
DOKUMENTÁCIA K PROJEKTU Z PREDMETOV IFJ A IAL

# IMPLEMENTÁCIA PREKLADAČA IMPERATÍVNEHO JAZYKA IFJ19

TÍM 36, VARIANTA 02

Vedúci: Žitňanský Adam, xzitna02 (30%)

Otčenáš Matej, xotcen01 (30%)

Dúdik Samuel, xdudik01 (30%)

Žitňanská Lívia, xzitna03 (10%)

# Obsah:

# 1. Úvod:

Dokumentácia zaznamenáva postup pri riešení projektu z predmetov IFJ a IAL, ktorého úlohou je načítať zdrojový kód Ifj2019 a generovať výsledný medzikód v IFJcode2019 aj s návratovou hodnotou, kde jazyk Ifj2019 je podmnožina jazyka Python $3^2$ . Zameriava sa na návrh riešenia, implementáciu, prácu v tíme a zahrnutie. Okrem toho obsahuje návrh konečného automatu a tabuľky ako prílohy.

# 2. Návrh riešenia:

Táto kapitola popisuje návrh riešenia, časti prekladača a predávanie informácií medzi nimi rovnako ako náš prístup a spôsob riešenia.

## 2.1. Lexikálna analýza:

Lexikálna analýza bola prvá časť, na ktorej sa začalo pracovať. Jej implementácia sa nachádza v súbore **scanner.s** s hlavičkovým súborom **scanner.h**. a vychádza z návrhu konečného automatu (viz 7.1. ). Skener číta vstupný súbor po znakoch a vytvorí z nich takzvané tokeny. Tieto tokeny prechádzajú funkciou *read_toke()n,* v ktorej sa v nekonečnom cykle while vyhodnotí, či daný token patrí medzi operátory, identifikátory, kľúčové slová, názvy premenných či funkcií, alebo predstavuje číselnú hodnotu. Na základe tohto vyhodnotenia sa tok riadenia posúva do príslušného stavu, vykonajú sa požadované inštrukcie, vráti sa spracovaný token a presunie sa na ďalší, pre ktorý celý proces zopakuje. Spracované tokeny si vyžiada Parser na vykonanie syntaktickej analýzy. Dôležitá pre Skener je dátová štruktúra zásobník implementovaná v súboroch **stack.c** a **stack.h**, ktorá sa využíva hlavne pri rátaní počtu odsadení a generovaní príslušného tokenu pre ne, a knižnica pre dynamický reťazec v súboroch **string_lib.c** a **string_lib.h**, ktorá slúži na prácu s reťazcami, hlavne pri názvoch premenných a funkcií.

## 2.2. Syntaktická analýza:

Syntaktická analýza sa riadi pravidlami LL gramatiky (viz 7.2. ) a je reprezentovaná súbormi **parser.c** a **expr_parser.c** s rovnomennými hlavičkovými súbormi. V hlavnom tele Parseru, *start_compiler()*, sa zavolá funkcia pre inicializáciu Parsera a tabuľky pre lokálne a globálne symboly, potom sa vráti spracovaný príkaz cez funkciu *statement()*. V tejto funkcii Parser žiada Skener o tokeny, zostavuje „strom programu“ a na základe pravidiel LL gramatiky ho redukuje. Parser je implementovaný pomocou odporúčanej metódy rekurzívneho zostupu. Pre prácu s tokenmi sa zaviedol typ zodpovedajúci symbolom, čo uľahčilo prevod tokenu na symbol. Niektoré si automaticky zodpovedali, iné bolo treba manuálne konvertovať, na čo sa použil podmienený príkaz *if*. Okrem toho sme sa rozhodli v Parseri implementovať pomocné makrá pre sprehľadnenie a zefektívnenie kódu.

### 2.2.1. Vyhodnocovanie výrazov

Na vyhodnocovanie výrazov pre Parser slúži Výrazový parser **expr_parser.c**, ktorý je Parserom zavolaný v prípade, že sa za premennou nevyskytuje ľavá zátvorka – teda sa na vstupe očakáva výraz, nie funkcia. V hlavnom tele *expression()* sa po inicializácií Výrazového parseru v nekonečnom cykle while spracovávajú tokeny prislúchajúce výrazu. Keďže pre určenie, či sa jedná a funkciu alebo výraz sa v Parseri musia načítať dva tokeny, Výrazovému parseru sú tiež predané dva, miesto jedného. Na základe precedentnej tabuľky, ktorá je uložená v matici 16x16 (viz príloha.č.4 )., definuje Výrazový parser prednosť jednotlivých symbolov a shiftuje či redukuje daný symbol.

