**Formální jazyky a překladače**

2016/2017

**Dokumentácia interpretu imperativního jazyka IFJ16**

**Tým 013, varianta b/3/I**

Rozšírenia: BOOLOP

**Zoznam autorov:**

Šuba Adam xsubaa00 25% vedúci tímu

Tóth Adrián xtotha01 25%

Šuhaj Peter xsuhaj02 25%

Paliesek Jakub xpalie00 25%

**Obsah**

[1. Úvod 2](#_Toc468466936)

[2. Interpret 3](#_Toc468466937)

[2.1. Lexikálny analyzátor 3](#_Toc468466938)

[2.2. Syntaktický analyzator 3](#_Toc468466939)

[2.3. Sémantický analyzátor 3](#_Toc468466940)

[2.4. Interpret 3](#_Toc468466941)

[3. Algoritmy 4](#_Toc468466942)

[3.1. (b) vyhľadávanie s použitím Boyer-Moorovho algoritmu 4](#_Toc468466943)

[3.2. (3) radenie pomocou Shell sort algoritmu 4](#_Toc468466944)

[3.3. (I) implementácia tabuľky symbolov pomocou binárneho vyhľadávacieho stromu 4](#_Toc468466945)

[4. Práca v tíme 4](#_Toc468466946)

[5. Záver 4](#_Toc468466947)

[6. Literatúra 4](#_Toc468466948)

[7. Príloha 5](#_Toc468466949)

[7.1. Diagram konečného automatu lexikálnej analýzy 5](#_Toc468466950)

[7.2. LL-gramatika syntaktického analyzátora 6](#_Toc468466951)

[7.3. LL-tabuľka syntaktického analyzátora 7](#_Toc468466952)

[7.4. Precedenčná tabuľka syntaktického analyzátora 8](#_Toc468466953)

# Úvod

Dokumentácia popisuje vývoj interpretu imperatívneho jazyka IFJ16. Jazyk IFJ16 je podmnožina jazyka Java SE 8, čo je staticky typovaný objektovo orientovaný jazyk.

Úlohou nášho interpretu je kontrola zdrojového kódu napísaného v jazyku IFJ16 a následne jeho interpretácia.

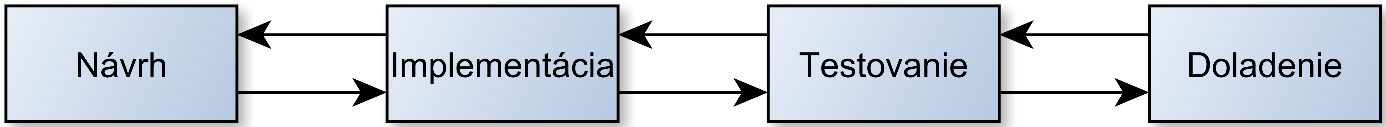
Nami zvolený Tím 013 mal priradenú variantu b/3/I. Varianta obsahovala: (b) vyhľadávanie s použitím Boyer-Moorovho algoritmu, (3) radenie pomocou Shell sort algoritmu, (I) implementácia tabuľky symbolov pomocou binárneho vyhľadávacieho stromu.

Dokumentácia je rozdelená na kapitoly, ktoré bližšie popisujú jednotlivé časti interpretu a spôsob ich riešenia. Taktiež obsahuje aj popis algoritmov, popis rozdelenia úloh a spôsobu práce v tímu. V závere dokumentácie sa nachádza zhrnutie našej práce.

V prílohe sa nachádzajú: diagram konečného automatu lexikálnej analýzy, LL-gramatika a precedenčná tabuľka syntaktického analyzátora.

# Návrh

## Vývojový cyklus



xxxxxxxxxxxxxxx

# Interpret

## Lexikálny analyzátor

Lexikálny analyzátor, nazývaný aj ako „scanner“, je prvá časť interpretu. Lexikálny analyzátor je implementovaný je vo forme konečného automatu. Vstupom je textový súbor z ktorého sa čítajú jednotlivé znaky ktoré sú vyhodnocované.

