Специфікація мови програмування MP_2

1 Вступ

Представлена тут мова програмування MP_2 — імперативна мова загального призначення. Прототип синтаксичних конструкцій — мова `Pascal`.

Назва промовляється як "ем пе два". Назва асоціюється з абревіатурою фрази "Мова Програмування" (транслітерація латиницею), індекс натякає на її неунікальність.

1.1 Обробка

Програма, написана мовою MP_2, подається на вхід транслятора (компілятора або інтерпретатора) для трансформації (перекладу, трансляції) до цільової форми (мови). Результат трансляції виконується у системі часу виконання (run-time system), для чого приймає вхідні дані та надає результат виконання програми.

Трансляція передбачає фази лексичного, синтаксичного та семантичного аналізу, а також фазу генерації коду. Фаза лексичного аналізу здійснюється окремим проходом.

1.2 Нотація

Для опису мови MP_2 використовується розширена нотація (форма) Бекуса – Наура, див. табл. 1.

Ланцюжки, що починаються з великої літери вважаються нетерміналами (нетермінальними символами). Термінали (термінальні символи) — ланцюжки, що починаються з маленької літери, або ланцюжки, що знаходяться між одинарними чи подвійними лапками. Для графічного представлення граматики використовуються синтаксичні діаграми Вірта.

Метасимвол	Значення
`=`	визначається як
`|`	альтернатива
`[ x ]`	0 або 1 екземпляр x
`{ x }`	0 або більше екземплярів x
`( x | y )`	групування: будь -який з x або y
`Abc`	нетермінал
`abc`	термінал
`'1'`	термінал (не літера)
`''1''`	термінал (не літера)
`''+''`	термінал (не літера)

Таблиця 1: Прийнята нотація РБНФ

2 Лексична структура

Лексичний аналіз[1] виконується окремим проходом, отже не залежить від синтаксичних та семантичних аспектів. Лексичний аналізатор розбиває текст програми на лексеми.

2.1 Алфавіт

Текст програми мовою MP_2 може містити тільки такі символи (characters): літери, цифри та спеціальні знаки.

```
Letter = 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' | 'i' | 'j'
       | 'k' | 'l' | 'm' | 'n' | 'o' | 'p' | 'q' | 'r' | 's' | 't'
       | 'u' | 'v' | 'w' | 'x' | 'y' | 'z'

Digit  = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

SpecSign = '.' | ',' | ':' | ';' | '(' | ')'
         | '=' | '+' | '-' | '*' | '/' | '<' | '>'
         |WhiteSpace | EndOfLine

WhiteSpace = ' ' | '\t'

EndOfLine   = '\n' | '\r'
```

2.2 Спеціальні символи

У програмі мовою MP_2 використовуються лексеми, що класифікуються як спеціальні символи, ідентифікатори, беззнакові цілі константи, беззнакові дійсні константи, логічні константи та ключові слова.

```
SpecSymbols = ArithOp | RelOp | BracketsOp | AssignOp | Punct

ArithOp =  AddOp | MultOp

AddOp   = '+' | '-'

MultOp  =  '*' | '/' | div

RelOp   = '=' | '<= ' | '<' | '>' | '>=' | '<>'

BracketsOp = '(' | ')'

AssignOp   = ':='
```

[1] Вся інформація, необхідна для побудови лексичного аналізатора, міститься у цьому розділі.

```
Punct      =  '.' | ',' | ':' | ';'
```

До спеціальних символів належать арифметичні оператори, оператори відношення, оператор присвоювання та знаки пунктуації.

2.3 Ідентифікатори

У мові MP_2 ідентифікатори використовуються для позначення змінних та програм.

Першим символом ідентифікатора може бути тільки літера, наступні символи, якщо вони є, можуть бути цифрами або літерами, або у формі правила:

```
Ident = Letter {Letter | Digit }
```

Довжина ідентифікатора не обмежена.

Жоден ідентифікатор не може збігатись із ключовим (вбудованим, зарезервованим) словом або словами `true`, `false` чи `div`. Якщо у фазі лексичного аналізу визначена лексема, яка синтаксично могла би бути ідентифікатором, але збігається із ключовим словом, то вона вважається ключовим словом. Якщо збігається з словами `true` чи `false`, то вважається значенням логічної константи. Якщо збігається з словами `div`, то вважається оператором цілочислового ділення.

