Специфікація мови програмування Tyson

1. Вступ

Tyson - мова програмування сімейства C, варіація мови Javascript з статичною типизиацією. Tyson — мультипарадигмова мова програмування загального призначення, що підтримує імперативний і функціональний стилі програмування.

1.1 Завдання

Розробити власну мову програмування з реалізацією:

- Змінних
- Умовних операторів
- Циклів
- Виразів

1.2 Обробка

Програма, написана мовою Tyson, подається на вхід транслятора для обробки. 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 здійснюються окремими проходами.

1.3 Нотація

Для опису мови Tyson використовується розширена форма Бекуса-Наура. Ланцюжки, що починаються з великої літери вважаються нетерміналами (нетермінальними символами). Термінали — ланцюжки, що починаються з маленької літери, або знаходяться між одинарними, або подвійними лапками. Для графічного представлення граматики використовуються синтаксичні діграми Вірта.

1.4 Алфавіт

Програма може містити текст з використанням літер, цифр та спеціальних знаків:

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 | Z

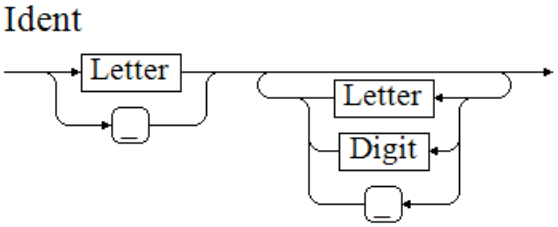
Digit = ’0’ | ’1’ | ’2’ | ’3’ | ’4’ | ’5’ | ’6’ | ’7’ | ’8’ | ’9’

Symbol = '!' | '@' | '#' | '$' | '%' | '^' | '&' | '*' | '(' | ')' | '|' | '_' | '-' | '+' | '=' | '\' | '?' | '<' | '>' | '~' | '`' | ';' | ':' | '[' | ']' | '{' | '}' .

2. Лексика

Лексичний аналіз виконується окремим проходом і не залежить від синтаксичного та семантичного аналізу. Лексичний аналізатор розбиває вихідний текст на лексеми. У програмі мовою Tyson можуть використовуватись лексичні елементи, що класифікуються як спеціальні символи, ідентифікатори, цілі числа, дійсні числа, логічні вирази та ключові слова.

2.1 Ідентифікатори



Іденифікатор – це змінна, що містить деяке значення. Першим символом ідентифікатора може бути літера або нижнє підкреслення, наступні символи, якщо вони є, можуть бути цифрами, нижнім підкресленням або літерами. Довжина ідентифікатора не обмежена.

Ідентифікатор не може збігатись з ключовим зарезервованим словом.

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

b) Елемент, який у фазі лексичного аналізу може бути визначений як ідентифікатор, або як логічна константа, вважається логічною константою.

2.2 Ключові слова

З потоку символів вхідної програми на етапі лексичного аналізу виокремлюються послідовності символів з певним сукупним значенням, — токени. Список токенів:

int	Ціле число
float	Число з плавоючою точкою
double	Float з подвійною точністю
let	Локальна змінна
var	Глобальна змінна
for	Цикл
if	Умовний оператор
while	Цикл
do	Цикл
return	Повернути
true	Логічний +
false	Логічний -
const	Константна змінна
string	Символьна строка
boolean	Логічний тип
break	Кінець циклу
continue	Перехід на наступну ітерацію
null	Нема значення
undefined	Невизначено

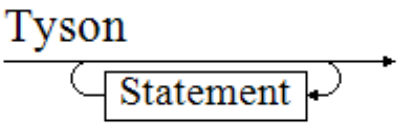
2.3 Типи

Мова Tyson розпізнає 9 базових типів: var, let, int, float, double, string, double, boolean та два додаткових null та undefined.

