**Специфікація мови програмування KK**

1. **Вступ**

Представлена тут мова програмування KK – імперативна мова загального призначення.

* 1. **Завдання**

Варіант №12

**Арифметика**: цілі та дійсні числа, основні чотири арифметичні операції (додавання, віднімання, ділення та множення), піднесення до степеня(правоасоціативна операція), дужки

**Особливості**: унарний мінус

**Інструкція повторення**: for <ід>=<вираз> by <вираз> while<відношення> do <оператор>

**Інструкція розгалуження:** if <відношення> {<список операторів>}

* 1. **Обробка**

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

Трансляція передбачає фази лексичного, синтаксичного та семантичного аналізу, а також фазу генерації коду. Фази лексичного та синтаксичного аналізу здійснюється окремими проходами.

* 1. **Нотація**

| Метасимвол | Значення |
| --- | --- |
| = | визначається як |
| | | альтернатива |
| [ x ] | 0 або 1 екземпляр х |
| { x } | 0 або більше екземплярів х |
| ( x | y ) | групування: будь-який з х або у |
| Zxy | нетермінал |
| zxy | термінал |
| ‘1’ | термінал |
| “1” | термінал |

Табл. 1. Прийнята нотація РБНФ

Для опису мови KK використовується розширена форма Бекуса-Наура. Ланцюжки, що починаються з великої літери, вважаються нетермінальними символами. Термінальні символи – ланцюжки, що починаються з маленької літери, або ланцюжки, що знаходяться між одинарними або подвійними лапками. Для графічного представлення граматики використовуються синтаксичні діаграми Вірта.

* 1. **Алфавіт**

Програма може містити текст з використанням таких символів (character) – літер, цифр та спеціальних знаків.

Синтаксис:

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’

SpecialSign = ‘.’ | ‘,’ | ‘:’ | ‘;’ | ‘(’ | ‘)’ | ‘{’ | ‘}’ | ‘=’ | ‘+’ | ‘-’ | ‘\*’ | ‘/’ | ‘^’ | ‘<’ | ‘>’ | ‘WhiteSpace | EndOfLine

WhiteSpace = ‘ ’ | ‘\t’

EndOfLine = ‘\n’ | ‘\r’ | ‘\n\r’ | ‘\r\n’

1. **Лексика**

Лексичний аналіз виконується окремим проходом, отже не залежить від синтаксичних та семантичних аспектів. Лексичний аналізатор розбиває текст програми на лексеми. У програмі мовою KK можуть використовуватись лексичні елементи, що класифікуються як спеціальні символи, ідентифікатори, беззнакові цілі константи, беззнакові дійсні константи, логічні константи та ключові слова.

* 1. **Спеціальні символи**

*Синтаксис*

SpecSymbols = ArithOp | RelOp | BracketsOp | AssignOp | Punct

ArithOp = AddOp | MultOp

AddOp = ’+’ | ’-’

MultOp = ’\*’ | ’/’

NeltOp = ‘^’

RelOp = ’==’ | ’<= ’ | ’<’ | ’>’ | ’>=’ | ‘!=’

BracketsOp = ’(’ | ’)’

AssignOp = ’=’

Punct = ’,’ | ’;’

*Опис*

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

*Примітка*

Набір токенів, див табл.2, містить токени add\_op, mult\_op та nelt\_op, але не містить arith\_op – відповідника нетермінала ArithOp.

* + 1. **Ідентифікатори**

*Синтаксис*

Ident = Letter { Letter | Digit }

*Опис*

Першим символом ідентифікатор може бути тільки літера, наступні символи, якщо вони є, можуть бути цифрами або літерами. Довжина ідентифікатора не обмежена.

*Обмеження*

Жоден iдентифiкатор не може збiгатись iз ключовим (вбудованим, зарезервованим) словом або словом true, або false.

Елемент, який у фазi лексичного аналiзу може бути визначений як iдентифiкатор або як ключове слово, вважається ключовим словом.

Елемент, який у фазi лексичного аналiзу може бути визначений як iдентифiкатор або як логiчна константа, вважається логiчною константою.