Діаграма Вірта ідентифікатора представлена на рис. 1.

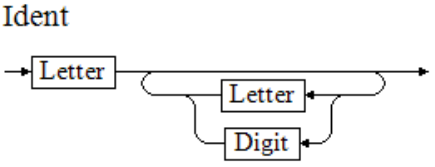

Рис. 1: Синтаксична діаграма ідентифікатора

Приклади:

```
a, x1, time24
```

2.4 Константи

Літерали використовуються для представлення постійних значень: цілих числових констант `IntNumb`, дійсних числових констант `RealNumb` та логічних констант `BoolConst`.

Їх синтаксис визначається такими продукціями:

```
Const    = IntNumb | RealNumb | BoolConst

IntNumb  = [Sign] UnsignedInt
```

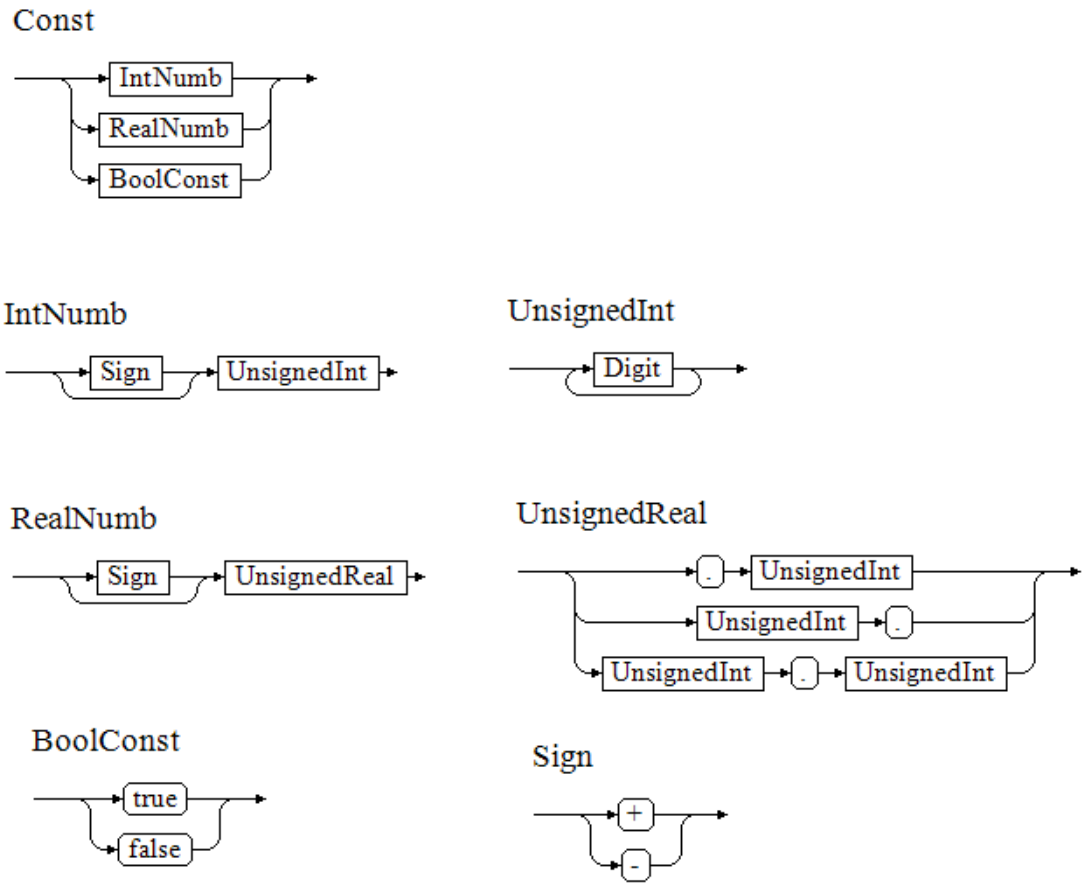

Рис. 2: Константи

```
RealNumb = [Sign] UnsignedReal

Sign     = '+' | '-'

UnsignedInt  = Digit {Digit}

UnsignedReal = '.' UnsignedInt
             | UnsignedInt '.'
             | UnsignedInt '.' UnsignedInt

BoolConst = true | false
```

Тип кожної константи визначається її формою, а значення має знаходитись у діапазоні репрезентативних значень для її типу.