Konečný automat prevádza znaky resp. postupnosť znakov t.j. lexémy do tokenov. Znaky typu medzera, komentáre sú ignorované. Lexikálny analyzátor má špecifikované určité znaky ktoré môže prijímať. Ak znak nepatrí medzi prijímané znaky, je vyhlásená lexikálna chyba. Pri chybnej štruktúre aktuálneho lexému je program ukončený s návratovou hodnotou 1.

Lexikálny analyzátor sa využíva keď syntaktická analýza žiada o token. Token je štruktúra ktorá obsahuje typ tokenu a typu zodpovedajúce dáta. Token sa z lexikálne analýzy posiela syntaktickej analýze práve vtedy keď konečný automat skončil v koncovom stave. Konečný automat sa spúšťa zakaždým keď je žiadaný token.

## Syntaktický analyzator

Druhou časťou interpretu je syntaktická analýza „parser“. Úlohou syntaktického analyzátora je kontrola reťazcov tokenov či je syntakticky správne napísaný program. Ak k danému reťazci tokenov je úspešne vytvorený derivačný strom, program sa považuje za syntakticky správny. Niektoré časti syntaktickej kontroly sú spúšťané na základe sémantických akcii (xxxxxxxxxxxx).

Pri chybe je vyhlásená syntaktická chyba. Pri chybnej syntaxi programu je program ukončený s návratovou hodnotou 2.

## Sémantický analyzátor

Ďalšou časťou interpretu je sémantická analýza. Vstupom je derivačný strom od syntaktického analyzátora. Sémantický analyzátor skontroluje sémantické aspekty ktorými sú kontrola typov a kontrola deklarácii premenných stručne povedané správnosť operácii. Výstupom sémantického analyzátora je abstraktný syntaktický strom. Z abstraktného syntaktického stromu sú vytvorené 3-adresné inštrukcie ktoré sú potrebné pre samotný interpret.

Ak nastala chyba, je vyhlásená sémantická chyba a program je ukončený s návratovou hodnotou 3,4,6 (hodnota závisí od typu chyby).

## Interpret

Poslednou fázou je interpret. Interpret odsimuluje vstupnú inštrukčnú pásku ktorá sa skladá z trojadresných inštrukcii a na základe nich vykoná interpretáciu. Interpret má na starosti hlavne typovú konverziu dátových typov pri operáciách. Napríklad pri súčte int a double musí vykonať konverziu int na double.

# Algoritmy

Všetky nižšie popísané algoritmy sú implementované v súbore ial.c.

## (b) vyhľadávanie s použitím Boyer-Moorovho algoritmu

Algoritmus Boyera-Moora bol objavený Robertem S. Boyerem and J. Strotherem Moorem v roku 1977. Boyer-Moorov algoritmus je ten najefektívnejší algoritmus na vyhľadane v reťazci podreťazec. Funguje na základe dvoch heuristík a na princípe spracovania reťazca zprava doľava. Podreťazec vyhodnocuje od posledného znaku k prvému znaku. Heuristické funkcie spracujú podreťazec najskôr do poľa kam sa uložia hodnoty skokov.

## (3) radenie pomocou Shell sort algoritmu

Shell sort bol objavený Donaldom Shellom v roku 1959. Je to kvadratický triediaci algoritmus ktorého zložitosť je O(n^2) a slúži na usporiadanie postupnosti prvkov do poradia. Algoritmus pracuje „in situ“ a na základe vkladania.

## (I) implementácia tabuľky symbolov pomocou binárneho vyhľadávacieho stromu

Binárny vyhľadávací strom sa využíva pri tabuľke symbolov. Implementovaný bol vo forme rekurzie a s priechodom postorder. V strome sú uložené názvy funkcii a im zodpovedajúce dáta.

# Práca v tíme

Určili sme si kto má na starosti akú časť projektu.

