VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

# Dokumentácia projektu z predmetov IFJ a IAL

## **Implementácia prekladača imperatívneho jazyka IFJ 18**

Tím č. 115, varianta I

| | | |
|---|---|---|
| **Daniel Štěpán** | **(xstepa60)** | **25%** |
| Filip Čaládi | (xcalad01) | 25% |
| Erika Fašánková | (xfasan00) | 25% |
| Jakub Timko | (xtimko02) | 25% |

05. decembra 2018

# 1 Úvod

Úlohou projektu bol návrh a implementácia prekladača imperatívneho jazyka IFJ18 (podmožina jazyka Ruby), ktorý načíta zdrojový kód v tomto jazyku a preloží ho do medzikódu = IFJcode18.
Zadanie malo dve varianty. My sme si vybrali variantu I, v ktorej je tabuľka symbolov implementovaná ako binárny vyhľadávací strom.
K implementácii nášho prekladača sme využili zdrojové súbory zjednodušenej implementácie interpretu jednoduchého jazyka zo stránky projektu (najmä *str.c* a *str.h*). Pre účely testovania sme používali referenčný interpret cieľového jazyka IFJcode18.

# 2 Rozdelenie práce

| **Meno a priezvisko** | **Rozdelenie práce** |
|---|---|
| Daniel Štěpán | vedúci tímu, návrh DKA, implementácia lexikálneho analyzátora, implementácia zásobníka, návrh a implementácia precedenčnej analýzy, generovanie výrazov, if a while, testovanie |
| Filip Čaládi | návrh a implementácia tabuľky symbolov, návrh a tvorba LL gramatiky, implementácia syntaktickej analýzy, implementácia vstavaných funkcií, generovanie kódu okrem výrazov, if a while, testovanie |
| Erika Fašánková | návrh a tvorba diagramu DKA, implementácia lexikálneho analyzátora, generovanie priradenia a výrazov, if a while testovanie, tvorba dokumentácie a prezentácie |
| Jakub Timko | návrh a implementácia tabuľky symbolov, implementácia zásobníka, návrh a implementácia precedenčnej analýzy, generovanie výrazov, if a while, testovanie |

# 3 Lexikálna analýza

Pri vytváraní nášho prekladača sme začali tvorbou lexikálneho analyzátora (scanner). Najskôr sme navrhli konečný deterministický automat (Príloha 1) na spracovanie lexém, na základe ktorého sme ho následne implementovali.
Hlavná funkcia lexikálneho analyzátora je `getNextToken()`, v ktorej načítame znaky zo štandardného vstupu, spracovávame ich a vraciame príslušný **token** syntaktickému analyzátoru, keď si oň požiada.

Náš scanner tiež využíva funkciu `hex_to_int()`, pomocou ktorej prevádzame hexadecimálne čísla zadané prostredníctvom escape sekvencie na ich ASCII hodnotu a vložíme do stringu ako jediný znak.
Pri implementácii nás najviac potrápili blokové komentáre. Využívame pri nich premennú `new_line`, podľa ktorej vieme, či sa nachádzame na novom riadku (jej hodnota je 1). Do pomocného dynamického stringu `begin_str` načítame postupne 5 znakov. Ak za nimi nasleduje medzera alebo znak nového riadku a súčasne obsah dynamického stringu je totožný s „begin“, tak vieme že sa nachádzame na začiatku blokového komentára. V ostatných prípadoch vieme, že nepôjde o blokový komentár a znaky, ktoré sme si načítali dopredu vrátime. Analogicky sme postupovali aj v prípade ukončenia blokového komentára.

# 4 Tabuľka symbolov

Tabuľka symbolov je implementovaná ako binárny vyhľadávací strom. Na jej implementáciu sme využili poznatky z predmetu IAL.
Implementácia zahŕňa tabuľku symbolov pre funkcie a pre premenné. Každý prvok v tabuľke symbolov pre funkcie obsahuje vlastnú tabuľku symbolov pre lokálne premenné danej funkcie.