*Семантика*

Ідентифікатор може позначати змінну та програму

*Візуальне представлення*

Синтаксична діаграма

*Приклади*

а, xyz1, data3

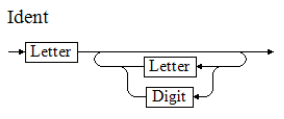


Рисунок 1 Ідентифікатор

* 1. **Константи**

*Синтаксис*

Const = IntNumb | RealNumb | BoolConst

IntNumb = [Sign] UnsignedInt

RealNumb = [Sign] UnsignedReal

Sign = ’-’

UnsignedInt = Digit {Digit}

UnsignedReal = ’.’ UnsignedInt

| UnsignedInt ’.’

| UnsignedInt ’.’ UnsignedInt

BoolConst = true | false

*Обмеження*

Кожна константа повинна мати тип, а величина константи повинна знаходитись у діапазоні репрезентативних значень для її типу.

На етапі лексичного аналізу виявляються тільки беззнакові цілі константи UnsіgnedІnt, беззнакові дійсні константи UnsіgnedFloat та логічні константи BoolConst.

*Семантика*

Кожна константа має тип, визначений її формою

*Візуальне представлення*

Синтаксичні діаграми див. на рис. 1.

*Приклади*

12, -5, 234, -1.54, 34.567, 23. , true, false

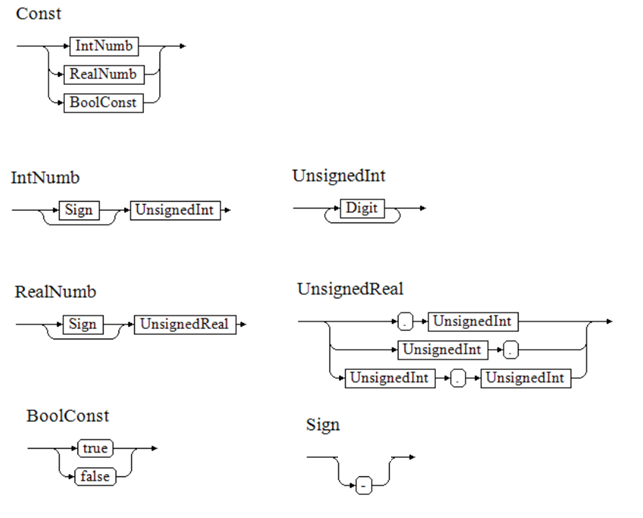


Рисунок 2 Константи

**2.3 Ключові слова**

*Синтаксис*

KeyWords = start | read | print | for | by | while | do | if | end

**2.4 Токени**

З потоку символів вхідної програми на етапі лексичного аналізу виокремлюються послідовності символів з певним сукупним значенням, — токени. Список токенів мови KK див. у табл. 2. Єдиний приклад засвідчує унікальність лексеми. Текст "символ . . . ", наприклад "символ program", означає, що вказаний символ (symbol) є терміналом.

| Код | Приклади лексем | Токен | Неформальний опис |
| --- | --- | --- | --- |
| 1 | а, xyz1, data3 | іdent | ідентифікатор |
| 2 | 123, 0, 521 | іntnum | ціле без знака |
| 3 | 34.76 | realnum | дійсне без знака |
| 4 | -6.7 | realnum | дійсне зі знаком |
| 5 | true | boolval | логічне значення |
| 6 | start | keyword | символ start |
| 7 | read | keyword | символ read |
| 8 | print | keyword | символ print |
| 9 | for | keyword | символ for |
| 10 | by | keyword | символ by |
| 14 | do | keyword | символ do |
| 16 | while | keyword | символ while |
| 17 | if | keyword | символ if |
| 18 | end | keyword | символ end |
| 19 | false | boolval | символ false |
| 20 | = | assіgn\_op | символ = |
| 21 | + | add\_op | символ + |
| 22 | - | add\_op | символ - |
| 23 | \* | mult\_op | символ \* |
| 24 | / | mult\_op | символ / |
| 25 | ^ | nelt\_op | символ ^ |
| 26 | < | rel\_op | символ < |
| 27 | <= | rel\_op | символ <= |
| 28 | == | rel\_op | символ === |
| 29 | > | rel\_op | символ > |
| 30 | >= | rel\_op | символ >= |
| 31 | != | rel\_op | символ !== |
| 32 | ( | par\_op | символ ( |
| 33 | ) | par\_op | символ ) |
| 34 | { | par\_op | символ { |
| 35 | } | par\_op | символ } |
| 36 | . | punct | символ . |
| 37 | , | punct | символ , |
| 38 | : | punct | символ : |
| 39 | ; | punct | символ ; |
| 40 | \32,\t | ws | пробільні символи |
| 41 | \n, \r, \r\n, \n\r | ls | кінець рядка |

