Специфiкацiя мови програмування Crundras

# Вступ

Представлена тут мова програмування Crundras призначена слугувати піддослідним кроликом для курсу, що має на меті вивчення основ трансляції. Crundras — імперативна мова загального призначення.

## Завдання

Розробити Сі-подiбну мову програмування з оператором (інструкцією) циклу:

for <ід>=<вираз> to <вираз> by <вираз> while (<відношення>)  
<список операторів> rof;

та умовною інструкцією:

if (<відношення>) {<сп. операторів>}

## Обробка

Програма, написана мовою Crundras, подається на вхід транслятора (компілятора або інтерпретатора) для трансформації до цільової форми. Результат трансляції виконується у система 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. Фази лексичного та синтаксичного аналізу здійснюються окремими проходами.

## Нотацiя

Для опису мови Crundras використовується розширена форма Бекуса – Наура. Ланцюжки, що починаються з великої літери вважаються нетермiналами (нетермiнальними символами). Термінали — ланцюжки, що починаються з маленької літери, або знаходяться між одинарними, або подвійними лапками. Для

|  |  |
| --- | --- |
| Метасимвол | Значення |
| = | визначається як |
| | | альтернатива |
| [ x ] | 0 або 1 екземпляр x |
| { x } | 0 або бiльше екземплярiв x |
| ( x | y ) | групування: будь -який з x або y |
| Zxy | нетермiнал |
| zxy | термiнал |
| ’1’ | термiнал |
| ”1” | термiнал |

Табл. 1: Прийнята нотація РБНФ

графічного представлення граматики використовуються синтаксичні діаграми Вiрта.

## Алфавіт

Програма може містити текст з використанням таких символів (character) — літер, цифр, спеціальних знаків та ознаки кінця файлу:

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' | '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'.

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

SpecSsign = '.' | ';' | '(' | ')' | '=' | '+' | '-' | '\*' | '/' | '<' | '>' | ‘%’ | ’@’ | ‘$’

| WhiteSpace | EndOfLine.

WhiteSpace = ' ' | '\t'.

EndOfLine = '\n' | '\r' | '\r\n' | '\n\r'.

EndOfFile = '\u0000'.

# Лексика

Лексичний аналіз виконується окремим проходом, отже не залежить від синтаксичного розбору та семантичного аналізу. Лексичний аналізатор розбиває вихідний текст на лексеми. У програмі мовою Crundras можуть використовуватись лексичні елементи, що класифікуються як спеціальні символи, ідентифікатори, беззнакові цілі константи, беззнакові дійсні константи та ключові слова.

## Спецiальнi символи

Синтаксис

1. SpecSymbols = ArithOp | RelOp | BracketsOp | AssignOp | Punct

ArithOp = AddOp | MultOp

AddOp = ’+’ | ’-’

MultOp = ’\*’ | ’/’ | ‘\*\*’ | ‘%’

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

BracketsOp = ’(’ | ’)’ | ‘{’ | ‘}’

AssignOp = ’=’

Punct = ’.’ | ’;’

Опис

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

Обмеження

1. Набір токенiв див. табл. 2.

### Ідентифікатори

Синтаксис

1. Ident = Letter {Letter | Digit}.

Опис

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

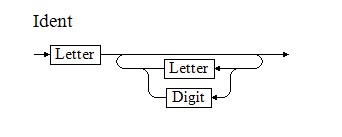
Обмеження

1. Жоден ідентифікатор не може збігатись із ключовим (вбудованим, зарезервованим) словом.

Семантика

1. Елемент, який у фазі лексичного аналізу може бути визначений як ідентифікатор, або як ключове слово, вважається ключовим словом.

Візуальне представлення



1. Синтаксична діаграма

Приклади

1. a, x1, time24, as4me17

## Літерали

Синтаксис

Literal = IntegerLiteral | FloatingLiteral.

IntegerLiteral = Digit {Digit}.

FloatingLiteral = IntegerLiteral '.' [IntegerLiteral] | '.'[IntegerLiteral].

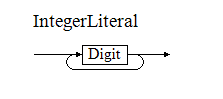
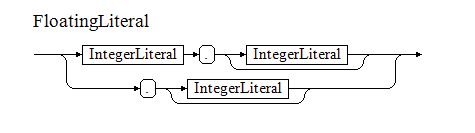
Обмеження

1. Кожен літерал повинен мати тип, а величина літералу повинна знаходитись у діапазоні репрезентативних значень для її типу.

Семантика

1. Кожен літерал має тип, визначений її формою та значенням.