На етапі лексичного аналізу виявляються тільки беззнакові цілі константи `UnsignedInt`, беззнакові дійсні константи `UnsignedReal` та логічні константи `BoolConst`. Унарні мінус чи плюс розпізнаються при синтаксичному розборі.

Синтаксичні діаграми констант представлені на рис. 2

Приклади беззнакових числових констант:

```
12, 234, 1.54, 34.567, 23. ,  true, false
```

2.5 Ключові слова

Мова MP_2 містить такі ключові слова:

```
program  var  begin end end. integer real boolean read write
for to downto do
```

2.6 Токени

З потоку символів вхідної програми на етапі лексичного аналізу виокремлюються послідовності символів з певним сукупним значенням, — *токени*. Список токенів мови MP_2 див. у табл. 2 на стор. 5.

Єдиний приклад засвідчує унікальність лексеми. Текст "символ ... ", наприклад "символ program", означає, що вказаний символ (symbol) є терміналом.

3 Типи даних і змінні

Значення у мові MP_2 можуть бути представлені або літералами (константами), або значеннями змінних, представлених у тексті програми своїми ідентифікаторами.

3.1 Типи даних

Мова MP_2 підтримує значення трьох скалярних типів: `integer`, `real` та `bool`.

1. Цілий тип `integer` може бути представлений оголошеною змінною типу `integer`, або константою `IntNumb`. Діапазон значень визначається реалізацією мови.

2. Дійсний тип `real` може бути представлений оголошеною змінною типу `real`, або константою `RealNumb`. Діапазон значень визначається реалізацією мови.

3. Логічний тип `bool` може бути представлений оголошеною змінною типу `bool`, або константою `BoolConst` (`true` або `false`).
   Прийнято, що `false < true`

Інших типів чи механізмів конструювання типів у мові MP_2 не передбачено.

При обчисленні виразів, за необхідності, виконується неявне приведення числових типів, див. розд. 5.2.1.

Код	Приклади лексем	Токен	Неформальний опис
1	a, x1, z12f	id	ідентифікатор
2	123, 0, 521	intnum	ціле без знака
3	.012, 34.76, 876.	realnum	дійсне без знака
4	true	boolval	логічне значення
5	false	boolval	логічне значення
6	program	keyword	символ program
7	var	keyword	символ var
8	begin	keyword	символ begin
9	end	keyword	символ end
10	end.	keyword	символ end.
11	integer	keyword	символ integer
12	real	keyword	символ real
13	boolean	keyword	символ boolean
14	read	keyword	символ read
15	write	keyword	символ write
16	for	keyword	символ for
17	to	keyword	символ to
18	downto	keyword	символ to
19	do	keyword	символ do
20	:=	assign_op	символ :=
21	+	add_op	символ +
22	-	add_op	символ -
23	*	mult_op	символ *
24	/	mult_op	символ /
25	div	mult_op	символ div
26	<	rel_op	символ <
27	<=	rel_op	символ <=
28	=	rel_op	символ =
29	>=	rel_op	символ >=
30	<>	rel_op	символ <>
31	(	brackets_op	символ (
32	)	brackets_op	символ)
33	.	punct	символ .
34	,	punct	символ ,
35	:	punct	символ :
36	;	punct	символ ;
37	\32	ws	пробільні символи (пробіл)
38	\t	ws	пробільні символи (г. табуляція)
39	\n	eol	роздільник рядків (новий рядок)
40	\r\n	eol	роздільник рядків

Таблиця 2: Таблиця лексем мови MP_2

3.2 Змінні

Тип кожної змінної має бути визначений у розділі оголошень, див. розд. 4.

Використання неоголошеної змінної, викликає помилку на етапі трансляції.

Використання змінної, що не набула значення, викликає помилку. Змінна набуває значення в інструкціях (statements) присвоювання `Assign` та/або введення `Inp`.

4 Оголошення

Оскільки мова MP_2 не передбачає конструювання типів чи структур даних, то оголошення стосуються тільки змінних.