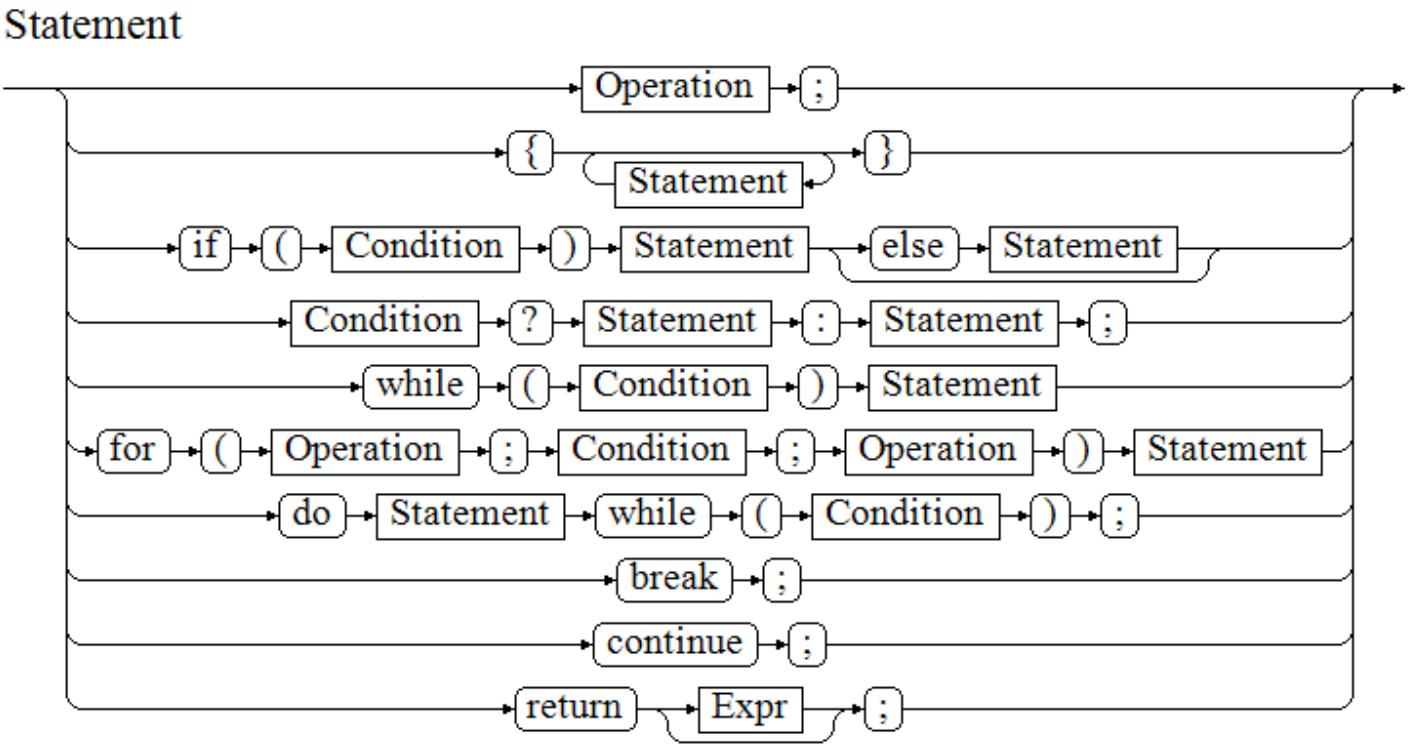
1. var – глобальная динамічна змінна, видима в межах програми.
2. let – локальна динамічна змінна, видима в межах блоку.
3. int – ціле число.
4. float – дійсне число.
5. double – дійсне число подвійноі точності.
6. string – строковий тип.
7. boolean - логічний тип true або false.
8. null – значення відсутнє.
9. undefined – значення не визначено.

3. Мовні конструкції

Программа складається з нуля або більше виразів:



3.1.Вираз

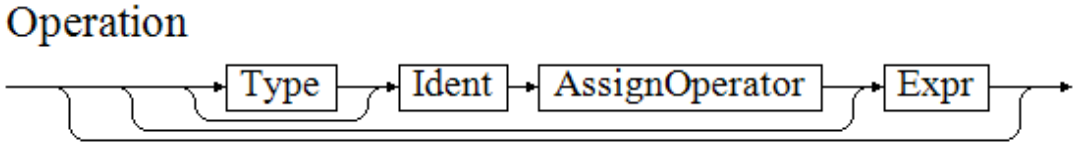


Вираз – це операція присвоювання, цикл, умова або повернення результату через ключове слово return.