# 5 Syntaktická analýza

Syntaktická analýza kontroluje syntaktickú správnosť a taktiež riadi celý preklad. Je založená na LL gramatike (Príloha 2) a využíva metódu rekurzívneho zostupu.
Hlavnými funkciami sú `statList()`, ktorá prebieha do konca súboru alebo kým nenájde chybu a funkcia `stat()`, ktorá je rekurzívne volaná a spracováva vstupný program na základe tokenov od lexikálneho analyzátora. Každá funkcia v syntaktickej analýza zhora nadol zastupuje jeden neterminál z návrhu gramatiky.
Výrazy spracovávame pomocou precedenčnej tabuľky (Príloha 3). Index do tabuľky získavame prostredníctvom funkcie `getIndex()`. Pri implementácii sme využili algoritmus z prednášky. Precedenčná analýza pozostáva z dvoch hlavných funkcií. Funkcia `get_expression()` spracováva výrazy a priradenia, pokiaľ aktuálny token nie je EOL. Na spracovanie podmienok vo funkciách `if/else` a `while` je určená funkcia `get_if_or_while_expressions()`, ktorá sa ukončí ak aktuálny token bude `then` alebo `do`.
Počas redukcie výrazov sa volajú funkcie, ktoré generujú spracovanie výrazov v medzikóde IFJcode18.