Вiзуальне представлення

Приклади

1. 12, 234, 1.54, 34.567, 23. , .42

## Ключовi слова

Синтаксис

1. KeyWords = if | for | to | by | while | rof | int | float.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Код | Лексема | Токен | Приклад | Неформальний опис |
| 1 |  | id | a, x1, z12f | iдентифiкатор |
| 2 |  | int\_literal | 123, 0, 521 | цiле без знаку |
| 3 |  | float\_literal | .012, 34.76, 876. | дiйсне без знаку |
| 4 | int | keyword | integer | термiнал int |
| 5 | float | keyword | float | термiнал float |
| 6 | $ | input\_op | $ | термiнал $ |
| 7 | @ | output\_op | @ | термiнал @ |
| 8 | if | keyword | if | термiнал if |
| 9 | for | keyword | for | термiнал for |
| 10 | to | keyword | to | термiнал to |
| 11 | by | keyword | by | термiнал by |
| 12 | while | keyword | while | термiнал while |
| 13 | rof | keyword | rof | термiнал rof |
| 14 | = | assign\_op | = | термiнал = |
| 15 | + | add\_op | + | термiнал + |
| 16 | - | add\_op | - | термiнал - |
| 17 | \* | mult\_op | \* | термiнал \* |
| 18 | \*\* | mult\_op | \*\* | термiнал \*\* |
| 19 | / | mult\_op | / | термiнал / |
| 20 | % | mult\_op | % | термiнал % |
| 21 | < | rel\_op | < | термiнал < |
| 22 | > | rel\_op | > | термiнал > |
| 23 | <= | rel\_op | <= | термiнал <= |
| 24 | == | rel\_op | == | термiнал = |
| 25 | >= | rel\_op | >= | термiнал >= |
| 26 | != | rel\_op | != | термiнал != |
| 27 | ( | brackets\_op | ( | термiнал ( |
| 28 | ) | brackets\_op | ) | термiнал ) |
| 29 | { | brackets\_op | { | термiнал { |
| 30 | } | brackets\_op | } | термiнал } |
| 31 | . | punct | . | термiнал . |
| 32 | ; | punct | ; | термiнал ; |
| 33 |  | white\_space | ‘ ’,\t |  |
| 34 |  | eol | \n, \r\n |  |
| 35 |  | eof | \u0000 |  |

Табл. 2: Таблиця лексем мови Crundras

## Токени

З потоку символiв вхiдної програми на етапi лексичного аналiзу виокремлюються послiдовностi символiв з певним сукупним значенням, — *токени*. Список токенiв мови Crundras див. у табл. 2.

# Типи

Мова Crundras обробляє значення двох типів: int та float.

1. Цілий тип int може бути представлений оголошеною змінною типу int, або константою IntegerLiteral. Діапазон значень залежить від реалізації.
2. Дійсний тип float може бути представлений оголошеною змінною типу float, або константою FloatingLiteral. Діапазон значень залежить від реалізації.

# Синтаксис

## Declaration

Оголошення (декларацiї) специфiкує iнтерпретацiю та атрибути набору iдентифiкаторiв.

Синтаксис

1. DeclarationStatement = TypeSpecifier Identifier ';'.

Приклад

int a; float d34;

## Вирази

Синтаксис

1. Expression = '(' Expression ')'| PrimaryExpression | ArithmeticExpression | AssigmentExpression | UnaryExpression | RelationalExpression.

PrimaryExpression = Literal | Identifier.

UnaryExpression = [Sign] PrimaryExpression.

RelationalExpression = Expression RelationalOperator Expression.

ArithmeticExpression = Expression ArithmeticOperator Expression.

AssigmentExpression = Identifier '=' Expression.

Опис

1. Вираз - це послiдовнiсть операторiв i операндiв, що визначає порядок обчислення значення.
2. Значення, обчислене за арифметичним виразом, має тип real або integer.
3. Всi бiнарнi оператори у виразах є лiвоасоцiативними.
4. Найвищий прiоритет в унарного мiнуса та унарного плюса, далi, у порядку зменшення прiоритету слiдують MultOp, AddOp та RelOp.
5. Послiдовнiсть двох або бiльше операторiв з однаковим прiоритетом асоцiативна.

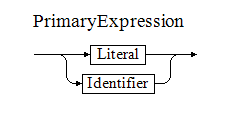
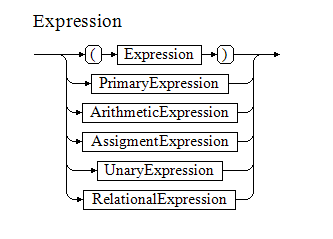
Обмеження

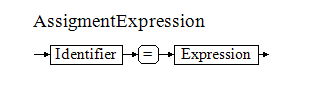
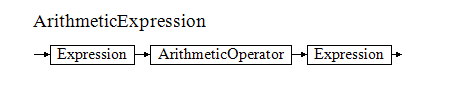
1. Використання змінної, з не визначеним на момент обчислення виразу значенням, викликає помилку.

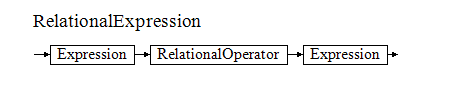
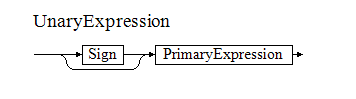
Семантика

1. Кожна константа має тип, визначений її формою та значенням.

Вiзуальне представлення







Приклади

1. a = -(f-3)\*2

## Оператори

Типом результату бінарних операторів +, -, \*, \*\* є тип обох операндів, якщо вони одного типу, інакше обирається тип float.

Типом результату бінарного оператору / завжди є float.