Приклад:

```
int n = 1;
do {
  n++;
} while(n < 3);
{
   const _var =  “this is block statement”
}
```

3.2.Операція



Операція складається з правої частини Expr, що повертає якесь значення, і опціональної лівої частини, що це значення приймає.

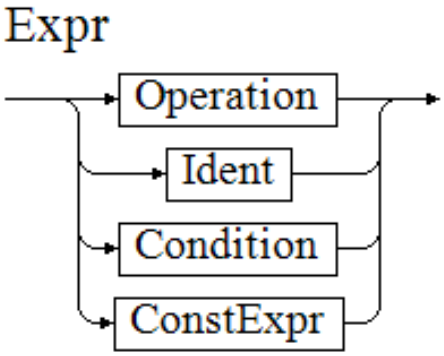
Приклад:

int n = 1

variable

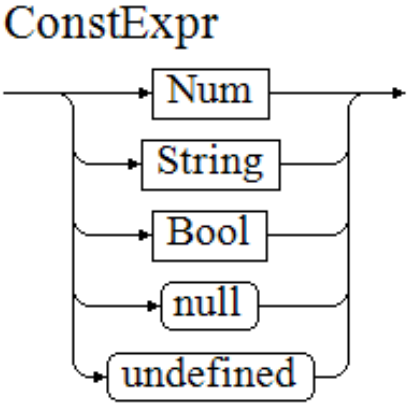
const var temp = variable

3.3 Expression

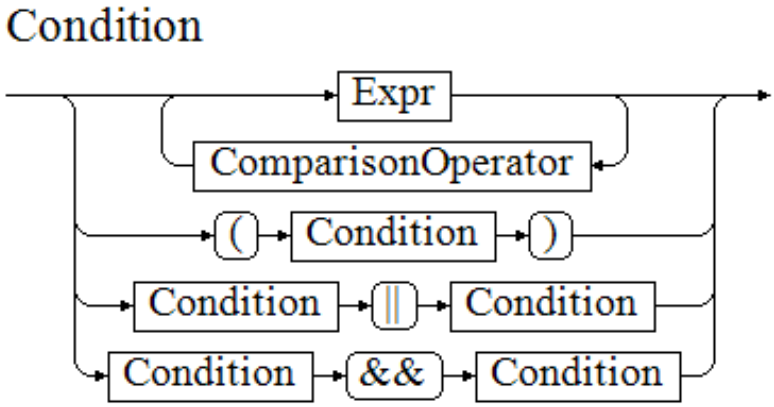


Expression – це будь-що, що повертає значення: змінна, константа, умовний оператор.

3.4 Константний вираз



Const Expression – число, символьна строка, логічне значення, null або undefined.

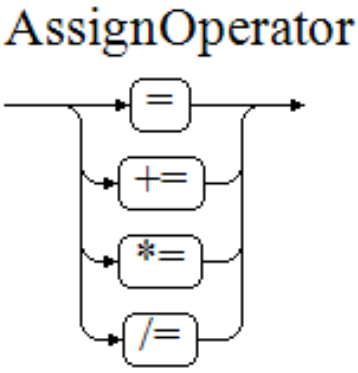


3.5 Умова

Condition – це логічний вираз, що повертає true або false. Він може складатись з багатьох логічних виразів, що складаються між собою завдяки операторами логічного І '||' та логічного АБО '&&'. Логічні вирази можна об'єднувати завдяки круглим дужкам.