Adam Šuba – lexikálny analyzátor, syntaktický analyzátor, správa tímu

Peter Šuhaj – interpret, algoritmy

Adrián Tóth – testy, algoritmy, dokumentácia

Jakub Paliesek – syntaktický analyzátor, sémantický analyzátor, algoritmy, generátor 3-adresného kódu

Systém pre správu verzii sme si zvolili GIT kde sme si vytvorili vlastný privátny repozitár. Často sme sa schádzali prediskutovať rôzne implementačné detaily a zmeny v zdrojových kódoch ktoré sme museli uskutočniť.

Práca prebiehala rýchlo a efektívne a tak sme mali plne funkčný interpret s menšími implementačnými chybami pripravený na prvé pokusné odovzdanie.

# Záver

Zmyslom tohto projektu bolo porozumenie práci interpretu a na získanie skúseností v tímovej práce.

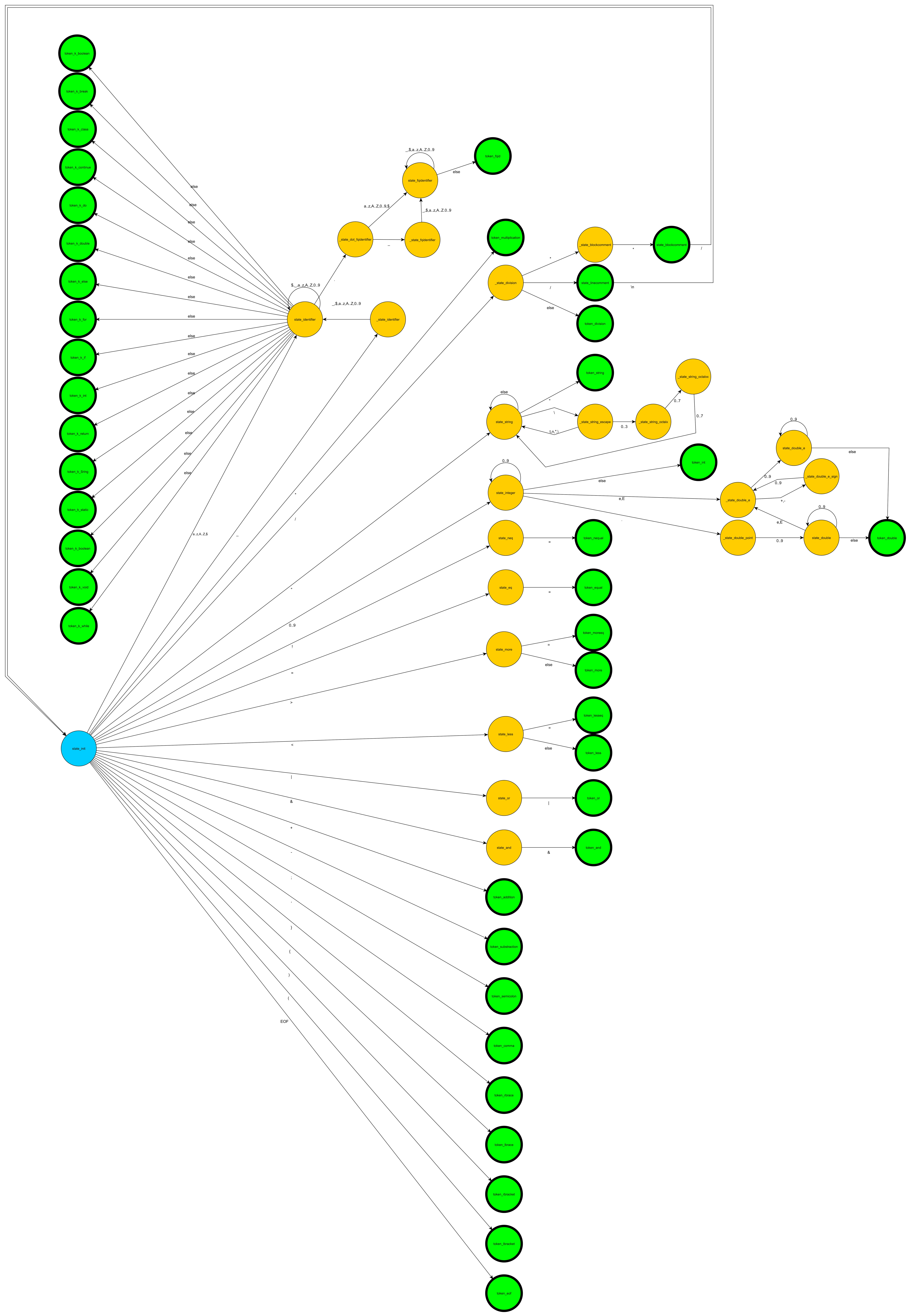
Výsledkom našej tímovej práce je interpret imperatívneho jazyka IFJ16 doplnený o boolop rozšírenie.