Табл. 2: Таблиця лексем мови KK

1. **Типи**

Мова KK пiдтримує значення трьох типiв: integer, real та boolean.

Цiлий тип integer може бути представлений оголошеною змiнною типу

integer, або константою IntNumb. Дiапазон значень — [ -32768; 32767 ].

Дiйсний тип real може бути представлений оголошеною змiнною типу

real, або константою RealNumb. Дiапазон значень, який підтримує транслятор.

Логiчний тип boolean може бути представлений оголошеною змiнною типу boolean, або константою BoolConst (true або false).

Прийнято, що false < true

**4. Синтаксис**

**4.1 Вирази**

*Синтаксис*

Expression = ArithmExpression | BoolExpr

BoolExp = ArithmExp RelOp ArithmExp | BoolExp ( '==' | '!=' ) BoolExp | BoolConst

ArithmExpression = [Sign] Term

| ArithmExpression ’+’ Term

| ArithmExpression ’-’ Term

Term = Factor

| Term ’\*’ Factor

| Term ’/’ Factor

| ‘(‘ Tern ‘^’ Factor ‘)’

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

*Опис*

Вираз - це послідовність операторів і операндів, що визначає порядок обчислення значення.

Розрізняються арифметичні та логічні вирази.

Значення, обчислене за арифметичним виразом, має тип float або int.

Значення, обчислене за логічним виразом, має тип boolean.

Всі бінарні оператори у виразах цієї мови лівоасоціативні, за винятком оператора ‘^’ та унарного мінуса.

Найвищий пріоритет в унарного мінуса, далі, у порядку зменшення пріоритету слідують NeltOp, MultOp, AddOp та RelOp.

Послідовність двох або більше операторів з однаковим пріоритетом асоціативна.

*Обмеження*

Тип кожної змінної має бути визначений у розділі оголошень.

Повторне оголошеної змінної викликає помилку на етапі (у фазі) трансляції.

Використання неоголошеної змінної, викликає помилку на етапі трансляції.

Використання змінної, що не набула значення, викликає помилку.

*Семантика*

Кожна константа має тип, визначений її формою та значенням.

Змінна набуває значення в інструкції присвоювання Assіgn або в інструкції введення Іnp.

*Візуальне представлення*

Синтаксична діаграма.

Приклади

Factor:

x, 12, (a + 234)

Term:

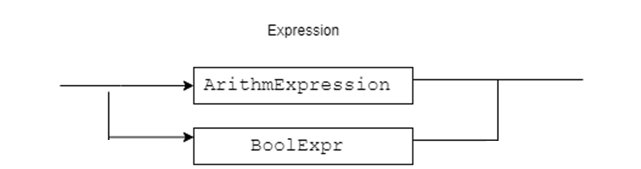
m\*z, 32/(b + 786), k^n

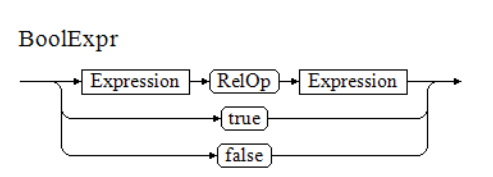
ArіthmExpr:

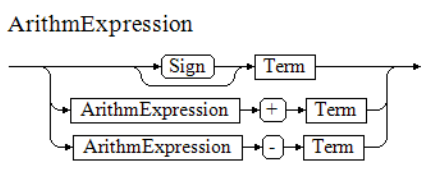
-b, f1 + g, c - 24

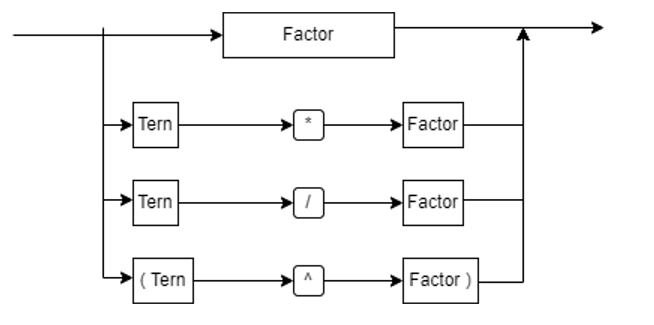
BoolExpr:

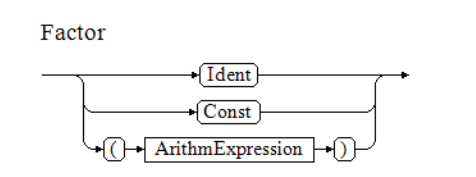
-b = 2, (a\*x + b/z) >= (k + t), true > false, true, false











* 1. **Оператори**
     1. **Арифметичні оператори**

*Синтаксис*

AddOp = ’+’ | ’-’

MultOp = ’\*’ | ’/’

NeltOp = ‘^’

*Семантика*

1. Тип результату при застосуванні бінарних операторів див. табл. 3.
2. Тип результату при застосуванні унарного мінуса див. табл. 4.
3. Ділення на нуль викликає помилку

*Приклад*

0.2 \* y / 5.6, -3.4 + 3, 7 / 8, 5 ^ 3.

| Оператор | Операція | Типи операндів | Тип результату |
| --- | --- | --- | --- |
| + | додавання | integer, real | real, якщо хоч один real, інакше - integer |
| - | віднімання |
| \* | множення | real |
| ^ | піднесення до степеня |
| / | ділення |

Таблиця 3 Бінарні арифметичні оператори

| Оператор | Операція | Типи операнда | Тип результату |
| --- | --- | --- | --- |
| - | зміни знаку | int або float | int або float |

Табл. 4. Унарні арифметичні оператори

* + 1. **Оператори відношення**

*Синтаксис*

RelOp = ‘==’ | ‘<=’ | ‘<’ | ‘>’ | ‘>=’ | ‘!=’

*Обмеження*

Значення обох операндів мають бути або числовими (типу int або float), або логічними (типу boolean).

Результат завжди має тип boolean.

*Семантика*

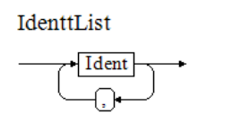
Якщо один з операндів має тип int, а інший - float, то значення типу приводяться до типу float.

*Приклад*

1. x1+3 < 1, 0.5 \* 2.34 <= 15 / 7, (z – 3 > 0) != (2 >= m), true > false
2. **Оголошення**

*Синтаксис*

IdentList = Ident{‘,’ Ident }



*Опис*

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

Оголошення ідентифікатор означає оголошення змінної.

*Обмеження*

Розділ оголошень знаходиться перед розділом інструкцій.

Кожен ідентифікатор має бути оголошений рівно один раз.

*Семантика*

Оголошення змінної означає виділення пам’яті для зберігання значення декларованого типу.

Значення оголошеної змінної залишається невизначеним аж до присвоєння їй значення у інструкції присвоєння або введення.

Область видимості змінної (scope) – вся програма.

*Приклад*

x, k;

1. **Розділ інструкцій**

Синтаксис

StatementList = Statement ‘;’ {Statement ‘;’}

Statement = Declaration | Assign | Input | Output | ForStatement | IfStatement

*Опис*

Iнструкцiї (statements) визначають алгоритмiчнi дiї, якi мають бути виконанi у програмi.

*Вiзуальне представлення*

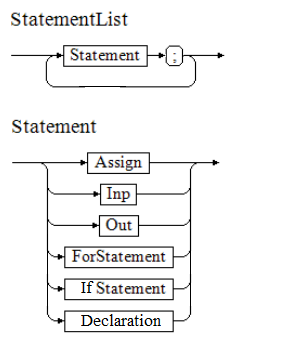


Рисунок 5. Інструкції

*Приклад*

real r;

read (r);

r = r + 5;

* 1. **Оператор (інструкція) присвоювання**

*Синтаксис*

Assign = Ident ‘=’ Expression

*Опис*

Значення, 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).