4 Оператори

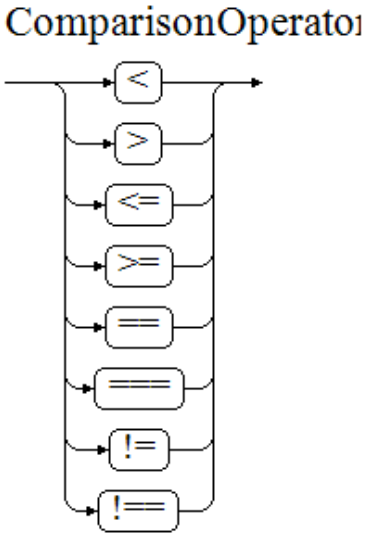
4.1 Оператори присвоння



'=' присвоює значення справа змінній зліва

'var +=' те ж саме, що 'var = var +'

4.2 Оператори порівняння



‘<’ – менше

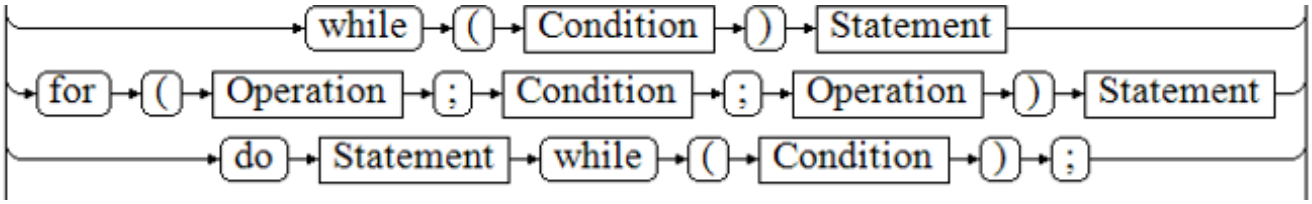
‘>’ - більше

‘<=’ – менше або дорівнює

‘>=’ – більше або дорівнює

‘==’ – нестроге дорівнює
‘===’ – строге дорівнює
‘!=’ – нестроге не дорівнює
‘!==’ – строге не дорівнює

4.3 Оператори циклу



Цикл виконує заданий набір операцій, доки виконується перевірочна умова.

5 Повна граматика мови Tyson

```
Tyson = { Statement }.

Statement   =   Operation ';'
|         "{" {Statement} "}"
|         "if" "(" Condition ")" Statement  ["else" Statement]
|          Condition "?" Statement ":" Statement ";"
|         "while" "(" Condition ")" Statement
|         "for" "(" Operation ";" Condition ";" Operation ")" Statement
|         "do" Statement "while" "(" Condition ")" ";"
|         "break" ";"
|         "continue" ";"
|         "return" [Expr] ";" .

Operation = [[[Type] Ident AssignOperator] Expr].

Ident = (Letter | '_') {Letter | Digit | '_'} .

Type = [const] (Primitive) .
Primitive =  "var"
| "let"
| "int"
| "float"
| "double"
| "string"
| "boolean" .

AssignOperator =  '=' | '+=' | '*=' | '/=' .
ComparisonOperator = '<' | '>' | '<=' | '>=' | '==' | '===' | '!=' | '!==' .


Function    =     function Ident '(' { Param } ')' Statement.
Param       =     Type Ident ['=' ConstExpr] .
Expr        =     Operation | Ident | Condition | ConstExpr .

Condition        =          Expr {ComparisonOperator Expr}
| '(' Condition ')'
| Condition '||' Condition
| Condition '&&' Condition .


ConstExpr       =        Num | String | "null" | "undefined".

Num = [Sign] (Float | Int) .
Float = Digit {Digit} '.' {Digit} .
```

Int = Digit {Digit} .

Sign = '+' | '-' .

String = "" { Digit | Letter | Symbol } ""
| """ { Digit | Letter | Symbol } """.

Letter = (LetterSmall | LetterBig) .
LetterSmall = 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 .
LetterBig = 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' .
Symbol = '!' | '@' | '#' | '$' | '%' | '^' | '&'
| '*' | '(' | ')' | '|' | '_' | '-' | '+'
| '=' | '\' | '?' | '<' | '>' | '~' | '`'
| ';' | ':' | '[' | ']' | '{' | '}' .

WhiteSpace = (Blank|Tab) {Blank|Tab} .
Blank = " " .
Tab = '\t' .
Newline = '\n' .