**Zadanie 2.**

Podaj semantykę denotacyjną dla języka o następującej gramatyce:

$$
\begin{array}{lcl}
\mathit{Num} \ni n & ::= & \ldots \mid -\texttt{1} \mid \texttt{0} \mid \texttt{1} \mid \ldots \\
\mathit{Var} \ni x & ::= & \texttt{x} \mid \texttt{y} \mid \texttt{z} \mid \ldots \mid \ldots \\
\mathit{PVar} \ni p & ::= & \texttt{p} \mid \texttt{q} \mid \ldots \mid \ldots \\
\mathit{Expr} \ni E & ::= & n \mid x \mid E_1 + E_2 \mid E_1 - E_2 \\
\mathit{Instr} \ni I & ::= & x{:=}E \mid I_1;I_2 \mid \texttt{skip} \mid \texttt{if } E = 0 \texttt{ then } I_1 \texttt{ else } I_2 \mid \texttt{begin } d\ I \texttt{ end} \mid \\
 & & \texttt{call } p(x) \mid \texttt{export } p \mid \texttt{exit } p \\
\mathit{Decl} \ni d & ::= & \texttt{var } x = E \mid \texttt{proc } p(x) \texttt{ is } (I) \mid d_1;d_2
\end{array}
$$

Instrukcje postaci $x{:=}E$, $I_1;I_2$, `skip`, `if` $E = 0$ `then` $I_1$ `else` $I_2$ oraz bloki `begin` $d\ I$ `end` wykonuje się standardowo (w rozwiązaniach zadania należy jednak opisać ich semantykę).

Deklaracje zmiennych w blokach są standardowe, z inicjalizacją wartości deklarowanej zmiennej określoną jako wartość podanego w deklaracji wyrażenia. Deklaracje procedur wprowadzają jednoparametrowe procedury rekurencyjne wywoływane instrukcjami postaci `call` $p(x)$. Przyjmujemy statyczne wiązanie globalnych zmiennych i, dla wywołań procedur, identyfikatorów procedur. Parametry są przekazywane przez wartość (z parametrem formalnym wykorzystywanym jako lokalna zmienna w ciele procedury). Ale:

- wykonanie instrukcji `export` $p$ w trakcie bieżącego (ostatniego) wywołania procedury $p$ powoduje ustalenie wartości zmiennej będącej parametrem aktualnym tego wywołania na bieżącą wartość parametru formalnego procedury $p$ (i dalej wykonanie ciała procedury jest kontynuowane), oraz
- wykonanie instrukcji `exit` $p$ w trakcie wywołania procedury $p$ powoduje zakończenie (przerwanie) wykonywania bieżącego (ostatniego) wywołania procedury $p$ i zwiększenie wartości parametru aktualnego tego wywołania o 1.

Znaczenie instrukcji `export` $p$ i `exit` $p$ w momentach, gdy procedura $p$ nie jest wywołana, może być przez semantykę definiowane dowolnie.

W rozwiązaniu można wykorzystać pomocniczą funkcję $\mathit{newloc}\colon \mathsf{Store} \to \mathsf{Loc}$, która dla każdego składu $s \in \mathsf{Store}$ podaje nową („dotychczas niewykorzystywaną”) lokację $\mathit{newloc}(s) \in \mathsf{Loc}$. Dla wyrażeń wystarczy podać „typ” funkcji semantycznej, pomijając klauzule semantyczne, chyba że semantyka wyrażeń istotnie odbiega od podawanej na wykładzie. Jeśli trzeba, można utożsamiać stałe całkowitoliczbowe $n \in \mathit{Num}$ z odpowiadającymi im liczbami całkowitymi $n \in \mathbb{Z}$.

*. . . verte. . .*

**Przykład:**

Rozważmy następującą instrukcję:

```
 0: begin
 1:   proc q(z) is (
 2:     z := z + 3;
 3:     export q;
 4:     export p;
 5:     exit p     );
 6:   proc p(x) is (
 7:     if x = 0 then
 8:       x := x + 7;
 9:       begin var y = x + 2
10:         call p(y);
11:         x := y
12:         export p
13:       end
14:     else
15:       call q(x) )
16:   call p(a)
17: end
```

Instrukcja ta, wykonana w środowisku zawierającym zmienną *globalną* o nazwie $a$ i wartości 0, działać będzie następująco (dla ułatwienia podane są numery wierszy instrukcji):