# Literatúra

1. HONZÍK, Jan M., Algoritmy: Studijní opora. Verzia: 16-B, Brno: Vysoké učení Technické, 2014
2. https://sk.wikipedia.org/wiki/Shell\_sort

# Príloha

## Diagram konečného automatu lexikálnej analýzy



## LL-gramatika syntaktického analyzátora

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Rule** | **Empty** | **First** | **Follow** | **Predict** |
| C-LIST → **class id {** MEMB-LIST **}** C-LIST | 0 | class, $ | $ | class |
| C-LIST → **$** | $ |
| MEMB-LIST → C-MEMB MEMB-LIST | 1 | static | } | static |
| MEMB-LIST → ε | } |
| C-MEMB → **static** C-MEMB1 | 0 | static | static, } | static |
| C-MEMB1 → TYPE **id** C-MEMB2 | 0 | void, int, double, String, boolean | static, } | int, double, String, boolean |
| C-MEMB1 → **void id** C-MEMBFUNC | void |
| TYPE → **int** | 0 | int, double, String, boolean | id | int |
| TYPE → **double** | double |
| TYPE → **String** | String |
| TYPE **→ boolean** | boolean |
| C-MEMBFUNC → **(** FN-DEF-PLIST **) {** FN-BODY **}** | 0 | ( | static, } | ( |
| C-MEMB2 → **=** E**;** | 0 | ;, =, ( | static, } | = |
| C-MEMB2 → **;** | ; |
| C-MEMB2 **→** C-MEMBFUNC | ( |
| FN-DEF-PLIST → ε | 1 | int, double, String, boolean | ) | ) |
| FN-DEF-PLIST → PAR-DEF FN-DEF-PLIST1 | int, double, String, boolean |
| FN-DEF-PLIST1 → **,** PAR-DEF FN-DEF-PLIST1 | 1 | "," | ) | , |
| FN-DEF-PLIST1 → ε | ) |
| PAR-DEF → TYPE **id** | 0 | int, double, String, boolean | ",", ) | int, double, String, boolean |
| FN-BODY → STAT FN-BODY | 1 | id, fqid, {, if, while, return, int, double, String, boolean | } | id, fqid, {, if, while, return |
| FN-BODY → TYPE **id** OPT-ASSIGN **;** FN-BODY | int, double, String, boolean |
| FN-BODY → ε | } |
| STAT-COM → **{** STAT-LIST **}** | 0 | { | else, id, fqid, {, if, while, return,  int, double, String, boolean, } | { |
| STAT-LIST → STAT STAT-LIST | 1 | id, fqid, {, if, while, return | } | id, fqid, {, if, while, return |
| STAT-LIST → ε | } |
| ID → **id** | 0 | id, fqid | (, =, ), “,” | id |
| ID → **fqid** | fqid |
| STAT → IDAS-CA **;** | 0 | id, fqid, {, if, while, return | fqid, id, {, if, while, return, int,  double, String, boolean, } | id, fqid |
| STAT **→** STAT-COM | { |
| STAT → **if** **(** E **)** STAT-COM **else** STAT-COM | if |
| STAT → **while** **(** E **)** STAT-COM | while |
| STAT **→ return** RET-VAL **;** | return |
| AS-CA → **(** FN-PLIST **)** | 0 | (, = | ; | ( |
| AS-CA → **=** ASSIGN | = |
| ASSIGN → E | 0 | E, id, fqid | ; | E |
| ASSIGN → ID **(** FN-PLIST **)** | id, fqid |
| OPT-ASSIGN → ε | 1 | = | ; | ; |
| OPT-ASSIGN → **=** ASSIGN | = |
| FN-PLIST → ε | 1 | id, fqid, string-literal, int-literal,  double-literal,  boolean-literal | ) | ) |
| FN-PLIST → VAL-ID FN-PLIST1 | id, fqid, string-literal,  int-literal, double-literal, boolean-literal |
| FN-PLIST1 → ε | 1 | "," | ) | ) |
| FN-PLIST1 → **,** VAL-ID FN-PLIST1 | , |
| VAL-ID → ID | 0 | id, fqid, string-literal, int-literal,  double-literal,  boolean-literal | ",", ) | id, fqid |
| VAL-ID → **string-literal** | string-literal |
| VAL-ID **→ int-literal** | int-literal |
| VAL-ID **→ double-literal** | double-litera |
| VAL-ID **→ boolean-literal** | boolean-literal |
| RET-VAL → E | 1 | E | ; | E |
| RET-VAL → ε | ; |