Типом результату бінарного оператору % завжди є int.

Тип результату унарних операторів завжди рівний типу операнду.

Тип результату операторів відношення завжди int.

### Арифметичнi оператори

### Бiнарнi

ArithmeticOperator = '+' | '-' | '\*' | "\*\*" | '/' | '%'.

### Унарнi

Sign = '+' | '-'.

4.2.2 Оператори вiдношення

RelationalOperator = '<' | '>' | "<=" | ">=" | "==" | "!=".

# Програма

Програма складається з набору виразів. Програма не може мiстити неоголошену змiнну.

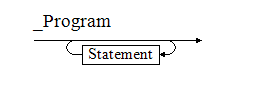


Рис. 1: Програма

# Iнструкцiї

Iнструкцiї (Statements) визначають алгоритмiчнi дiї, якi мають бути виконанi. За винятком зазначених далi випадкiв, iнструкцiї виконуються послiдовно.

## Оператор (iнструкцiя) присвоювання

Синтаксис

1. AssigmentExpression = Identifier '=' Expression.

Обмеження

1. Тип змiнної з iдентифiкатором Identifier має вiдповiдати типу виразу праворуч оператора =.

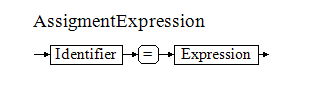
Семантика

1. Інструкція присвоювання заносить значення лівого виразу у змінну, що є лівим операндом.
2. Після виконання присвоєння, AssignmentExpression має значення лівого операнду.

Приклад

1. foo = 35

Візуальне представлення



## Iнструкцiя введення

Синтаксис

1. InputStatement = '$' Identifier ';'.

Приклад

1. $d;

## Iнструкцiя виведення

Синтаксис

1. OutputStatement = '@' Expression ';'.

Приклад

1. @42;

## Оператор вибору

Синтаксис

### SelectionStatement = "if" '('Expression')' Statement.

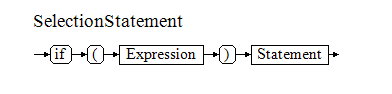
### Обмеження

### Керуючий вираз повинен бути скалярним.

### Семантика

### Якщо Expression не дорівнює 0 – виконується Statement.

Візуальне представлення



## Оператор циклу

Синтаксис

1. IterationStatement = "for" AssigmentExpression "to" Expression "by"

Expression "while" '(' Expression ')' Statement "rof" ';'.

Обмеження

1. Керуючий вираз повинен бути скалярним.

Семантика

1. Інструкція:  
   for <assignment> to <expression-1> by <expression-2> while(<expression-3>) <statement>

rof;

* 1. <assignment> виконується до будь-яких інструкцій тілу цикла.
  2. Кожного кроку вирази виконуються у такій послідовності:

<expression-3>

<expression-2>

<expression-1>

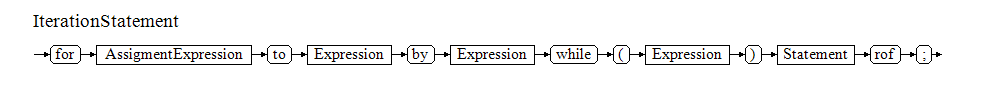
<statement>

* 1. Тіло циклу повторно виконується доки результат <expression-3> не дорівнює 0 або змінна, ініціалізована у <assignment>, та <expression-1> не рівні.

Приклад

1. for d = 15 to 73%3 by 0.34 while(d >= eps) a = d\*3-7; rof;

Візуальне представлення



# Повна граматика мови Crundras

\_Program = {Statement}.

Statement = InputStatement | OutputStatement | CompoundStatement | ExpressionStatement | SelectionStatement | IterationStatement | DeclarationStatement.

ExpressionStatement = Expression ';'.

SelectionStatement = "if" '('Expression')' Statement.

IterationStatement = "for" AssigmentExpression "to" Expression "by" Expression "while" '(' Expression ')' Statement "rof" ';'.

CompoundStatement = '{' {Statement} '}'.

DeclarationStatement = TypeSpecifier Identifier ';'.

Expression = '(' Expression ')'| PrimaryExpression | ArithmeticExpression | AssigmentExpression | UnaryExpression | RelationalExpression.

PrimaryExpression = Literal | Identifier.

UnaryExpression = [Sign] PrimaryExpression.

RelationalExpression = Expression RelationalOperator Expression.

ArithmeticExpression = Expression ArithmeticOperator Expression.

AssigmentExpression = Identifier '=' Expression.

InputStatement = '$' Identifier ';'.

OutputStatement = '@' Expression ';'.

TypeSpecifier = "int" | "float".

Identifier = Letter {Letter | Digit}.

Literal = IntegerLiteral | FloatingLiteral.

IntegerLiteral = Digit {Digit}.

FloatingLiteral = IntegerLiteral '.' [IntegerLiteral] | '.'[IntegerLiteral].

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' | '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'.

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

Sign = '+' | '-'.

ArithmeticOperator = '+' | '-' | '\*' | "\*\*" | '/' | '%'.

RelationalOperator = '<' | '>' | "<=" | ">=" | "==" | "!=".