

Programski jezik PINS'22

1 Leksikalna zgradba

Prevajalniki v programskem jeziku PINS'22 so napisani v abecedi ASCII in so sestavljeni iz naslednjih leksikalnih elementov:

- *Konstante:*
 - konstanta tipa void: `none`
 - konstante tipa int:
 - Neprazen končen niz števk (0...9), pred katerim lahko (ni pa nujno) stoji predznak (+ ali -).
 - konstante tipa char:
 - ASCII znak s kodo iz intervala {32...126}, ki je spredaj in zadaj obdan z enojnim zgornjim narekovajem ('); enojni zgornji narekovaj in obratna poševnica (\) morata biti uvedena z dodatno obratno poševnico.
 - konstanta kazalčnih tipov: `nil`
- *Simboli:*
 - () { } [] , : ; & | ! == != < > <= >= * / % + - ^ =
- *Ključne besede:*
 - `char del do else end fun if int new then typ var void where while`
- *Imena:*
 - Neprazen niz črk (A...Z in a...z), števk (0...9) in podčrtajev (_), ki (a) se začne s črko ali podčrtajem (b) in ni ključna beseda ali konstanta.
- *Komentarji:*
 - Niz znakov, ki se začne z `#{` in konča z `}`. Ti komentarji se lahko raztegajo čez več vrstic in so lahko gnezdeni.
- *Belo besedilo:*
 - Presledek ter znaki HT, LF in CR. Znak LF določa konec vrstice. Znak HT je širok 8 znakov.

Leksikalni elementi morajo biti razpoznavni od leve proti desni po pravilu najdaljšega ujemanja.

2 Sintaksna zgradba

Konkretna zgradba programskega jezika PINS'20 je definirana z naslednjo neodvisno gramatiko:

(program)	$prg \rightarrow decl \{ decl \}$
(type declaration)	$decl \rightarrow \text{typ identifier} = type ;$
(variable declaration)	$decl \rightarrow \text{var identifier} : type ;$
(function declaration)	$decl \rightarrow \text{fun identifier} ([\text{identifier} : type \{ , \text{identifier} : type \}]) : type = expr ;$
(atomic type)	$type \rightarrow \text{void} \text{char} \text{int}$
(named type)	$type \rightarrow \text{identifier}$
(array type)	$type \rightarrow [expr] type$
(pointer type)	$type \rightarrow \wedge type$
(enclosed type)	$type \rightarrow (type)$
(constant expression)	$expr \rightarrow \text{const}$
(variable access)	$expr \rightarrow \text{identifier}$
(function call)	$expr \rightarrow \text{identifier} ([expr \{ , expr \}])$
(allocation expression)	$expr \rightarrow (\text{new} \text{del}) expr$

(compound expression)	$expr \rightarrow \{ stmt \{ stmt \} \}$
(enclosed expression)	$expr \rightarrow (expr)$
(typecast expression)	$expr \rightarrow (expr : type)$
(where expression)	$expr \rightarrow (expr \textbf{ where } decl \{ decl \})$
(infix expression)	$expr \rightarrow expr (\& == != < > <= >= * / \% + -) expr$
(prefix expression)	$expr \rightarrow (! + - ^) expr$
(postfix expression)	$expr \rightarrow expr ([expr] ^)$
(expression statement)	$stmt \rightarrow expr ;$
(assignment statement)	$stmt \rightarrow expr = expr ;$
(conditional statement)	$stmt \rightarrow \textbf{ if } expr \textbf{ then } stmt \{ stmt \} \textbf{ end } ;$
(conditional statement)	$stmt \rightarrow \textbf{ if } expr \textbf{ then } stmt \{ stmt \} \textbf{ else } stmt \{ stmt \} \textbf{ end } ;$
(loop statement)	$stmt \rightarrow \textbf{ while } expr \textbf{ do } stmt \{ stmt \} \textbf{ end } ;$

pri čemer je *prg* začetni simbol gramatike in *const* označuje poljubno konstanto.