Оголошення (декларація) специфікує набір ідентифікаторів, які можуть бути використані у програмі.

Синтаксис розділу оголошень визначається такими правилами:

```
var DeclarList = Declaration {';' Declaration }

Declaration    = IdenttList ':' Type

IdenttList     = Ident {',' Ident}

Type           = integer
               | real
               | boolean
```

Кожен ідентифікатор має бути оголошений і тільки один раз. Отже, повторне оголошеної змінної та використання неоголошеної змінної викликають помилку на етапі трансляції.

Значення оголошеної змінної залишається невизначеним аж до присвоєння їй значення у інструкції присвоєння або введення. Використання змінної, що не набула значення, викликає помилку.

Область видимості змінної (scope) — вся програма.

Діаграми Вірта розділу декларацій див. на рис. 3

Приклад:

```
var
    x1, y, a : real;
    c7, b : boolean;
    k, m, r, w : integer
```

5 Вирази

Вираз — це послідовність операторів і операндів, що визначає порядок обчислення нових значень.

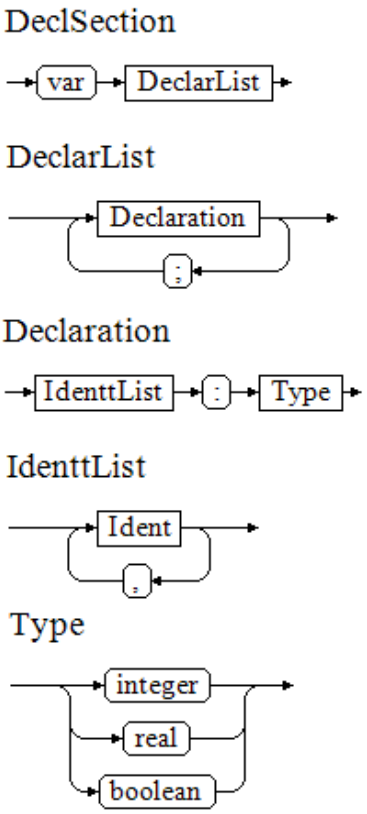

Рис. 3: Синтаксичні діаграми розділу оголошень

5.1 Граматика виразів

У мові MP_2 розрізняються арифметичні та логічні вирази. Значення, обчислене за арифметичним виразом, має тип `real` або `integer`. Значення, обчислене за логічним виразом, має тип `boolean`.

Синтаксичні правила[2] виразів такі:

```
Expression = ArithmExpression
           | BoolExpr

BoolExpr   = Expression RelOp Expression
           | true
           | false

ArithmExpression = [ Sign] Term
                 | ArithmExpression '+' Term
                 | ArithmExpression '-' Term

Term       = Factor
           | Term '*' Factor
           | Term '/' Factor
```

[2]Надалі ця граматика має бути трансформована з метою усунення лівої рекурсії.

```
Factor      = Ident
            | Const
            | '(' ArithmExpression ')'
```