*Обмеження*

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

*Семантика*

l-значення має тип вказівника на місце зберігання значення змінної з ідентифікатором Ident.

r-значення має тип значення, обчисленого за виразом Expression.

*Візуальне представлення*

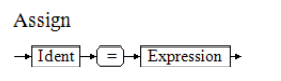


Рис. 4. Оператор присвоювання

*Приклад*

result = 3 ^ 4 + 1.2; check = 2 + 3 < 7;

* 1. **Інструкція введення**

*Синтаксис*

Inp = read ‘(’ IdentList ‘)’

*Опис*

Значення вводяться з клавіатури.

Введення кожного окремого значення підтверджується клавішею Enter.

*Обмеження*

Відмінність типу введеного значення від типу змінної викликає помилку.

*Візуальне представлення*

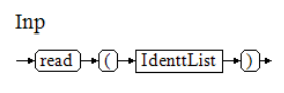


Рис. 5. Інструкція введення

Синтаксичну діаграму див. на рис. 5.

*Приклад*

read (value);

* 1. **Інструкція виведення**

*Синтаксис*

Out = print ‘(’ IdentList ‘)’

*Опис*

Всі значення списку виводяться у один рядок консолі.

Кожне значення виводиться у форматі \tIdent=значення

*Обмеження*

Виведення змінної з невизначеним значенням викликає помилку.

*Візуальне представлення*

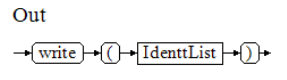


Рис.6. Інструкція виведення

Синтаксичну діаграму див. на рис. 6.

*Приклад*

print(value);

**6.4 Оператор циклу (iнструкцiя повторення)**

*Синтаксис*

ForStatement = for ‘(‘ IndExpr1 ‘)’ by ‘(‘ IndExpr2 ‘)’ while ’(‘ BoolExpr ‘)’ do ‘{‘ DoBlock ‘}’

IndExpr1 = Ident ’=’ ArithmExpression1

IndExpr2 = ArithmExpression2

DoBlock = ’{’ Statement ’}’ | ’{’ StatementList ’}’

*Опис*

Тiло оператора циклу DoBlock виконується один або бiльше разiв.

*Обмеження*

Параметр циклу Ident — змiнна типу intege, float.

Значення ArithmExpression1, ArithmExpression2 мають тип integer, real або приводяться до типу integer.

*Семантика*

Перед першою iтерацiєю обчислюються значення IndExpr1, IndExpr2, BoolExpr

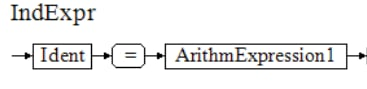
Перед першою iтерацiєю параметр циклу IndExpr1приймає значення ArithmExpression1.

DoBlock виконується , якщо значення BoolExpr = true.

*Вiзуальне представлення*

Синтаксичнi дiаграми див. на рис.9.





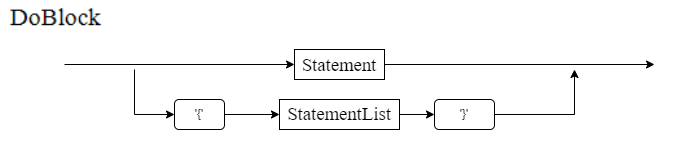


Рисунок 9 Оператор циклу

*Приклад*

for ( I = 1 ) by ( I = I + 1) while (I < 10) do {

print(i)

}

**6.5 Умовний оператор (iнструкцiя розгалуження)**

*Синтаксис*

IfStatement = if ‘(’ Relation ‘)’ ‘{‘ DoBlock ‘}’

Relation = BoolExpr

*Опис*

Тiло оператора циклу StatementList виконується при дотриманні умови Relation.

*Семантика*

StatementList виконується , якщо значення BoolExpr = true.

*Вiзуальне представлення*

Синтаксичнi дiаграми див. на рис. 10.

