



**Projektová dokumentace**  
**Implementace překladače imperativního jazyka IFJ23**  
Tým xdanie14, varianta vv-BVS

5. prosince 2023

|                       |                 |     |
|-----------------------|-----------------|-----|
| <b>Tomáš Daniel</b>   | <b>xdanie14</b> | X % |
| Martin Slezák         | xsleza26        | X % |
| Jakub Antonín Štigler | xstigl00        | X % |
| Milan Vrbas           | xvrbas01        | X % |

# Obsah

|          |   |          |
|----------|---|----------|
| <b>1</b> | <b>Úvod</b>                               | <b>1</b> |
| <b>2</b> | <b>Jednotlivé části překladače</b>        | <b>1</b> |
| <b>3</b> | <b>Návrh a implementace</b>               | <b>1</b> |
| 3.1      | Lexikální analýza . . . . .               | 1        |
| 3.1.1    | Diagram konečného automatu . . . . .      | 2        |
| 3.2      | Syntaktická analýza . . . . .             | 3        |
| 3.2.1    | Rekurzivní sestup . . . . .               | 3        |
| 3.2.2    | Precedenční syntaktická analýza . . . . . | 5        |
| 3.3      | Sémantická analýza . . . . .              | 6        |
| 3.3.1    | Tabulka symbolů . . . . .                 | 6        |
| 3.4      | Generování kódu . . . . .                 | 6        |
| 3.5      | Rozšíření . . . . .                       | 6        |
| <b>4</b> | <b>Práce v týmu</b>                       | <b>7</b> |
| 4.1      | Organizace . . . . .                      | 7        |
| 4.2      | Rozdělení práce . . . . .                 | 7        |
| <b>5</b> | <b>Závěr</b>                              | <b>7</b> |

---

# 1 Úvod

Tato dokumentace slouží k popisu implementace a vývoje **překladače pro jazyk IFJ23**, který je podmnožinou jazyka **Swift 5** od společnosti Apple. Program má za úkol načíst kód v tomto jazyku a přeložit ho do cílového jazyka **IFJcode23**. Vybrali jsme si variantu zadání **vv-BVS**, podle které jsme pomocí **výškově vyváženého binárního vyhledávacího stromu** implementovali **tabulku symbolů**.

## 2 Jednotlivé části překladače

Realizovaný překladač se dá rozdělit na následující dílní moduly:

- **Scanner** - lexikální analyzátor
- **Parser**
  - Syntaktický analyzátor
  - Sémantický analyzátor
- **Codegen** - Genérátor cílového kódu

Samotný překlad nejprve provede lexikální analýzu, následuje syntaktický analyzátor spolu se sémantickou analýzou a poté se přistoupí k generování cílového kódu.