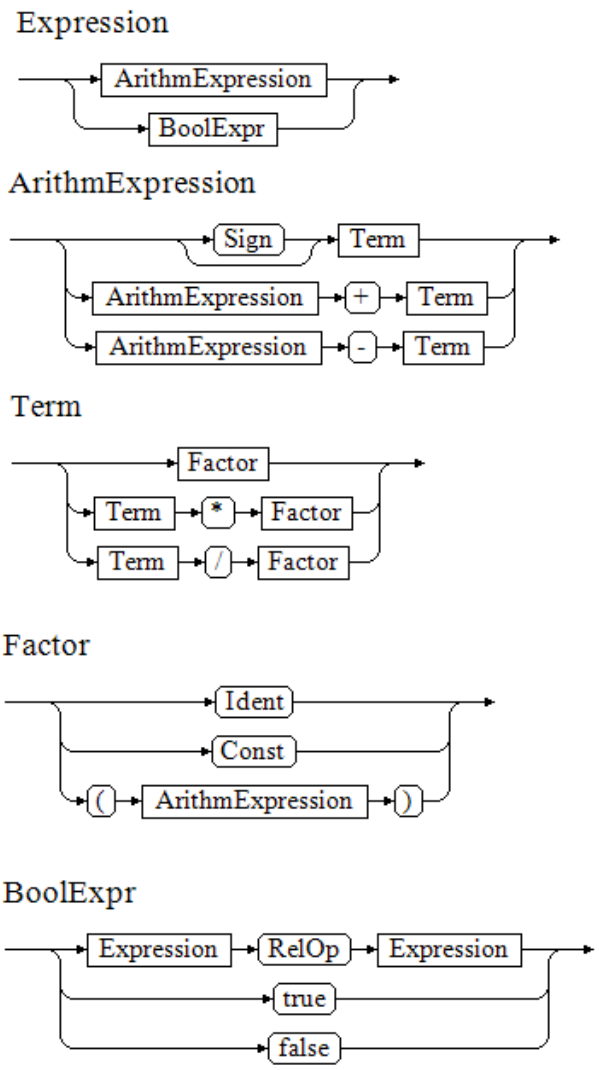

Рис. 4: Синтаксичні діаграми виразів мови MP_2

Приклади нетерміналів:
Factor:

```
x, 12, (a + 234)
```

Term:

```
m*z, 32/(b + 786)
```

ArithmExpr:

```
-b, f1 + g, c - 24
```

BoolExpr:

```
-b = 2, (a*x + b/z) >= (k + t), true > false, true, false
```

5.2 Оператори (operators)

Всі бінарні оператори у виразах цієї мови лівоасоціативні[3].

Найвищий пріоритет мають унарний мінус та унарний плюс, далі, у порядку зменшення пріоритету слідують `MultOp`, `AddOp` та `RelOp`.

5.2.1 Арифметичні оператори

Арифметичні оператори граматично представлені у формі:

```
AddOp  = '+'
       | '-'

MultOp = '*'
       | '/'
       | div
```

Тип результату при застосуванні бінарних операторів див. у табл. 3.

Оператор	Операція	Типи операндів	Тип результату
`+`	додавання	`integer` або `real`	`real`, якщо хоч один `real`, інакше `integer`
`-`	віднімання	`integer` або `real`	
`*`	множення	`integer` або `real`	
`/`	ділення	`integer` або `real`	`real`
`div`	цілочисельне ділення	`integer`	`integer`

Таблиця 3: Бінарні арифметичні оператори

Неявне приведення типів відбувається шляхом інтерпретації значення цілого числа як дійсного з нульовою дробовою частиною.

Ділення на нуль викликає помилку.

Тип результату при застосуванні унарних операторів не змінюється, див. табл. 4.

Приклади:

```
1.234*x1/45.67,  -3.4+6, 7/8, 12 div 5
```

[3]Адже мова не містить правоасоціативних операторів, наприклад, оператора піднесення до степеня

Оператор	Операція	Типи операнда	Тип результату
+	ідентичності	integer real	integer real
-	зміни знаку	integer real	integer real

Таблиця 4: Унарні арифметичні оператори

5.2.2 Оператори відношення

Оператори відношення представлені у граматиці правилами:

```
RelOp = '=' | '<= ' | '<' | '>' | '>=' | '<>'
```

Значення обох операндів мають бути або числовими (типу `integer` або `real`), або логічними (типу `bool`). Результат завжди має тип `bool`.

Якщо один з операндів має тип `integer`, а інший — `real`, то значення типу `integer` приводиться до типу `real`.

Приклади:

```
x1+3 < 1, 0.5*2.34 <= 15 div 7,  (z-3>0) <> (2>=m),  true > false
```

6 Розділ інструкцій (statements)

Інструкції (statements) визначають алгоритмічні дії, які мають бути виконані у програмі.

Мова MP_2 передбачає наявність у програмі секції (розділу) інструкцій (statements), представленої граматично правилами[4]:

```
DoSection     = begin  StatementList  'end.'

StatementList = Statement {';' Statement }

Statement     = Assign
              | Inp
              | Out
              | ForStatement
```

Розділі інструкцій може містити одну чи більше інструкцій, але не менше однієї.

Синтаксичні діаграми розділу інструкцій див. на рис. 5

Приклад:

```
begin
   read(z,a,b);
   x := (z + a)/( z+b )
end.
```

[4]Мова MP_2 визначена як приклад для специфікації, тож вона не містить, наприклад, обов'язкових для імперативних мов інструкцій розгалуження (чи умовного переходу).