## LL-tabuľka syntaktického analyzátora

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **;** | **,** | **void** | **id** | **fqid** | **static** | **int** | **double** | **String** | **boolean** | **E** | **(** | **)** | **{** | **}** | **int-literal** | **double-literal** | **String-literal** | **boolean-literal** | **=** | **if** | **else** | **while** | **return** | **$** | **class** |
| **C-LIST** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 2 | 1 |
| **C-MEMB** |  |  |  |  |  | 5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **C-MEMB1** |  |  | 7 |  |  |  | 6 | 6 | 6 | 6 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **MEMB-LIST** |  |  |  |  |  | 3 |  |  |  |  |  |  |  |  | 4 |  |  |  |  |  |  |  |  |  |  |  |
| **TYPE** |  |  |  |  |  |  | 8 | 9 | 10 | 11 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **C-MEMB-FUNC** |  |  |  |  |  |  |  |  |  |  |  | 12 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **C-MEMB2** | 14 |  |  |  |  |  |  |  |  |  |  | 15 |  |  |  |  |  |  |  | 13 |  |  |  |  |  |  |
| **FN-DEF-PLIST** |  |  |  |  |  |  | 17 | 17 | 17 | 17 |  |  | 16 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **FN-DEF-PLIST1** |  | 18 |  |  |  |  |  |  |  |  |  |  | 19 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **PAR-DEF** |  |  |  |  |  |  | 20 | 20 | 20 | 20 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **FN-BODY** |  |  |  | 21 | 21 |  | 22 | 22 | 22 | 22 |  |  |  | 21 | 23 |  |  |  |  |  | 21 |  | 21 | 21 |  |  |
| **STAT-COM** |  |  |  |  |  |  |  |  |  |  |  |  |  | 24 |  |  |  |  |  |  |  |  |  |  |  |  |
| **STAT-LIST** |  |  |  | 25 | 25 |  |  |  |  |  |  |  |  | 25 | 26 |  |  |  |  |  | 25 |  | 25 | 25 |  |  |
| **ID** |  |  |  | 27 | 28 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **STAT** |  |  |  | 30 | 30 |  |  |  |  |  |  |  |  | 31 |  |  |  |  |  |  | 32 |  | 33 | 34 |  |  |
| **AS-CA** |  |  |  |  |  |  |  |  |  |  |  | 35 |  |  |  |  |  |  |  | 36 |  |  |  |  |  |  |
| **OPT-ASSIGN** | 39 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 40 |  |  |  |  |  |  |
| **ASSIGN** |  |  |  | 38 | 38 |  |  |  |  |  | 37 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **RET-VAL** | 51 |  |  |  |  |  |  |  |  |  | 50 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **FN-PLIST** |  |  |  | 42 | 42 |  |  |  |  |  |  |  | 41 |  |  | 42 | 42 | 42 | 42 |  |  |  |  |  |  |  |
| **FN-PLIST1** |  | 44 |  |  |  |  |  |  |  |  |  |  | 43 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **VAL-ID** |  |  |  | 45 | 45 |  |  |  |  |  |  |  |  |  |  | 47 | 48 | 46 | 49 |  |  |  |  |  |  |  |

