Programski jezik PINS

pri predmetu Prevajalniki in navidezni stroji v študijskem letu 2022/23

1 Leksikalna pravila

- *Ključne besede*:
  `arr else for fun if then typ var where while`
- *Imena atomarnih podatkovnih tipov*:
  `logical integer string`
- *Konstante atomarnih podatkovnih tipov:*

`logical`	`true` `false`
`integer`	Poljubno predznačeno zaporedje števk.
`string`	Poljubno (lahko prazno) zaporedje znakov z ASCII kodami med vključno 32_{10} in 126_{10}, ki je obdano z enojnima navednicama (“`'`”, ASCII koda 39_{10}); izjema je sam znak “`'`”, ki je podvojen.

- *Imena*:
  Poljubno zaporedje črk, števk in podčrtajev, ki se ne začne s števko in ni ne ključna beseda ne ime ali konstanta atomarnega podatkovnega tipa.
- *Ostali simboli*:
  `+ - * / % & | ! == != < > <= >= ( ) [ ] { } : ; . , =`
- *Komentarji*:
  Komentar je poljubno besedilo, ki se začne z znakom “`#`” (ASCII koda 35_{10}) in se razteza do konca vrstice.
- *Belo besedilo*:
  Presledek (ASCII koda 32_{10}), tabulator (ASCII koda 9_{10}) in znaka za konec vrstice (ASCII kodi 10_{10} in 13_{10}) predstavljajo belo besedilo.

2 Sintaksna pravila

source $\longrightarrow$ *definitions*

definitions $\longrightarrow$ *definition*
definitions $\longrightarrow$ *definitions* `;` *definition*

definition $\longrightarrow$ *type_definition*
definition $\longrightarrow$ *function_definition*
definition $\longrightarrow$ *variable_definition*

type_definition $\longrightarrow$ `typ` identifier `:` *type*

type $\longrightarrow$ identifier

type $\longrightarrow$ `logical`
type $\longrightarrow$ `integer`
type $\longrightarrow$ `string`
type $\longrightarrow$ `arr` `[` int_const `]` *type*

function_definition $\longrightarrow$ `fun` identifier `(` *parameters* `)` `:` *type* `=` *expression*

parameters $\longrightarrow$ *parameter*
parameters $\longrightarrow$ *parameters* `,` *parameter*

parameter $\longrightarrow$ identifier `:` *type*

expression $\longrightarrow$ *logical_ior_expression*
expression $\longrightarrow$ *logical_ior_expression* `{` `WHERE` *definitions* `}`

logical_ior_expression $\longrightarrow$ *logical_ior_expression* `|` *logical_and_expression*
logical_ior_expression $\longrightarrow$ *logical_and_expression*

logical_and_expression $\longrightarrow$ *logical_and_expression* `&` *compare_expression*
logical_and_expression $\longrightarrow$ *compare_expression*

compare_expression $\longrightarrow$ *additive_expression* `==` *additive_expression*
compare_expression $\longrightarrow$ *additive_expression* `!=` *additive_expression*
compare_expression $\longrightarrow$ *additive_expression* `<=` *additive_expression*
compare_expression $\longrightarrow$ *additive_expression* `>=` *additive_expression*
compare_expression $\longrightarrow$ *additive_expression* `<` *additive_expression*
compare_expression $\longrightarrow$ *additive_expression* `<` *additive_expression*
compare_expression $\longrightarrow$ *additive_expression*

additive_expression $\longrightarrow$ *additive_expression* `+` *multiplicative_expression*
additive_expression $\longrightarrow$ *additive_expression* `-` *multiplicative_expression*
additive_expression $\longrightarrow$ *multiplicative_expression*

multiplicative_expression $\longrightarrow$ *multiplicative_expression* `*` *prefix_expression*
multiplicative_expression $\longrightarrow$ *multiplicative_expression* `/` *prefix_expression*
multiplicative_expression $\longrightarrow$ *multiplicative_expression* `%` *prefix_expression*
multiplicative_expression $\longrightarrow$ *prefix_expression*

prefix_expression $\longrightarrow$ `+` *prefix_expression*
prefix_expression $\longrightarrow$ `-` *prefix_expression*
prefix_expression $\longrightarrow$ `!` *prefix_expression*
prefix_expression $\longrightarrow$ *postfix_expression*

postfix_expression $\longrightarrow$ *postfix_expression* `[` *expression* `]`
postfix_expression $\longrightarrow$ *atom_expression*

atom_expression $\longrightarrow$ log_constant
atom_expression $\longrightarrow$ int_constant
atom_expression $\longrightarrow$ str_constant
atom_expression $\longrightarrow$ identifier
atom_expression $\longrightarrow$ identifier `(` *expressions* `)`
atom_expression $\longrightarrow$ `{` *expression* `=` *expression* `}`
atom_expression $\longrightarrow$ `{` `if` *expression* `then` *expression* `}`
atom_expression $\longrightarrow$ `{` `if` *expression* `then` *expression* `else` *expression* `}`
atom_expression $\longrightarrow$ `{` `while` *expression* `:` *expression* `}`
atom_expression $\longrightarrow$ `{` `for` identifier `=` *expression* `,` *expression* `,` *expression* `:` *expression* `}`
atom_expression $\longrightarrow$ `(` *expressions* `)`

expressions $\longrightarrow$ *expression*
expressions $\longrightarrow$ *expressions* `,` *expression*

variable_definition $\longrightarrow$ `var` identifer `:` *type*

3 Semantična pravila

Območja vidnosti

- Ime je vidno v celotnem območju vidnosti (od začetka do konca ne glede na mesto definicije).
- Izraz oblike *expression* `{ WHERE` *definitions* `}` ustvari novo vgnezdeno območje vidnosti: izraz in vse definicije so znotraj novega vgnezdenega območja vidnosti.
- Definicija funkcije ustvari novo vgnezdeno območje vidnosti, ki se začne za imenom funkcije in se razteza do konca definicije funkcije.