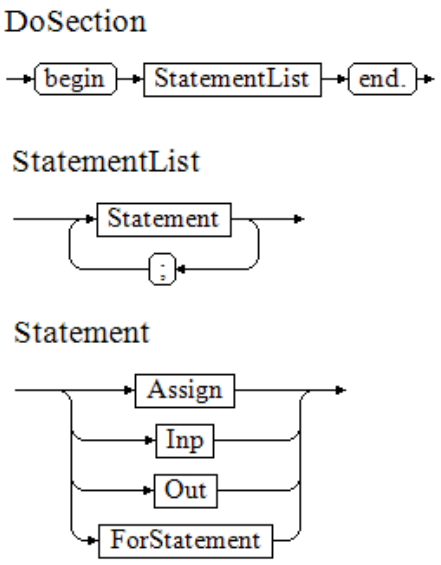

Рис. 5: Синтаксичні діаграми розділу інструкцій

6.1 Оператор (інструкція) присвоєння значення

Синтаксис інструкції присвоєння визначається правилом

```
Assign = Ident ':=' Expression
```

В інструкції присвоєння змінна, позначена ідентифікатором (`Ident`) отримує значення виразу (`Expression`).

В інструкції присвоєння змінні використовуються по різному, в залежності від того, знаходиться вона зліва чи справа від символу ':='.

Якщо змінна зліва від оператора (operator) присвоєння ':=', то кажуть про її `l-значення` (`lvalue` чи `left-value`). `l-значення` має тип вказівника на місце зберігання значення змінної з ідентифікатором `Ident`.

`r-значенням` (або `lvalue` та `rvalue`, або `left-value` та `right-value`). Кажуть, що змінна, якЗначення, які можуть використовуватись у лівій та правій частинах інструкції присвоювання називають `l-значенням` та `r-значенням` (або `lvalue` та `rvalue`, або `left-value` та `right-value`).

Якщо змінна знаходиться справа від оператора (operator) присвоєння ':=', то кажуть про її `r-значення` (`rvalue`, `right-value`).

Про значення виразу `Expression`, який теж знаходиться праворуч від оператора присвоювання ':=', також можна говорити як про `r-значенням`.

Тип змінної з ідентифікатором `Ident` повинен збігатись з типом значенням виразу `Expression` (типом `r-значення`).

Діаграма Вірта представлена на рис. 6.

Приклади:

```
f5 := 3/4+1.23,  b7 := 2+3 < 7 div 5
```

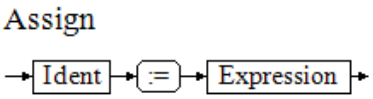

Рис. 6: Синтаксична діаграма оператор присвоювання

6.2 Інструкція введення

Для отримання вхідних даних у програмі мовою PL_2 передбачена інструкція введення з клавіатури, з таким синтаксисом:

```
Inp = read '(' IdenttList ')'.
```

Значення вводяться з клавіатури. Введення кожного окремого значення підтверджується клавішею `Enter`.

Прочитані з клавіатури значення присвоюються змінним зі списку ідентифікаторів `IdenttList`.

Введені значення мають допускати їх інтерпретацію як даних типу, що збігаються (або можуть бути приведені) з типом змінних, яким вони має бути присвоєне. У разі неможливості такої інтерпретації виникає помилка часу виконання.

Діаграма Вірта інструкції введення представлена на рис. 7.

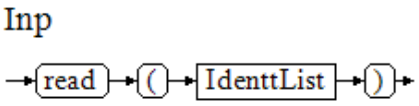

Рис. 7: Синтаксична діаграма інструкції введення

Приклад:

```
read(a,v2,len7) .
```

6.3 Інструкція виведення

Для повідомлення вихідних даних у програмі мовою PL_2 передбачена інструкція виведення, з таким синтаксисом:

```
Out = write '(' IdenttList ')'.
```

Всі значення списку виводяться у один рядок консолі. Кожне значення виводиться у форматі \t`Ident=значення`.

Спроба виведення змінної з невизначеним значенням викликає помилку.