Primerjalni operatorji niso asociativni, vsi ostali binarni operatorji so levo asociativni.

Prioriteto operatorjev določa naslednja tabela:

<i>postfix operators</i>	<code>[] ^</code>	NAJVIŠJA PRIORITETA
<i>prefix operators</i>	<code>! + - ^</code>	
<i>multiplicative operators</i>	<code>* / %</code>	
<i>additive operators</i>	<code>+ -</code>	
<i>relational operators</i>	<code>== != < > <= >=</code>	
<i>conjunctive operator</i>	<code>&</code>	
<i>disjunctive operator</i>	<code> </code>	NAJNIŽJA PRIORITETA

V zapisu gramatike zgoraj zaviti oklepaji, ki so zapisani kot `{ }`, oklepajo stavčno obliko, ki se lahko ponovi nič ali večkrat, oglati oklepaji, ki so zapisani kot `[]`, pa oklepajo stavčno obliko, ki je lahko izpuščena. Zaviti oklepaji, ki so zapisani kot `{ }`, in oglati oklepaji, ki so zapisani kot `[]`, označujejo simbole, ki so del programa.

3 Semantika

3.1 Območja dosega

Novo območje dosega se ustvari na dva načina:

1. Izraz

`(expr where decls)`

ustvari novo vgnezdено območje dosega, ki se razteza od `(do)`.

2. Deklaracija funkcije oblike

`fun identifier ([identifier : type { , identifier : type }]) : type = expr ;`

ustvari novo vgnezdено območje dosega. Ime funkcije, tipi parametrov in tip rezultata funkcije pripadajo zunanjemu območju dosega, imena parametrov in izraz, ki predstavlja izračun rezultata funkcije, pa pripadajo vgnezdenemu območju dosega, ki je ustvarjeno z deklaracijo funkcije.

Vsa imena, ki so deklarirana v danem območju dosega, so dosegljiva v celotnem območju dosega (razen, če so prekrita z deklaracijami v območjih dosega, ki so vgnezdena v to območje dosega). Vsako ime je lahko v vsakem območju dosega deklarirano največ enkrat.

3.2 Tipiziranost

Programski jezik PiNS'21 vsebuje podatkovne tipe

void, **char**, **int**, **arr**($n \times \tau$) in **ptr**(τ),

pri čemer je τ poljuben podatkovni tip.

Dva tipa sta enaka, če imata isto strukturo (strukturna enakost tipov).

3.2.1 Opis tipov

1. Ključne besede **void**, **char** in **int** opisujejo tipe **void**, **char** in **int**, zaporedoma.
2. Naj bo *int-const* konstanta tipa **int** z vrednostjo $n > 0$ in naj izraz *type* opisuje tip $\tau \neq \mathbf{void}$.
Tedad izraz [*int-const*] *type* opisuje tip **arr**($n \times \tau$).
Primer: [10] **int** opisuje tip **arr**($10 \times \mathbf{int}$).
3. Naj izraz *type* opisuje tip τ .
Tedad izraz \sim *type* opisuje tip **ptr**(τ).
Primer: **ptr char** opisuje tip **ptr(char)**.
4. Naj izraz *type* opisuje tip τ .
Tedad izraz (*type*) opisuje tip τ .
Primer: (([10] (**ptr int**))) opisuje tip **arr**($10 \times \mathbf{ptr(int)}$).