- `1-5`, `6-15`: zdefiniowane zostaną procedury $q$ i $p$,
- `16`: wywołana zostanie procedura $p$ z parametrem aktualnym $a$,
- `6`: powstanie nowa zmienna – parametr formalny $x$ procedury $p$ – z wartością 0,
- `7`: sprawdzony zostanie warunek instrukcji `if`, który okaże się prawdziwy,
- `8`: wartość parametru formalnego procedury $p$ zostanie zwiększona do wartości 7,
- `9`: zadeklarowana zostanie zmienna lokalna $y$ o początkowej wartości 9,
- `10`: dojdzie do rekurencyjnego wywołania procedury $p$ z parametrem aktualnym $y$ o wartości 9,
- `6`: powstanie nowa zmienna – parametr formalny $x$ procedury $p$ – z wartością 9,
- `7`: sprawdzony zostanie warunek instrukcji `if`, który okaże się fałszywy,
- `15`: dojdzie do wywołania procedury $q$ z parametrem aktualnym $x$ o wartości 9,
- `1`: powstanie nowa zmienna – parametr formalny $z$ procedury $q$ – z wartością 9,
- `2`: parametr formalny procedury $q$ zwiększy wartość do 12,
- `3`: instrukcja `export` $q$ sprawi, że parametr formalny $x$ ostatniego (drugiego) wywołania procedury $p$ (jako parametr aktualny obecnego wywołania procedury $q$) przyjmie wartość bieżącą parametru formalnego procedury $q$, czyli 12,
- `4`: instrukcja `export` $p$ sprawi, że parametr aktualny $y$ ostatniego (drugiego) wywołania procedury $p$ przyjmie wartość bieżącą parametru formalnego procedury $p$, czyli 12,
- `5`: instrukcja `exit` $p$ sprawi, że ostatnie (drugie) wywołanie procedury $p$ z wiersza `10` się zakończy, zwiększając wartość jego parametru aktualnego $y$ o 1 do wartości 13,
- `11`: parametr formalny $x$ pierwszego wywołania procedury $p$ otrzyma wartość 13,
- `12`: dojdzie do wykonania `export` $p$ co sprawi, że zmienna globalna $a$ (jako parametr aktualny obecnego wywołania procedury $p$) przyjmie wartość 13. Następnie zakończy się wywołanie procedury $p$ z wiersza `16` i wykonanie całej przykładowej instrukcji.

W związku z tym, na koniec zmienna globalna $a$ będzie miała wartość 13.

**Rozwiązanie:**

**Semantyka kontynuacyjna:**

UWAGI:

- Dla każdej procedury $p$, instrukcje `export` $p$ wykonywane w bieżącym wywołaniu $p$ potrzebują lokacji parametru aktualnego oraz parametru formalnego tego wywołania, a instrukcje `exit` $p$ kontynuacji dla bieżącego wywołania procedury $p$. Stąd poniższa postać znaczeń instrukcji oraz procedur.
- Proponowane rozwiązanie przyjmuje dla uproszczenia, że błędy (np. niezadeklarowana zmienna i jej użycie) są sygnalizowane przez $\bot$ ("wartość nieokreśloną") w odpowiedniej dziedzinie — oraz że wartości te w klauzulach semantycznych są propagowane. Oczywiście, można też odróżnić takie błędy od nieokreśloności wynikającej z nieskończonych obliczeń, jak to było robione na wykładzie.

Dziedziny semantyczne:

$$
\begin{aligned}
s \in \mathsf{Store} &= \mathsf{Loc} \to \mathbb{Z}_\bot \\
\kappa \in \mathsf{Cont_I} &= \mathsf{Store} \to \mathsf{Store} \\
\kappa_E \in \mathsf{Cont_E} &= \mathbb{Z} \to \mathsf{Cont_I} \\
\kappa_D \in \mathsf{Cont_D} &= \mathsf{VEnv} \to \mathsf{PEnv} \to \mathsf{Cont_I} \\
\rho \in \mathsf{VEnv} &= \mathit{Var} \to \mathsf{Loc}_\bot \\
\xi \in \mathsf{PEnv} &= \mathit{PVar} \to \mathsf{Proc} \\
\gamma \in \mathsf{CP} &= \mathit{PVar} \to (\mathsf{Loc} \times \mathsf{Loc} \times \mathsf{Cont_I}) \\
P \in \mathsf{Proc} &= \mathsf{Loc} \to \mathsf{CP} \to \mathsf{Cont_I} \to \mathsf{Cont_I}
\end{aligned}
$$

Funkcje semantyczne:

$$
\begin{aligned}
&\mathcal{E}\colon \mathit{Expr} \to \mathsf{VEnv} \to \mathsf{Cont_E} \to \mathsf{Cont_I} \\
&\mathcal{I}\colon \mathit{Instr} \to \mathsf{VEnv} \to \mathsf{PEnv} \to \mathsf{CP} \to \mathsf{Cont_I} \to \mathsf{Cont_I} \\
&\mathcal{D}\colon \mathit{Decl} \to \mathsf{VEnv} \to \mathsf{PEnv} \to \mathsf{Cont_D} \to \mathsf{Cont_I}
\end{aligned}
$$

Klauzule semantyczne:

*Standardowe klauzule dla semantyki wyrażeń:*

$$
\begin{aligned}
&\mathcal{E}[\![n]\!]\,\rho\,\kappa_E = \kappa_E\,n \\
&\mathcal{E}[\![x]\!]\,\rho\,\kappa_E\,s = \kappa_E\,n\,s \ \textbf{where}\ n = s\,l, l = \rho\,x \\
&\mathcal{E}[\![E_1 + E_2]\!]\,\rho\,\kappa_E = \mathcal{E}[\![E_1]\!]\,\rho\,\lambda n_1.\mathcal{E}[\![E_2]\!]\,\rho\,\lambda n_2.\kappa_E(n_1 + n_2) \\
&\mathcal{E}[\![E_1 - E_2]\!]\,\rho\,\kappa_E = \mathcal{E}[\![E_1]\!]\,\rho\,\lambda n_1.\mathcal{E}[\![E_2]\!]\,\rho\,\lambda n_2.\kappa_E(n_1 - n_2)
\end{aligned}
$$

*Klauzule semantyczne dla standardowych instrukcji:*

$$
\begin{aligned}
&\mathcal{I}[\![x{:=}E]\!]\,\rho\,\xi\,\gamma\,\kappa\,s = \mathcal{E}[\![E]\!]\,\rho\,\lambda n.\kappa\,s[l{\mapsto}n] \ \textbf{where}\ l = \rho\,x \\
&\mathcal{I}[\![I_1;I_2]\!]\,\rho\,\xi\,\gamma\,\kappa = \mathcal{I}[\![I_1]\!]\,\rho\,\xi\,\gamma\,(\mathcal{I}[\![I_2]\!]\,\rho\,\xi\,\gamma\,\kappa) \\
&\mathcal{I}[\![\texttt{skip}]\!]\,\rho\,\xi\,\gamma\,\kappa = \kappa \\
&\mathcal{I}[\![\texttt{if}\ E = 0\ \texttt{then}\ I_1\ \texttt{else}\ I_2]\!]\,\rho\,\xi\,\gamma\,\kappa = \\
&\qquad \mathcal{E}[\![E]\!]\,\rho\,\lambda n.\textbf{if}\ n = 0\ \textbf{then}\ \mathcal{I}[\![I_1]\!]\,\rho\,\xi\,\gamma\,\kappa\ \textbf{else}\ \mathcal{I}[\![I_2]\!]\,\rho\,\xi\,\gamma\,\kappa \\
&\mathcal{I}[\![\texttt{begin}\ d\ I\ \texttt{end}]\!]\,\rho\,\xi\,\gamma\,\kappa = \mathcal{D}[\![d]\!]\,\rho\,\xi\,\lambda\rho'.\lambda\xi'.\mathcal{I}[\![I]\!]\,\rho'\,\xi'\,\gamma\,\kappa \\
&\mathcal{I}[\![\texttt{call}\ p(x)]\!]\,\rho\,\xi\,\gamma\,\kappa = P\,l_a\,\gamma\,\kappa \ \textbf{where}\ P = \xi\,p, l_a = \rho\,x
\end{aligned}
$$

*Klauzule semantyczne dla nowych instrukcji w ciele procedury:*

$$
\begin{aligned}
&\mathcal{I}[\![\texttt{export}\ p]\!]\,\rho\,\xi\,\gamma\,\kappa\,s = \kappa\,s[l_a{\mapsto}(s\,l_f)] \ \textbf{where}\ \langle l_a, l_f, \_\rangle = \gamma\,p \\
&\mathcal{I}[\![\texttt{exit}\ p]\!]\,\rho\,\xi\,\gamma\,\kappa = \kappa_p \ \textbf{where}\ \langle \_, \_, \kappa_p\rangle = \gamma\,p
\end{aligned}
$$

*Klauzule semantyczne dla deklaracji:*

$\mathcal{D}[\![d_1;d_2]\!]\,\rho\,\xi\,\kappa_D = \mathcal{D}[\![d_1]\!]\,\rho\,\xi\,(\lambda\rho'.\lambda\xi'.\mathcal{D}[\![d_2]\!]\,\rho'\,\xi'\,\kappa_D)$

$\mathcal{D}[\![\texttt{var}\ x = E]\!]\,\rho\,\xi\,\kappa_D\,s = \mathcal{E}[\![E]\!]\,\rho\,\lambda n.\kappa_D\,\rho[x{\mapsto}l]\,\xi\,s[l{\mapsto}n]$

**where** $l = newloc(s)$

$\mathcal{D}[\![\texttt{proc}\ p(x)\ \texttt{is}\ (I)]\!]\,\rho\,\xi\,\kappa_D = \kappa_D\,\rho\,\xi[p{\mapsto}P]$

**where** $P\,l_a\,\gamma\,\kappa\,s = \mathcal{I}[\![I]\!]\,\rho[x{\mapsto}l_f]\,\xi[p{\mapsto}P]\,\gamma[p{\mapsto}\langle l_a, l_f, \kappa'\rangle]\,\kappa\,s[l_f{\mapsto}(s\,l_a)]$

**where** $l_f = newloc(s), \kappa' = \lambda s'.\kappa\,s'[l_a{\mapsto}((s'\,l_a)+1)]$