## 3 Návrh a implementace

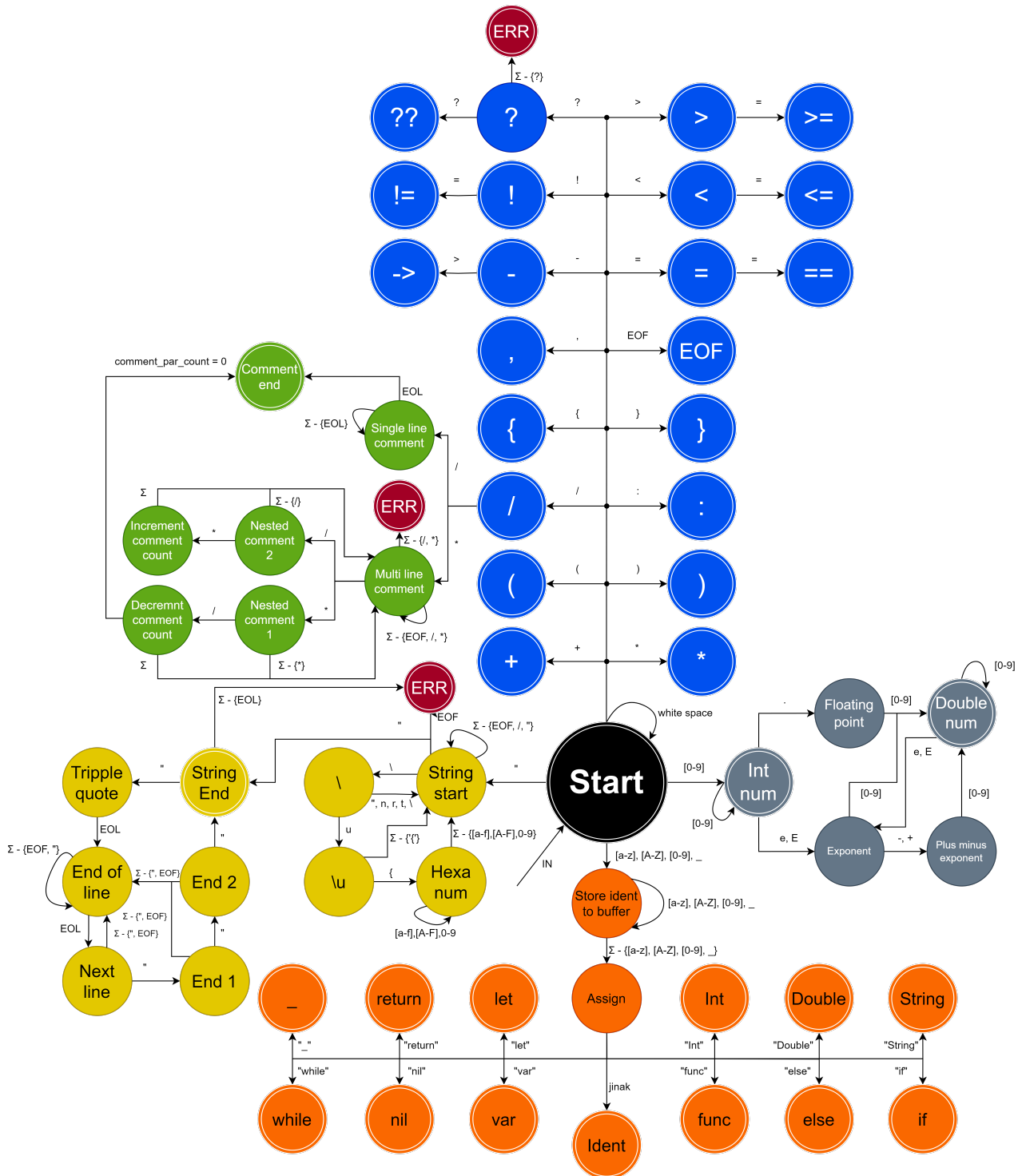
### 3.1 Lexikální analýza

Jako první část projektu, kterou jsme implementovali byla lexikální analýza, kterou lze najít v modulu *lexer.c*. Ta se stará o načítání znaků neboli lexémů ze standardního vstupu a rozdělování vstupních dat na jednotlivé tokeny, u nichž následně určuje jejich typ a posílá je k syntaktické a sémantické analýze k dalšímu zpracování.

Lexikální analýza využívá koncept **konečného automatu** pro identifikaci a zpracování tokenů ve vstupním kódu. Tento přístup umožňuje systematicky analyzovat postupně načtené znaky a identifikovat je jako jednotlivé tokeny na základě definovaných pravidel a přechodů mezi stavy automatu. Tímto způsobem je struktura zdrojového kódu efektivně analyzována a převedena do podoby tokenů pro další zpracování.

Funkce *lex\_next()* zajišťuje většinu vnitřní logiky scanneru tím, že načte token a uloží ho do struktury *Lexer*, která obsahuje důležité informace o každém předaném tokenu. V případě, že ve zdrojovém souboru narazí na neplatný token či neplatnou posloupnost znaků, překladač skončí s návratovou hodnotou, která odpovídá konkrétnímu příznaku lexikální chyby.

### 3.1.1 Diagram konečného automatu



Obrázek 1: Diagram zobrazující konečný automat

Poznámka: pokud automat nemá možnost přejít do dalšího stavu z nekonečného stavu, je to automaticky bráno jako chyba. (pro zjednodušení návrhu automatu)

---

## 3.2 Syntaktická analýza

### 3.2.1 Rekurzivní sestup

Implementace syntaktické analýzy se nachází v souboru **parser.c**, který se zabývá aplikací metody shora dolů, respektive **rekurzivního sestupu**, který je složen z **LL-gramatiky** a **LL-tabulky**. Parser si za svůj vstup bere jednotlivé tokeny z lexikální analýzy a následně nalpni tabulku symbolů v podobě výškově vyváženého binárního vyhledávacího stromu (vv-BVS). Pro každý neterminál z LL-gramatiky, je sestavena funkce, která následně volá další funkce na základě předcházejícího tokenu. Tyto tokeny se získávají pomocí funkce *lex\_next()*.