## 2.3. Sémantická analýza:

Sémantická analýza je úzko zviazaná so Syntaktickou analýzou a s generovaním kódu. Je reprezentovaná tabuľkou symbolov **symtable.c** , ktorá sa používa v dvoch verziách – lokálnej a globálnej tabuľke symbolov. Do globálnej tabuľky sa ukladajú funkcie, pretože musia byť kedykoľvek k dispozícii. Do lokálnej tabuľky sa ukladajú parametre a premenné. Keďže je Ifj2019 dynamicky orientovaný jazyk, chyby sémantickej analýzy sa vyhodnocujú nielen v Parsery, ale aj priamo počas generovania kódu.

# 3. Implementácia

## 3.1. Tabulka symbolov

Implementácia tabuľky symbolov sa nachádza v súbore **symtable.c**. Inicializuje sa spolu s Parserom a to až dvakrát – ako tabuľka lokálnych a ako tabuľka globálnych symbolov. Po inicializácii sa do tabuľky pridajú vstavané funkcie ako *inputs, print* a iné. Ako alokačnú konštantu pre veľkosť tabuľky sme zvolili prvočíslo, aby sme sa vyhli zoskupovaniu hodnôt a zabezpečili tak rovnomernejšie distribuovanú tabuľku. Hashovaciu funkciu sme zvolili pre jej rýchlosť a malé množstvo kolízii, čo zvyšuje jej efektivitu.

## 3.2. Generovanie kódu

V súbore **generator.c** sa nachádza popis generovania kódu. Pozostáva z deklarácií pomocných funkcií pre medzikód a fuknkcií pre zásobník interpretu. Vzhľadom na to, že niektoré operácie nepodporujú operandy iných typov, generátor musí prekonvertovať typy na rovnaké, keďže je výsledný kód dynamicky typovaný. Okrem toho obsahuje generovanie pre funkcie a vstavané funkcie na uľahčovanie práce s Parserom. Generátor kódu musí zariadiť aj konverziu medzier a odsadení na Ascii znaky v reťazcoch, pretože by mohli vytvárať problémy pre inštrukčnú sadu. Priame volanie jeho funkcii sa nachádza v Parseri, ktorý mu dodá všetky potrebné vstupné parametre.

## 3.3. Zásobník

Popri inicializácii Skeneru sa súčasne vytvára aj dátová štruktúra zásobník a spoločne so Skenerom je aj uvoľnený. Jeho hlavná úloha je pri lexikálnej analýze, a to na kontrolu počtu odsadení a jeho zmeny, aby sme správne generovali tokeny pre indent a dedent a tým poslali syntaktickej analýze správu o zanoreniach príkazov. Podstatnú rolu však táto dátová štruktúra zohráva aj pri spracovaní výrazov. Z tohto dôvodu bolo nutné, aby bol schopný podporovať viacero dátových typov - na vyjadrenie veľkosti odsadenia podporuje dátový typ integer a pri spracovaní výrazov pre Výrazový parser zas štruktúru symbol. Ďalšou dôležitou úlohou zásobníku je zaznamenávanie počtu zanorení pri vnorených blokoch (statement inside) príkazov ako *if* a *while*.

# 4. Práca v tíme:

## 4.1. Komunikácia:

Ako komunikačný kanál sme zvolili Discord pre jeho dostupnosť. Preferovali sme osobné stretnutia a skupinovú prácu - členovia tímu sa stretli, na mieste sa rozdelili úlohy a všetci spoločne pracovali. Takýto prístup povzbudzoval pracovnú morálku a mal za výsledok, že všetci členovia tímu bol aspoň čiastočne zapojený pri vývoji každej časti prekladača a jednoduchšie sa mohli konzultovať návrhy a prípadne si objasniť spôsob riešenia.

## 4.2. Verzovací systém:

Vzhľadom na predošlé skúsenosti sme sa rozhodli pre prácu s Githubom, kvôli jeho prehľadnosti. Repozitár bol založený začiatkom septembra a každý člen tímu postupne pridával svoju prácu a prípadné úpravy do vetiev, ktoré sa po istej dobe alebo určitom počte úprav spájali v hlavnú.