Діаграма Вірта інструкції введення представлена на рис. 8.

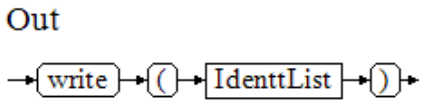

Рис. 8: Синтаксична діаграма інструкції виведення

6.4 Оператор циклу (інструкція повторення)

Для реалізації ітеративного виконання певної групи інструкцій використовується інструкція повторення з таким синтаксисом:

```
ForStatement = for Ident ':='  ArExpr1 (to | downto) ArExpr2 do DoBlock

ArExpr1 = ArithmExpression

ArExpr2 = ArithmExpression

DoBlock = Statement
        | 'begin' StatementList 'end'
```

Виконання інструкція повторення `ForStatement` у мові PL_2 зводиться до виконання тіла циклу `DoBlock` у кількох ітераціях.

Параметр циклу `Ident` — змінна типу `integer`, оголошена у секції декларацій.

Перед першою ітерацією: обчислюється арифметичний виразу `ArExpr1`. Його значенням (типу `integer`) ініціюється параметр циклу `Ident`.

Виконується тіло циклу `DoBlock`.

Перед другою та наступними ітераціями: нове значення параметра циклу `Ident` обчислюється як $Ident = Ident + 1$, якщо використано ключове слово `to`, або як $Ident = Ident - 1$, якщо використано ключове слово `downto`.

Виконання тіла циклу у другій та наступних ітераціях здійснюється тільки якщо поточне значення $Ident \leq ArExpr2$ (ключове слово `to`), або $Ident \geq ArExpr2$ (ключове слово `downto`). Інакше інструкція повторення вважається виконаною.

Отже, тіло інструкції повторення `DoBlock` виконується один або більше разів.

Діаграма Вірта інструкції повторення наведено на рис. 9

Приклади: Синтаксис

1. ```
   for i := 3 - 5 to 8 div 3 do
       write(i)
   ```

2. ```
   for i := 2 to 5 do
       begin
         write(i);
         for j := 10  to 15 do
           begin
   ```

```
            x := i * j;
            write(i,j,x)
        end
    end
```

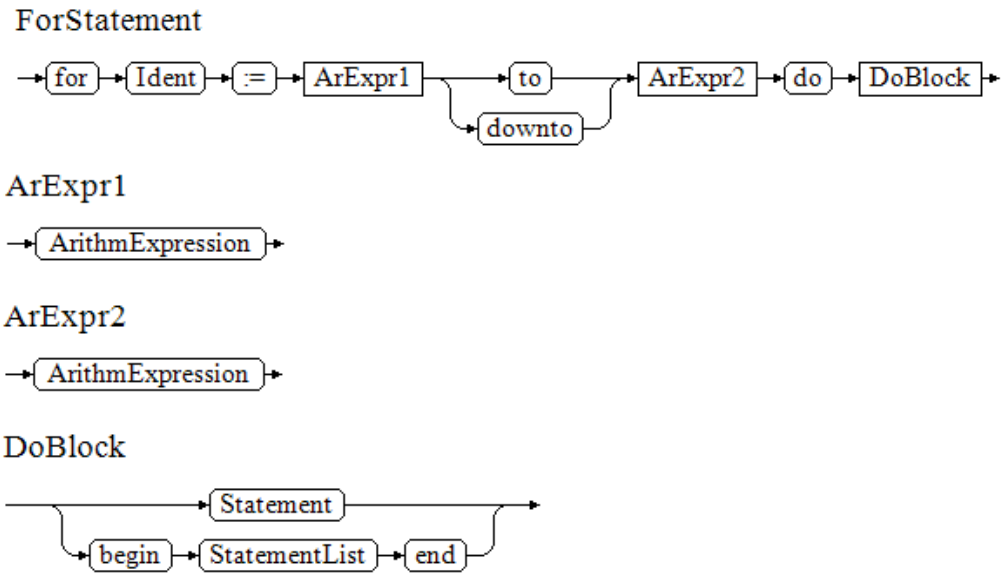

Рис. 9: Синтаксичні діаграми інструкції повторення

7 Програма

У мові PL_2, яка не надає інших засобів комунікації із оточенням окрім взаємодії зі стандартним потоком введення/виведення даних, кожна програма є монолітною програмною сутністю, ізольованою від будь-якого іншого коду.