#### LL-gramatika

1. MAIN -> **''EOF''**
2. MAIN -> STATEMENT MAIN
3. MAIN -> **''func''** ident **''(''** PARAMS **''(''** RETVAL **''{''** STATEMENT **''}''** MAIN
4. STATEMENT -> **€**
5. PARAMS -> **€**
6. PARAMS -> name ident **'':''** TYPE PARAMS\_NEXT
7. TYPE -> **''Int''** Q\_MARK
8. TYPE -> **''Double''** Q\_MARK
9. TYPE -> **''String''** Q\_MARK
10. Q\_MARK -> **€**
11. Q\_MARK -> **''?''**
12. PARAMS\_NEXT -> **€**
13. PARAMS\_NEXT -> **''',''** PARAMS
14. RETVAL -> **€**
15. RETVAL -> **''->''** TYPE
16. STATEMENT -> VAR RIGHTSIDE STATEMENT
17. VAR -> **''let''** ident **'':''** TYPE
18. VAR -> **''var''** ident **'':''** TYPE
19. RIGHTSIDE -> **€**
20. RIGHTSIDE -> **''=''** EXPR
21. EXPR -> **''Do it using precedence table''**
22. STATEMENT -> ident **''=''** EXPR STATEMENT
23. STATEMENT -> **''if''** COND **''{''** STATEMENT **''}''** ELSESTAT
24. COND -> EXPR
25. COND -> **''(''** **''let''** ident **''(''**
26. COND -> **''let''** ident
27. ELSESTAT -> STATEMENT
28. ELSESTAT -> **''else''** **''{''** STATEMENT **''}''** STATEMENT
29. WHILESTAT -> **''while''** COND **''{''** STATEMENT **''}''** STATEMENT
30. FUNCCALL -> LEFTSIDE ident **''(''** FUNCPARAMS **''(''** STATEMENT
31. LEFTSIDE -> **€**
32. LEFTSIDE -> VAR **''=''**
33. FUNCPARAMS -> **€**
34. FUNCPARAMS -> ident **'':''** name FUNCPARAMS\_NEXT
35. FUNCPARAMS -> name FUNCPARAMS\_NEXT
36. FUNCPARAMS\_NEXT -> **€**
37. FUNCPARAMS\_NEXT -> **''',''** FUNCPARAMS
38. STATEMENT -> **''return''** VALUE STATEMENT
39. VALUE -> EXPR

## LL-tabulka

|                 | EOF | func | ident | (  | ) | { | } | $\epsilon$ | name | : | Int | Double | String | ?  | ,  | -> | let | var | =  | "Expression" | if | else | while | return |
|-----------------|-----|------|-------|----|---|---|---|------------|------|---|-----|--------|--------|----|----|----|-----|-----|----|--------------|----|------|-------|--------|
| MAIN            | 1   | 3    | 2     |    |   |   |   | 2          |      |   |     |        |        |    |    |    | 2   | 2   |    |              | 2  |      |       | 2      |
| STATEMENT       |     |      | 22    |    |   |   |   | 4          |      |   |     |        |        |    |    |    | 16  | 16  |    |              | 23 |      |       | 38     |
| PARAMS          |     |      |       |    |   |   |   | 5          | 6    |   |     |        |        |    |    |    |     |     |    |              |    |      |       |        |
| TYPE            |     |      |       |    |   |   |   |            |      |   | 7   | 8      | 9      |    |    |    |     |     |    |              |    |      |       |        |
| Q_MARK          |     |      |       |    |   |   |   | 10         |      |   |     |        |        | 11 |    |    |     |     |    |              |    |      |       |        |
| PARAMS_NEXT     |     |      |       |    |   |   |   | 12         |      |   |     |        |        |    | 13 |    |     |     |    |              |    |      |       |        |
| RETVL           |     |      |       |    |   |   |   | 14         |      |   |     |        |        |    |    | 15 |     |     |    |              |    |      |       |        |
| VAR             |     |      |       |    |   |   |   |            |      |   |     |        |        |    |    |    | 17  | 18  |    |              |    |      |       |        |
| RIGHTSIDE       |     |      |       |    |   |   |   | 19         |      |   |     |        |        |    |    |    |     |     | 20 |              |    |      |       |        |
| EXPR            |     |      |       |    |   |   |   |            |      |   |     |        |        |    |    |    |     |     |    | 21           |    |      |       |        |
| COND            |     |      |       | 25 |   |   |   |            |      |   |     |        |        |    |    |    | 26  |     |    | 24           |    |      |       |        |
| ELSESTAT        |     |      |       |    |   |   |   | 27         |      |   |     |        |        |    |    |    |     |     |    |              |    | 28   |       |        |
| WHILESTAT       |     |      |       |    |   |   |   |            |      |   |     |        |        |    |    |    |     |     |    |              |    |      | 29    |        |
| FUNCCALL        |     |      |       |    |   |   |   | 30         |      |   |     |        |        |    |    |    | 30  | 30  |    |              |    |      |       |        |
| LEFTSIDE        |     |      |       |    |   |   |   | 31         |      |   |     |        |        |    |    |    | 32  | 32  |    |              |    |      |       |        |
| FUNCPARAMS      |     |      | 34    |    |   |   |   | 33         | 35   |   |     |        |        |    |    |    |     |     |    |              |    |      |       |        |
| FUNCPARAMS_NEXT |     |      |       |    |   |   |   | 36         |      |   |     |        |        |    | 37 |    |     |     |    |              |    |      |       |        |
| VALUE           |     |      |       |    |   |   |   |            |      |   |     |        |        |    |    |    |     |     |    | 39           |    |      |       |        |