## 4.3. Rozdelenie úloh:

Žitňanský Adam – lexikálna analýza, testy, zásobník, spracovanie výrazov
Otčenáš Matej – syntaktická analýza, tabuľka symbolov
Dúdik Samuel - generovanie kódu, testy
Žitňanská Lívia – dokumentácia, testy

Členovia sa súčasne podieľali na viacerých častiach implementácie, počas osobných stretnutí sa konzultovali idei, podávali návrhy na riešenie a rady. Rozdiely v hodnotí sú v množstve práce vykonanej jednotlivými členmi.

# 5. Záver

Projekt hodnotíme veľmi kladne. Keďže náš tím bol rovnako zložený aj na predmete IVS, bolo jednoduchšie nadviazať na predošlý systém práce, využiť nadobudnuté skúsenosti a vylepšiť predošlé nedostatky. Prípadné problémy a nejasnosti zadania sme riešili osobne vzájomnou diskusiou, hľadali v materiáloch z prednášok, prípadne sme nahliadli do fóra k projektu. Pracovať sme začali dostatočne skoro, takže sme stihli projekt dokončiť do pokusného odovzdania a vďaka nemu a priebežnému testovaniu sme zvládli dodatočne vylepšiť naše riešenie. Okrem komunikácie a spoločnej práce, sme si zlepšili svoje schopnosti a počas implementácie sme aktívne využívali vedomosti z predmetov IFJ a IAL.