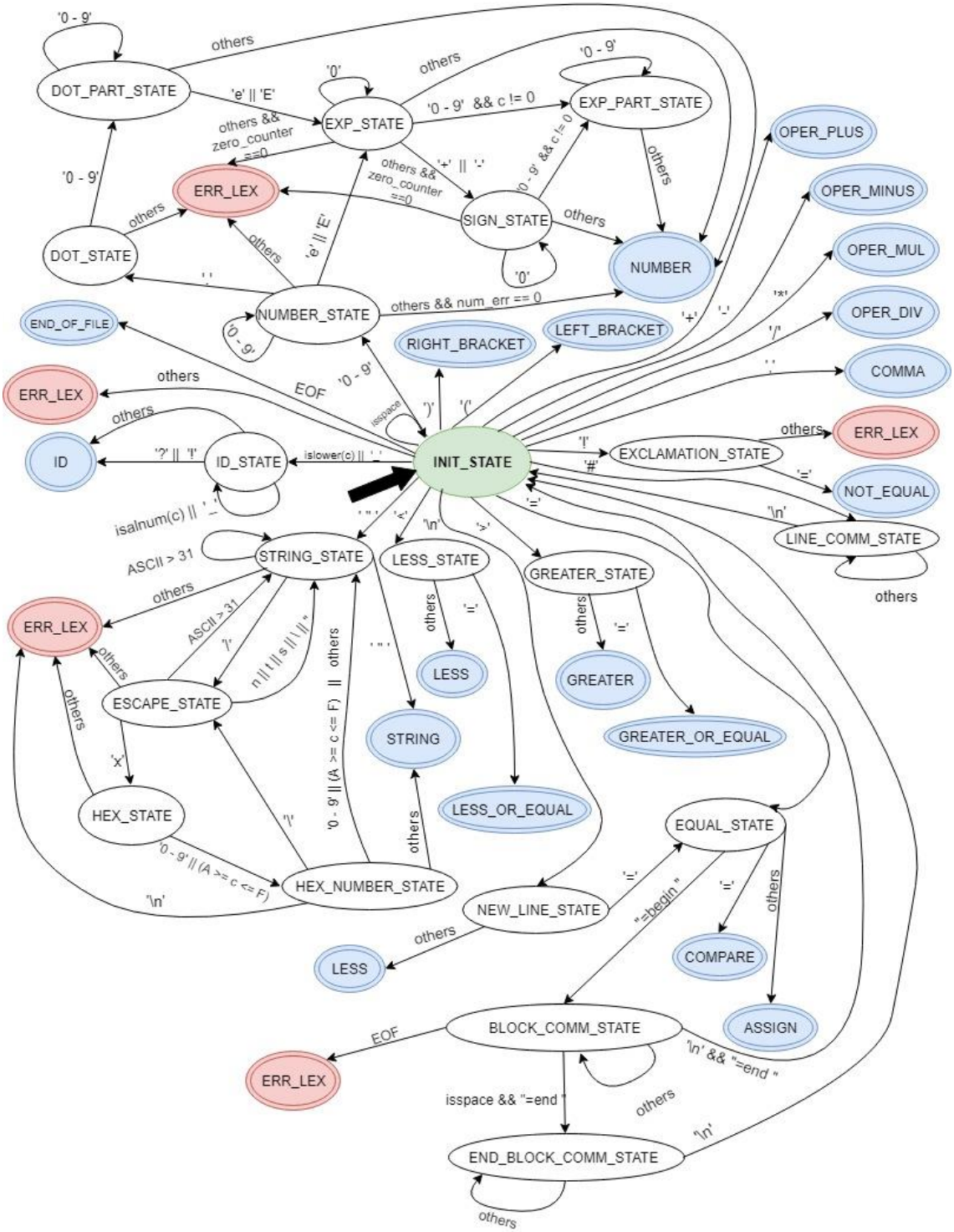
# 6 Generátor kódu

Na začiatku je vygenerovaná hlavička medzikódu a rámec. Postupne sú potom volané ďalšie funkcie na definovanie premenných, generovanie vstavaných funkcií, priradenia a výrazov.
V rámci *generate.c* máme vytvorené tiež funkcie `isFloat()` a `isInt()`, na základe ktorých rozlišujeme datový typ čísel. Pre správny zápis literálu typu string je určená funkcia `toRightString()`.
Pri generovaní spracovania výrazov kontrolujeme typy hodnôt daných premenných. V prípade potreby, dané typy hodnôt prevedieme na tie správne (napríklad pri sčítaní dvoch čísel typu int a float, sa číslo typu int prevedie na typ float a následne sa sčítajú). Ak prevod nie je možný, funkcia na generovanie výrazov končí so sémantickou chybou.
Z inštrukčnej sady jazyka IFJcode18 využívame inštrukcie pre prácu s datovým zásobníkom a zásobníkové verzie aritmetických a relačných inštrukcií.

# 7 Záver

Na začiatku sme mali časovú rezervu. Neskôr už nás tlačil čas a nepodarilo sa nám prvé pokusné odovzdanie, nakoľko sme neprávne zazipovali naše zdrojové súbory a taktiež sme vstup nenačítali zo stdin a negenerovali medzikód na stdout, ale z a do súboru. Preto sme využili druhé pokusné odovzdanie, po ktorom sme vedeli, na ktorých častiach treba ešte popracovať.
Tento projekt nám priniesol nové skúsenosti, najmä v oblasti tímovej práce. Tiež sme si vyskúšali prácu s verzovacím systémom Git a prakticky sme si vyskúšali teoretické znalosti z predmetov IFJ a IAL.

# 8 Prílohy




Konečný stav ERR_LEX je v skutočnosti len jeden (len v rámci prehľadnosti nákresu ich je viac)

Príloha 1: Deterministický konečný automat lexikálneho analyzátora

<prog> → <st-list>
<st-list> → <stat><st-list>
<st-list> → EOF
<stat>→ id = id(<item-list>) EOL
<stat> → id(<item-list>) EOL
<stat> → id = id <item-list> EOL
<stat> → id <item-list> EOL
<stat> → def id (<item-list>) EOL <st-list> end EOL
<stat> → id = inputs (<item-list>) EOL | id = inputi (<item-list>) EOL |id = inputf (<item-list>) EOL
<stat> → print (<item-list>) EOL | print <item-list> EOL
<stat> → length (<item-list>) EOL | length <item-list> EOL
<stat> → substr (<item-list>) EOL | substr <item-list> EOL
<stat> → ord (<item-list>) EOL | ord <item-list> EOL
<stat> → chr (<item-list>) EOL | chr <item-list> EOL
<stat> → if <expr> then EOL <st-list> else <st-list> end EOL
<stat> → while <expr> do EOL <st-list> end EOL
<stat> → id = <expr> EOL
<stat> → id = id EOL
<stat> → <expr> EOL
<stat> → <item> EOL
<item-list> → ε
<item-list> → <item><item-list2>
<item-list2> → ε
<item-list2> → ,<item><item-list2>
<item> → id
<item> → int
<item> → float
<item> → string

Príloha 2: Návrh LL gramatiky

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

Príloha 3: Návrh precedenčnej tabuľky