y hole) (+ x 3))))
'(x)
> (hole-context '(let (x hole) (let (y 7) (+ x 3))))
'()
> (hole-context '(let (piesek 1)
                  (let (kotek 7)
                  (let (piesek 9)
                  (let (chomik 5)
                  hole)))))
'(chomik piesek kotek)
> (hole-context '(+ (let (x 4) 5) hole))
'()
```

Ważnym jest, by poprawność dostarczonego rozwiązania nie zależała od we wnętrznej reprezentacji wyrażeń. W praktyce oznacza to, że procedura obliczająca kontekst powinna korzystać z predykatów i selektorów dla `let`-wyrażeń, a nie faktu, że są one reprezentowane jako listy z odpowiednimi wartościami w odpowiednich miejscach.

Do rozwiązania należy dołączyć bezargumentową procedurę `test`, która testuje na kilku własnych przykładach procedurę `hole-context` i odpowiada `#t` (jeśli `hole-context` podaje prawidłowy kontekst) lub `#f` (w przeciwnym przypadku). Definiuj procedurę `test` z myślą, że może ona być użyta do przete-

stowania rozwiązań Twoich kolegów i koleżanek.

Rozwiązanie należy przesłać jako plik o nazwie w formacie `nazwisko-imie.rkt`, jak zwykle bez spacji i polskich znaków. Rozwiązanie należy nadesłać w formie uzupełnienia szablonu dostępnego na SKOS-ie.