Tipiziranost

Podatkovni tipi:

- `logical`, `integer` in `string` opisujejo tipe LOGICAL, INTEGER in STRING, zaporedoma.
- Če je vrednost konstante int_const enaka n in *type* opisuje tip τ, tedaj

  `arr [` int_const `]` *type*

  opisuje tip ARR(n, τ).

Deklaracije:

- Deklaracija tipa

  `typ` identifier `:` *type* `,`

  pri kateri *type* opisuje tip τ, določa, da identifier opisuje tip τ.
- Deklaracija funckije

  `fun` identifier
  `(` identifier$_1$ `:` *type*$_1$ `,` identifier$_2$ `:` *type*$_2$ `,` ... `,` identifier$_n$ `:` *type*$_n$ `)`
  `:` *type* `=` *expression* `,`

  pri kateri (a) *type*$_i$ opisuje tip τ_i za $i \in \{1, 2, \ldots, n\}$, (b) *type* opisuje tip τ in (c) je *expression* tipa τ, določa, da je funkcija identifier tipa $\tau_1 \times \tau_2 \times \ldots \times \tau_n \to \tau$.
- Deklaracija spremenljivke

  `var` identifier `:` *type* `,`

  pri kateri *type* opisuje tip τ, določa, da je spremenljivka identifier tipa τ.
- Deklaracija parametra ali komponente

  identifier `:` *type* `,`

  pri kateri *type* opisuje tip τ, določa, da je parameter ali komponenta identifier tipa τ.

Izrazi:

- log_const, int_const in str_const so tipa LOGICAL, INTEGER in STRING, zaporedoma.
- Če je *expression* tipa LOGICAL, je `!` *expression* tipa LOGICAL.
- Če je *expression* tipa INTEGER, sta `+` *expression* in `-` *expression* tipa INTEGER.
- Če sta *expression*$_1$ in *expression*$_2$ tipa LOGICAL, potem je

  expression$_1$ *op* *expression*$_2$ pri *op* $\in$ {`&`, `|`}

  tipa LOGICAL.

- Če sta $expression_1$ in $expression_2$ tipa INTEGER, je

  $$expression_1 \ op \ expression_2 \quad \text{pri } op \in \{\texttt{+}, \texttt{-}, \texttt{*}, \texttt{/}, \texttt{\%}\}$$

  tipa INTEGER.
- Če sta $expression_1$ in $expression_2$ tipa $\tau \in \{\text{LOGICAL}, \text{INTEGER}\}$, je

  $$expression_1 \ op \ expression_2 \quad \text{pri } op \in \{\texttt{==}, \texttt{!=}, \texttt{<=}, \texttt{>=}, \texttt{<}, \texttt{>}\}$$

  tipa LOGICAL.
- Če je $expression_1$ tipa ARR(n, τ) in je $expression_2$ tipa INTEGER, je

  $$expression_1 \ \texttt{[} \ expression_2 \ \texttt{]}$$

  tipa τ.
- Če je identifier tipa $\tau_1 \times \tau_2 \times \ldots \times \tau_n \to \tau$ in so $expression_i$ tipa τ_i za $i \in \{1, 2, \ldots, n\}$, je izraz

  $$\text{identifier} \ \texttt{(} \ expression_1 \ \texttt{,} \ expression_2 \ \texttt{,} \ \ldots \ \texttt{,} \ expression_n \ \texttt{)}$$

  tipa τ.
- Če je *expression* tipa τ, je izraz oblike

  $$expression \ \texttt{\{} \ \texttt{where} \ definitions \ \texttt{\}}$$

  tipa τ.
- Če sta $expression_1$ in $expression_2$ tipa $\tau \in \{\text{LOGICAL}, \text{INTEGER}, \text{STRING}\}$, je

  $$\texttt{\{} \ expression_1 \ \texttt{=} \ expression_2 \ \texttt{\}}$$

  tipa τ.
- Če je *expression* tipa LOGICAL, so

  $$\begin{array}{l} \texttt{\{ while } expression \texttt{ : } expression' \texttt{ \}} \quad , \\ \texttt{\{ if } expression \texttt{ then } expression' \texttt{ \}} \quad \text{in} \\ \texttt{\{ if } expression \texttt{ then } expression' \texttt{ else } expression'' \texttt{ \}} \end{array}$$

  tipa VOID.
- Če so identifier, $expression_1$, $expression_2$ in $expression_3$ tipa INTEGER, je

  $$\texttt{\{ for } \text{identifier} \texttt{ = } expression_1 \texttt{ , } expression_2 \texttt{ , } expression_3 \texttt{ : } expression' \texttt{ \}}$$

  tipa VOID.
- Če so $expression_i$ tipa τ_i za $i \in \{1, 2, \ldots, n\}$, je

  $$\texttt{(} \ expression_1 \ \texttt{,} \ expression_2 \ \texttt{,} \ \ldots \ \texttt{,} \ expression_n \ \texttt{)}$$

  tipa τ_n.