3.2.2 Opis vrednosti

1. Izraz *none* je tipa **void**. Izraz *nil* je tipa **ptr(void)**.
2. Konstante *char-const* in *int-const* so tipa **char** in **int**, zaporedoma.
3. Operand in rezultat unarnih operatorjev + in - sta oba tipa **int**.
4. Operanda in rezultat binarnih operatorjev +, -, *, / in % so vsi tipa **int**.
5. Oba operanda binarnih operatorjev == in != sta oba istega tipa **char**, **int** ali **ptr**(τ).
Rezultat binarnih operatorjev == in != je tipa **int**.
6. Oba operanda binarnih operatorjev <, >, <= in >= sta oba istega tipa **char**, **int** ali **ptr**(τ).
Rezultat binarnih operatorjev <, >, <= in >= je tipa **int**.
7. Naj bo izraz *expr* tipa τ . Tedaj je izraz \sim *expr* tipa **ptr**(τ).
Naj bo izraz *expr* tipa **ptr**(τ). Tedaj je izraz *expr* \sim tipa τ .
8. Naj bo izraz *expr* tipa **int**. Tedaj je izraz **new** *expr* tipa **ptr(void)**.
Naj bo izraz *expr* tipa **ptr**(τ). Tedaj je izraz **del** *expr* tipa **void**.
9. Naj bosta izraza *expr*₁ in *expr*₂ tipa **arr**($n \times \tau$) in **int**. Tedaj je izraz tipa *expr*₁[*expr*₂] tipa τ .
10. Naj bo *identifier* ime spremenljivke tipa τ . Tedaj je izraz *identifier* tipa τ .
11. Naj bo *identifier* ime funkcije, katere argumenti so tipov $\tau_1, \tau_2, \dots, \tau_n$ in ki vrača rezultat tipa τ ; naj bodo izrazi *expr*₁, *expr*₂, ..., *expr*_n tipov $\tau_1, \tau_2, \dots, \tau_n$, zaporedoma. Tedaj je izraz *identifier*(*expr*₁, *expr*₂, ..., *expr*_n) tipa τ .
12. Naj bo izraz *expr* tipa τ_1 in naj izraz *type* opisuje tip τ_2 . Če sta tipa τ_1 in τ_2 (neodvisno drug od drugega) **char**, **int** ali **ptr**(τ), tedaj je izraz (*expr*:*type*) tipa τ_2 .
13. Naj bo izraz *expr* tipa τ . Tedaj je izraz (*expr* **where** *decls*) tipa τ .
14. Naj bo izraz *expr* tipa τ . Tedaj je izraz (*expr*) tipa τ .
15. Naj bodo stavki *stmt*₁, *stmt*₂, ..., *stmt*_n tipov $\tau_1, \tau_2, \dots, \tau_n$, zaporedoma. Tedaj je izraz { *stmt*₁ *stmt*₂ ... *stmt*_n } tipa τ_n .

3.2.3 Stavki

1. Naj bo izraz $expr$ tipa τ . Stavček $expr$; je tipa τ .
2. Naj bosta izraza $expr_1$ in $expr_2$ istega tipa, ki je lahko **char**, **int** ali **ptr**(τ).
Tedaj je stavček $expr_1 = expr_2$; tipa **void**.
3. Naj bo izraz $expr$ tipa **int** in naj bo tip stavčkov $stmts_1$ in $stmts_2$ tip **void**.
Tedaj sta stavka **if** $expr$ **then** $stmts_1$ **end**; in **if** $expr$ **then** $stmts_1$ **else** $stmts_2$ **end**; tipa **void**.
4. Naj bo izraz $expr$ tipa **int** in naj bo tip stavčkov $stmts$ tip **void**. Tedaj je stavček **while** $expr$ **do** $stmts$ **end**, tipa **void**.

3.2.4 Deklaracije

1. Tip spremenljivke ali parametra je določen s tipom, ki je naveden v deklaraciji spremenljivke ali parametra. Tip parametra je lahko zgolj **char**, **int** ali **ptr**(τ).
2. Tip rezultata funkcije je določen s tipom, ki je naveden v deklaraciji funkcije. Tip rezultata mora ustrezati tipu izraza v deklaraciji, lahko pa je zgolj **void**, **char**, **int** ali **ptr**(τ).