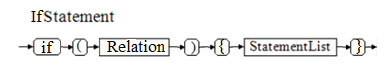


Рисунок 10 Умовний оператор

*Приклад*

I = 1;

If ( I < 10 ) {

print (i);

}

If ( I != 10 ) {

print (i);

}

If ( I = 10 ) {

print (i);

}

If ( I > 10 ) {

print (i);

}

If ( I =< 10 ) {

print (i);

}

If ( I => 10 ) {

print (i);

}

1. **Програма**

*Синтаксис*

Program = start ProgName ‘;’ StatementList end

ProgName = Ident

*Опис*

Кожна програма починається з термiнала start та iдентифiкатора програми; за iдентифiкатором програми — (єдиний) роздiл iнструкцiй. Закінчується терміналом — end.

Iдентифiкатор програми нiяк не використовується у програмi.

*Приклад*

start ProgramName

c = 1.0 \* 9.0 + 3.0;

c = -c;

d = 3.0 ^ (5.0 / 10.0);

for ( I = 1 ) by ( I = I + 1) while (I < 10) do {

print(i);

}

I = 1;

If ( I < 10 ) {

print (i);

}

If ( I != 10 ) {

print (i);

}

If ( I = 10 ) {

print (i);

}

If ( I > 10 ) {

print (i);

}

If ( I =< 10 ) {

print (i);

}

If ( I => 10 ) {

print (i);

}

end

1. **Повна граматика мови KK**

Program = start ProgName ‘;’ StatementList

ProgName = Ident

*Алфавіт*

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’

SpecSsign = ’.’ | ’,’ | ’:’ | ’;’ | ’(’ | ’)’| ’=’ | ’+’ | ’-’ | ’\*’ | ’/’ | ‘^’ | ’<’ | ’>’| WhiteSpace | EndOfLine

WhiteSpace = ’ ’ | ’\t’

EndOfLine = ’\n’ | ’\r’ | ’\r\n’ | ’\n\r’

*Символи*

SpecSymbols = ArithOp | RelOp | BracketsOp | AssignOp | Punct

ArithOp = AddOp | MultOp | NeltOp

AddOp = ’-’

MultOp = ’\*’ | ’/’

NeltOp = ‘^’

RelOp = ’==’ | ’<= ’ | ’<’ | ’>’ | ’>=’ | ’<>’ | ‘!=’

BracketsOp = ’(’ | ’)’

AssignOp = ’=’

Punct = ’.’ | ’,’ | ’:’ | ’;’

*Індифікатор*

Ident = Letter {Letter | Digit }

*Конастанти*

Const = IntNumb | RealNumb | BoolConst

IntNumb = [Sign] UnsignedInt

RealNumb = [Sign] UnsignedReal

Sign = ’-’

UnsignedInt = Digit {Digit}

UnsignedReal = ’.’ UnsignedInt| UnsignedInt ’.’ | UnsignedInt ’.’ UnsignedInt

BoolConst = true | false

*Ключові слова*

KeyWords = start | read | print | for | by | while | do | if | end

*Синтаксис*

Expression = ArithmExpression | BoolExpr

BoolExpr = Expression RelOp Expression | true | false

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

Term = Factor | Term ’\*’ Factor | Term ’/’ Factor | ‘(‘ Tern ‘^’ Factor ‘)’

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

*Арифметичні операції*

AddOp = ’+’| ’-’

MultOp = ’\*’ | ’/’

NeltOp = ‘^’

*Оголошення*

IdenttList = Ident {’,’ Ident}

*Інструкції*

DoSection = StatementList

StatementList = Statement {’;’ Statement }

Statement = Assign | Inp | Out | ForStatement | IfStatment

*Присвоєення*

Assign = Ident ’=’ Expression

*Введення*

Inp = read ’(’ IdenttList ’)’.

*Виведення*

Out = print ’(’ IdenttList ’)’.

*Інструкція повторення*

ForStatement = for ‘(‘ IndExpr1 ‘)’ by ‘(‘ IndExpr2 ‘)’ while ’(‘ BoolExpr ‘)’ do ‘{‘ DoBlock ‘}’

IndExpr1 = Ident ’=’ ArithmExpression1

IndExpr2 = ArithmExpression2

DoBlock = ’{’ Statement ’}’ | ’{’ StatementList ’}’

*Умовний оператор*

IfStatement = if ‘(’ Relation ‘)’ ‘{‘ DoBlock ‘}’

Relation = BoolExpr