Programski jezik PINS'21

1 Leksikalna zgradba

Prevajalniki v programskem jeziku PINS'21 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 $\{32 \ldots 126\}$, ki je spredaj in zadaj obdan z enojnim zgornjim narekovajem (`'`); enojni zgornji narekovaj in obratna poševnica (\) mora biti uveden 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 znakom `#` in se konča na koncu vrstice.
- *Belo besedilo*:
  Presledek ter znaki HT, LF in CR. Znak LF določa konec vrstice. Znak HT je širok 8 znakov.

Leksikalni elementi morajo biti razpoznani 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*	$\longrightarrow$	*decl* { *decl* }
(type declaration)	*decl*	$\longrightarrow$	`typ` identifier `=` *type* `;`
(variable declaration)	*decl*	$\longrightarrow$	`var` identifier `:` *type* `;`
(function declaration)	*decl*	$\longrightarrow$	`fun` identifier `(` [identifier `:` *type* { `,` identifier `:` *type* }] `)` `:` *type* `=` *expr* `;`
(atomic type)	*type*	$\longrightarrow$	`void` | `char` | `int`
(named type)	*type*	$\longrightarrow$	identifier
(array type)	*type*	$\longrightarrow$	`[` *expr* `]` *type*
(pointer type)	*type*	$\longrightarrow$	`^` *type*
(enclosed type)	*type*	$\longrightarrow$	`(` *type* `)`
(constant expression)	*expr*	$\longrightarrow$	const
(variable access)	*expr*	$\longrightarrow$	identifier
(function call)	*expr*	$\longrightarrow$	identifier `(` [*expr* { `,` *expr* }] `)`
(allocation expression)	*expr*	$\longrightarrow$	(`new` | `del`) *expr*

(compound expression) *expr* ⟶ `{` *stmt* { *stmt* } `}`
(enclosed expression) *expr* ⟶ `(` *expr* `)`
(typecast expression) *expr* ⟶ `(` *expr* `:` *type* `)`
(where expression) *expr* ⟶ `(` *expr* `where` *decl* { *decl* } `)`
(infix expression) *expr* ⟶ *expr* (`&` | `|` | `==` | `!=` | `<` | `>` | `<=` | `>=` | `*` | `/` | `%` | `+` | `-`) *expr*
(prefix expression) *expr* ⟶ (`!` | `+` | `-` | `^`) *expr*
(postfix expression) *expr* ⟶ *expr* (`[` *expr* `]` | `^`)

(expression statement) *stmt* ⟶ *expr* `;`
(assignment statement) *stmt* ⟶ *expr* `=` *expr* `;`
(conditional statement) *stmt* ⟶ `if` *expr* `then` *stmt* { *stmt* } `end` `;`
(conditional statement) *stmt* ⟶ `if` *expr* `then` *stmt* { *stmt* } `else` *stmt* { *stmt* } `end` `;`
(loop statement) *stmt* ⟶ `while` *expr* `do` *stmt* { *stmt* } `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:

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

Novo območje vidnosti se ustvari na dva načina:

1. Izraz

   `(` *expr* `where` *decls* `)`

   ustvari novo vgnezdeno območje vidnosti, ki se razteza od `(` do `)`.

2. Deklaracija funkcije oblike

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

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

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

3.2 Tipiziranost

Programski jezik PiNS'21 vsebuje podatkovne tipe

$$\textbf{void}, \textbf{char}, \textbf{int}, \textbf{arr}(n \times \tau) \text{ in } \textbf{ptr}(\tau),$$

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 $\tau \neq$ **void**.
   Tedaj izraz `[`*int-const*`]`*type* opisuje tip **arr**$(n \times \tau)$.
   Primer: `[10]int` opisuje tip **arr**$(10 \times$ **int**$)$.
3. Naj izraz *type* opisuje tip τ.
   Tedaj izraz `^`*type* opisuje tip **ptr**(τ).
   Primer: `ptr char` opisuje tip **ptr**(**char**).
4. Naj izraz *type* opisuje tip τ.
   Tedaj izraz `(`*type*`)` opisuje tip τ.
   Primer: `(([10](ptr int)))` opisuje tip **arr**$(10 \times$ **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 `^`*expr* tipa **ptr**(τ).
   Naj bo izraz *expr* tipa **ptr**(τ). Tedaj je izraz *expr*`^` 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 $\tau_1, \tau_2, \ldots, \tau_n$ in ki vrača rezultat tipa τ; naj bodo izrazi $expr_1, expr_2, \ldots, expr_n$ tipov $\tau_1, \tau_2, \ldots, \tau_n$, 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 $stmt_1, stmt_2, \ldots, stmt_n$ tipov $\tau_1, \tau_2, \ldots, \tau_n$, zaporedoma. Tedaj je izraz `{` $stmt_1$ $stmt_2$ ... $stmt_n$ `}` tipa τ_n.

3.2.3 Stavki

1. Naj bo izraz *expr* tipa τ. Stavek *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 stavek $expr_1$ `=` $expr_2$`;` tipa **void**.
3. Naj bo izraz *expr* tipa **int** in naj bo tip stavkov $stmts_1$ 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 stavkov *stmts* tip **void**. Tedaj je stavek `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).

3.3 Naslovljivost izrazov

1. Ime spremenljivke ali parametra določa naslov.
2. Izraz *expr*`^` določa naslov.
3. Izraz $expr_1$`[`$expr_2$`]` določa naslov.

3.4 Operacijska semantika

Izrazi:

- Vsi izrazi se računajo s 64-bitnimi predznačenimi števili.
- (function call)
  1. Od leve proti desni se izračunajo vrednosti argumentov.
  2. Sledi klic funkcije z izračunanimi vrednostmi argumentov.
- (allocation expression)
  1. Izračuna se vrednost izraza *expr*.
  2. `new`: klic funkcije `new` z vrednostjo izraza *expr*.
  3. `del`: klic funkcije `del` z vrednostjo izraza *expr*.
- (compound expression)
  1. Stavki se izvedejo en za drugim.
  2. Rezultat zadnjega stavka predstavlja rezultat celega izraza.
- (typecast expression)
  1. Izračuna se vrednost izraza.
  2. Binarna slika se ohrani kot vrednost v novem tipu.
- (prefix expression: `^`) Rezultat je naslov izraza *expr*.
- (postfix expression: `^`) Rezultat je vrednost na naslovu, ki ga določa izraz *expr*.

Stavki:

- (expression statement) Izračuna se vrednost izraza *expr*.
- (assignment statement)
  1. Izračuna se naslov levega izraza *expr*.
  2. Izračuna se vrednost desnega izraza *expr*.
  3. Vrednost desnega izraza *expr* se shrani na naslov levega izraza *expr*.
- (conditional statement)
  1. Izračuna se vrednost izraza *expr*.
  2. Če je vrednost izraza *expr* različna od 0, se (en za drugim) izvedejo stavki v `then` veji.
  3. Če je vrednost izraza *expr* enaka 0, se (en za drugim) izvedejo stavki v `else` veji (če obstaja).
- (loop statement)
  1. Izračuna se vrednost izraza *expr*.
  2. Če je vrednost izraza *expr* različna od 0, se (en za drugim) izvedejo stavki in celotno izvajanje stavka `while`...`do`...`end` se ponovi.