Obrázek 2: Tabulka zobrazující LL-tabulku

### 3.2.2 Precedenční syntaktická analýza

Precedenční syntaktická analýza neboli PSA se stará o syntaktickou analýzu metodou zdola-nahoru pomocí **precedenční tabulky**. Na základě žádosti PSA pošle lexikální analyzátor token, který se poté zpracuje s nejvyšším tokenem na zásobníku a funkce *prec\_table* na základě níže uvedených pravidel vrátí to, které se má na daný výraz aplikovat. Precedenční syntaktická analýza, kdy přijde prázdné pravidlo a zároveň je zásobník (*stack*) prázdný. Následně skončí úspěšně a vrátí poslední načtený znak. V opačném případě končí neúspěšně.

|          | $\pi$ | * | / | + | - | == | != | < | > | <= | >= | ?? | = | ( | ) | <i>t</i> | \$ |
|----------|-------|---|---|---|---|----|----|---|---|----|----|----|---|---|---|----------|----|
| $\pi$    | >     | > | > | > | > | >  | >  | > | > | >  | >  | >  | ! | c | > | .        | .  |
| *        | !     | > | > | x | x | >  | >  | > | > | >  | >  | >  | ! | < | > | <        | .  |
| /        | !     | > | > | x | x | >  | >  | > | > | >  | >  | >  | ! | < | > | <        | .  |
| +        | !     | 0 | 0 | > | > | >  | >  | > | > | >  | >  | >  | ! | < | > | <        | .  |
| -        | !     | 0 | 0 | > | > | >  | >  | > | > | >  | >  | >  | ! | < | > | <        | .  |
| ==       | !     | < | < | < | < | >  | >  | > | > | >  | >  | >  | ! | < | > | <        | .  |
| !=       | !     | < | < | < | < | >  | >  | > | > | >  | >  | >  | ! | < | > | <        | .  |
| <        | !     | < | < | < | < | >  | >  | > | > | >  | >  | >  | ! | < | > | <        | .  |
| >        | !     | < | < | < | < | >  | >  | > | > | >  | >  | >  | ! | < | > | <        | .  |
| <=       | !     | < | < | < | < | >  | >  | > | > | >  | >  | >  | ! | < | > | <        | .  |
| >=       | !     | < | < | < | < | >  | >  | > | > | >  | >  | >  | ! | < | > | <        | .  |
| ??       | !     | < | < | < | < | <  | <  | < | < | <  | <  | <  | ! | < | > | <        | .  |
| =        | !     | < | < | < | < | <  | <  | < | < | <  | <  | <  | ! | < | ! | <        | .  |
| (        | !     | < | < | < | < | <  | <  | < | < | <  | <  | <  | ! | < | = | <        | !  |
| )        | =     | > | > | > | > | >  | >  | > | > | >  | >  | >  | ! | c | > | .        | .  |
| <i>t</i> | >     | > | > | > | > | >  | >  | > | > | >  | >  | >  | > | c | > | .        | .  |
| \$       | <     | < | < | < | < | <  | <  | < | < | <  | <  | <  | < | ! | ! | <        | .  |

Tabulka 1: Tabulka pro precedenční analýzu

| Symbol | Název    | Popis   |
|--------|----------|---|
| <      | PA_SHIFT | Shift   |
| =      | PA_PUSH  | Push  |
| >      | PA_FOLD  | Fold  |
| .      | PA_END   | Successful end  |
| !      | PA_ERR   | Error   |
| x      | PA_TOP   | Shift if terminal is top, fold if non-terminal is top         |
| 0      | PA_STOP  | Fold if terminal after stop, shift if non-terminal after stop |
| c      | PA_CALL  | Parse function call   |