## Precedenčná tabuľka syntaktického analyzátora

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **+** | **-** | **\*** | **/** | **<** | **>** | **<=** | **>=** | **==** | **!=** | **(** | **)** | **&&** | **||** | **!** | **id** | **$** |
| **+** | **>** | **>** | **<** | **<** | **>** | **>** | **>** | **>** | **>** | **>** | **<** | **>** | **>** | **>** | **<** | **<** | **>** |
| **-** | **>** | **>** | **<** | **<** | **>** | **>** | **>** | **>** | **>** | **>** | **<** | **>** | **>** | **>** | **<** | **<** | **>** |
| **\*** | **>** | **>** | **>** | **>** | **>** | **>** | **>** | **>** | **>** | **>** | **<** | **>** | **>** | **>** | **<** | **<** | **>** |
| **/** | **>** | **>** | **>** | **>** | **>** | **>** | **>** | **>** | **>** | **>** | **<** | **>** | **>** | **>** | **<** | **<** | **>** |
| **<** | **<** | **<** | **<** | **<** |  |  |  |  | **>** | **>** | **<** | **>** | **>** | **>** | **<** | **<** | **>** |
| **>** | **<** | **<** | **<** | **<** |  |  |  |  | **>** | **>** | **<** | **>** | **>** | **>** | **<** | **<** | **>** |
| **<=** | **<** | **<** | **<** | **<** |  |  |  |  | **>** | **>** | **<** | **>** | **>** | **>** | **<** | **<** | **>** |
| **>=** | **<** | **<** | **<** | **<** |  |  |  |  | **>** | **>** | **<** | **>** | **>** | **>** | **<** | **<** | **>** |
| **==** | **<** | **<** | **<** | **<** | **<** | **<** | **<** | **<** | **>** | **>** | **<** | **>** | **>** | **>** | **<** | **<** | **>** |
| **!=** | **<** | **<** | **<** | **<** | **<** | **<** | **<** | **<** | **>** | **>** | **<** | **>** | **>** | **>** | **<** | **<** | **>** |
| **(** | **<** | **<** | **<** | **<** | **<** | **<** | **<** | **<** | **<** | **<** | **<** | **=** | **<** | **<** | **<** | **<** |  |
| **)** | **>** | **>** | **>** | **>** | **>** | **>** | **>** | **>** | **>** | **>** |  | **>** | **>** | **>** | **>** |  | **>** |
| **&&** | **<** | **<** | **<** | **<** | **<** | **<** | **<** | **<** | **<** | **<** | **<** | **>** | **>** | **>** | **<** | **<** | **>** |
| **||** | **<** | **<** | **<** | **<** | **<** | **<** | **<** | **<** | **<** | **<** | **<** | **>** | **<** | **>** | **<** | **<** | **>** |
| **!** | **>** | **>** | **>** | **>** | **>** | **>** | **>** | **>** | **>** | **>** | **<** | **>** | **>** | **>** | **<** | **<** | **>** |
| **id** | **>** | **>** | **>** | **>** | **>** | **>** | **>** | **>** | **>** | **>** |  | **>** | **>** | **>** | **>** |  | **>** |
| **$** | **<** | **<** | **<** | **<** | **<** | **<** | **<** | **<** | **<** | **<** | **<** |  | **<** | **<** | **<** | **<** |  |