7.1 Структура програми

Програма має синтаксис, визначений такими продукціями:

```
Program  = program ProgName
             DeclSection
             DoSection

ProgName = Ident
```

Кожна програма починається з термінала `program` та ідентифікатора програми; за ідентифікатором програми — (єдиний) розділ оголошень; далі — (єдиний) розділ інструкцій.

Ідентифікатор програми ніяк не використовується у самій програмі.

Діаграма Вірта представлена на рис. 10.

Приклад:

```
program counter
   var
      n:integer;
      x:real;
      b:bool
begin
   read(n);
   for i := 1 to n do
      begin
          read(x);
          b := x > 0;
          write(i,x,b)
      end
end.
```

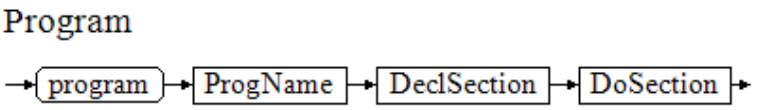

Рис. 10: Синтаксична діаграма кореневого нетермінала граматики мови PL_2

7.2 Базовий приклад програми

```
program basexample
  var
    i,j,n,k:integer;
    x,y1,y2:real;
    b1,b2,b3,b4,b5,b6:bool
begin
  read(n)
  for i := 1 to 2 do
    begin
      for j := n downto 1 do
        begin
          read(x);
          b1 := x > 0;
          b2 := x >= 0;
          b3 := x < 0;
          b4 := x <= 0;
          b5 := x = 0;
          b6 := x <> 0;
          write(i,j,x,b1,b2,b3,b4,b5,b6);
          k := n div j;
```

```
          write(n,j,k);
          y1 := (x+x) / 2 * 10.0;
          y2 := -(x+x) / 2.0 * (-10.0) - y1;
          write(x,y1,y2)
      end;
      b1 := true;
      b2 := false;
      b3 := b1 > b2;
      write(b1,b2,b3)
      end;
      b3 := b3 <> b3;
      write(b3)
end.
```

8 Повна граматика мови MP_2

```
Program = program ProgName DeclSection DoSection

ProgName = Ident

Ident = Letter {Letter | Digit}

DeclSection = var DeclarList

DeclarList = Declaration {';' Declaration}

Declaration = IdenttList ':' Type

IdenttList = Ident {',' Ident}

Type = integer
     | real
     | boolean

DoSection = begin  StatementList  'end.'

StatementList = Statement {';' Statement}

Statement = Assign
          | Inp
          | Out
          | ForStatement

Assign = Ident ':=' Expression
```

```
Expression = ArithmExpression
           | BoolExpr

BoolExpr   = Expression RelOp Expression
           | true
           | false

ArithmExpression = [Sign] Term
                 | ArithmExpression '+' Term
                 | ArithmExpression '-' Term

Term = Factor | Term '*' Factor | Term '/' Factor

Factor = Ident
       | Const
       | '(' ArithmExpression ')'

Inp = read '(' IdenttList ')'

Out = write '(' IdenttList ')'

ForStatement = for Ident ':='  ArExpr1 (to | downto) ArArExpr2 do DoBlock

ArExpr1 = ArithmExpression

ArExpr2 = ArithmExpression

DoBlock = Statement
        | 'begin' StatementList 'end'

Const = IntNumb | RealNumb | BoolConst

IntNumb = [Sign] UnsignedInt

RealNumb = [Sign] UnsignedReal

Sign = '+'
     | '-'

UnsignedInt = Digit {Digit}

UnsignedReal = '.' UnsignedInt
             | UnsignedInt '.'
             | UnsignedInt '.' UnsignedInt

Letter = 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' | 'i' | 'j'
```

```
       | 'k' | 'l' | 'm' | 'n' | 'o' | 'p' | 'q' | 'r' | 's' | 't'
       | 'u' | 'v' | 'w' | 'x' | 'y' | 'z'

Digit = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

BoolConst = true | false

RelOp = '='
      | '<= '
      | '<'
      | '>'
      | '>='
      | '<>'
```