Tabulka 2: Legenda precedenční analýzy

---

### 3.3 Sémantická analýza

Sémantická analýza je implementována v modulu *semantics.c*, kde jsou na určitých místech prováděny kontroly, které zahrnují například ověření definice použitých funkcí a proměnných, počty proměnných a výrazů přiřazených do nich, shodu datových typů. Dále se kontroluje, zda nedochází k dělení nulou a zda návratové hodnoty funkcí odpovídají jejich hlavičkám.

Po úspěšném dokončení sémantické analýzy se přechází k tvorbě **abstraktně syntaktického stromu** (AST), který lze najít v *ast.c* a na základě kterého se nadále pracuje s generováním cílového kódu IFJcode23. V opačném případě při zjištění chyby končí program s odpovídající návratovou hodnotou a zároveň vypisuje na standardní chybový výstup informace a podrobnosti o daném problému.

#### 3.3.1 Tabulka symbolů

Tabulky symbolů jsou implementovány jako **výškově vyvážené binární vyhledávací stromy** v modulu *symtable.c*. Každý uzel obsahuje identifikátor (klíč, díky kterému vyhledáváme uzly), data a 2 ukazatele na jeho podstromy.

Tabulky jsou uloženy ve vektoru a dělí na lokální a globální. Lokální pro lokální proměnné a globální pro funkce. Tabulky symbolů jsou implementovány jako náhodně rozptýlené tabulky.

### 3.4 Generování kódu

Generování cílového kódu IFJcode23 je poslední částí tohoto překladače, který lze najít v modulu *codegen.c*. Pro tvorbu finálního kódu se využívá abstraktně syntaktický strom na základě, kterého se tento výsledný kód vytváří.

### 3.5 Rozšíření

Vzhledem k omezenému časovému rozsahu projektu jsme se rozhodli soustředit pouze na základní implementaci a nezačleňovat do ní žádná rozšíření. Místo toho jsme se zaměřili na dokončení základní funkcionality, abychom mohli dodržet termíny a dosáhnout funkčního jádra projektu.



---

## 4 Práce v týmu

### 4.1 Organizace

Na projektu jsme snažili začít pracovat hned od začátku jeho zadání – tedy na začátku října. Celková komunikace probíhala především pomocí komunikační platformy **Discord**, kde jsme se domluvali na dalším postupu, a nebo na přednáškách, na kterých jsme se potkávali. Práci jsme si rozdělili na základě schopností a dovedností jednotlivých členů týmu. Na dílčích částech projektu pracovali především jednotlivci s pomocí tzv. *Pull request*, díky kterému jsme měli možnost poskytovat zpětnou vazbu na konkrétní části kódu a který nám umožnil náš verzovaný systém **Git**, přičemž jako úložiště našeho repozitáře jsme využili platformu **GitHub**. Právě Git nám umožnil pracovat na více úkolech projektu v tzv. větvích, které jsme po otestování a schválení začlenili do hlavní vývojové větve.

### 4.2 Rozdělení práce

| Jméno člena týmu      | Přidělená práce  |
|-----------------------|--|
| Tomáš Daniel          | vedení týmu, lexikální analýza, vv-BVS, sémantická analýza ... |
| Martin Slezák         | ...  |
| Jakub Antonín Štigler | generování cílového kódu ...                                   |
| Milan Vrbas           | testování, dokumentace, prezentace                             |

Tabulka 3: Rozdělení práce mezi jednotlivé členy týmu

## 5 Závěr

Cílem tohoto projektu bylo pochopit a následně si prakticky vyzkoušet metody probírané v předmětech IFJ a IAL, díky kterým jsme byli schopni implementovat překladač a jeho jednotlivé části. Bohužel jsme v době pokusného odevzdání projektu ještě nedosáhli fáze, kdy bychom ho mohli odevzdat a nechat otestovat. Většina členů týmu již měla zkušenost s prací na týmovém úkolu, což nám pomohlo s komunikací a celkovým postupem při plnění tohoto zadání úkolu.

Projekt byl vzhledem ke své časové náročnosti poměrně komplexní, což znamenalo, že vyžadoval podstatně více času a pozornosti než jsme původně předpokládali. Nicméně jsme se tomu dokázali postavit a úspěšně ho dokončili včas.