# 6. LITERATÚRA:

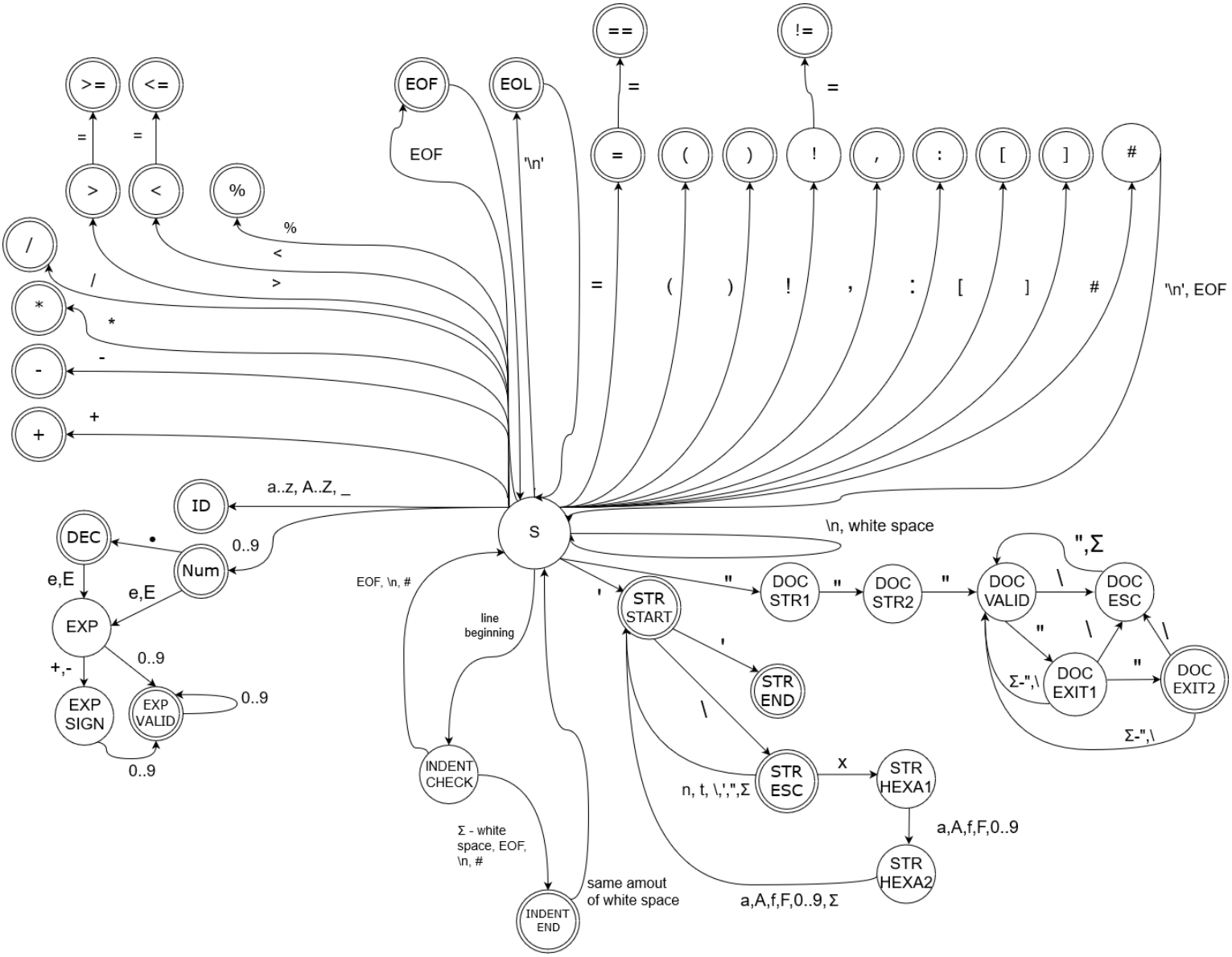
**[1]** Materiály k predmetu IFJ

**[2]** Študijná opora predmetu IAL

**[3]** Wikipédia

# 7. Prílohy:

## 7.1. Konečný automat lexikálneho analyzátoru



## 7.2. LL Gramatika

1. <statement> -> EOF

2. <statement> -> EOL <statement>

3. <statement> -> DEF ID ( <params> ): EOL INDENT <statement_inside> <end> DEDENT <statement>

4. <statement> -> IF <expression>: EOL INDENT <statement_inside> EOL DEDENT ELSE: EOL INDENT <statement_inside> <end> DEDENT <statement>

5. <statement> -> WHILE <expression>: EOL INDENT <statement_inside> <end> DEDENT <statement>

6. <statement> -> ID = <expression_start> <statement>

7. <statement> -> PASS <statement>

8. <statement> -> PRINT ( <arg> ) <statement>

9. <statement> -> <expression_start> <statement>

10. <statement_inside> -> IF <expression>: EOL INDENT <statement_inside> EOL DEDENT ELSE: EOL INDENT <statement_inside> <end> DEDENT <statement>

11. <statement_inside> -> WHILE <expression>: EOL INDENT <statement_inside> <end> DEDENT <statement>

12. <statement_inside> -> ID = <expression_start> <end> <statement_inside>

13. <statement_inside> -> RETURN <expression> <end> <statement_inside>

14. <statement_inside> -> PASS <end> <statement_inside>

15. <statement_inside> -> PRINT ( <arg> ) <end> <statement_inside>

16. <statement_inside> -> <expression_start> <end> <statement_inside>

17. <end> -> EOF

18. <end> -> EOL

19. <end> -> eps

20. <params> -> ID <next_params>

21. <params> -> eps

22. <next_params> -> , ID <next_params>

23. <next_params> -> eps

24. <expression_start> -> <value>

25. <expression_start> -> <function_call>

26. <value> -> ID <expression>

27. <value> -> INT <expression>

28. <value> -> DOUBLE <expression>

29. <value> -> STRING <expression>

30. <value> -> NONE <expression>
31. <function_call> -> ID ( <arg> )
32. <function_call> -> INPUTS()
33. <function_call> -> INPTUF()
34. <function_call> -> INPUTI()
35. <function_call> -> LEN ( <arg> )
36. <function_call> -> SUBSTR ( <arg> )
37. <function_call> -> ORD ( <arg> )
38. <function_call> -> CHR ( <arg> )
39. <arg> -> eps
40. <arg> -> <value> <arg>

## 7.3. LL tabuľka

| | EOF | EOL | DEF | IF | WHILE | , | ID | PASS | PRINT | RETURN | INT | DOUBLE | STRING | NONE | INPUTS | INPUTF | INPUTI | LEN | SUBSTZ | ORD | CHR | $ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| <statement> | 1 | 2 | 3 | 4 | 5 | | 6 | 7 | 8 | | | | | | | | | | | | | 9 |
| <statement_inside> | | | | 10 | 11 | | 12 | 14 | 15 | 13 | | | | | | | | | | | | 16 |
| <end> | 17 | 18 | | | | | | | | | | | | | | | | | | | | 19 |
| <params> | | | | | | | 20 | | | | | | | | | | | | | | | 21 |
| <next_params> | | | | | | 22 | | | | | | | | | | | | | | | | 23 |
| <expression_start> | | | | | | | | | | | | | | | | | | | | | | 24,25 |
| <value> | | | | | | | 26 | | | | 27 | 28 | 29 | 30 | | | | | | | | |
| <function_call> | | | | | | | 31 | | | | | | | | 32 | 33 | 34 | 35 | 36 | 37 | 38 | |
| <arg> | | | | | | | | | | | | | | | | | | | | | | 39,4 |

## 7.4. Precedentná tabuľka

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