

Projektová dokumentace

Překladač jazyka IFJ19

Tým ???, Varianta ??

**Boris Burkalo** (xburka00) XX%

Jiří Hergott (xhergo??) XX%

Jan Klusáček (xklusa14) XX%

Prosinec 2019

Obsah

[1. Úvod 3](#_Toc26120541)

[2. Tým – rozdělení 3](#_Toc26120542)

[2.1. Týmová práce 3](#_Toc26120543)

[2.3. Tabulka rozdělení práce 4](#_Toc26120544)

[3. Implementace 4](#_Toc26120545)

[3.1. Lexikální analýza 4](#_Toc26120546)

[3.2. Syntaktická a sémantická analýza 4](#_Toc26120547)

[3.3. Generování kódu 5](#_Toc26120548)

# Úvod

Zadáním projektu bylo vytvořit program v jazyce C, který načte zdrojový kód napsaný ve zdrojovém jazyce IFJ19, a přeloží jej do cílového jazyka IFJcode19 (mezikód). Jazyk IFJ19 je zjednodušenou podmnožinou jazyka Python3, a jedná se o dynamicky typovaný imperativní jazyk s funkcionálními prvky.

# Tým – rozdělení

## Týmová práce

V polovině října jsme se začali jako tým pravidelně scházet v prostorách školy. Nejprve jsme si společně pročítali zadání, a snažili jsme se nějak rozdělit následnou práci. Zpočátku jsme si rozdělili jen malý úsek, a potom jsme si práci dělili podle aktuálních potřeb. Mimo rozdělenou práci jsme si navzájem pomáhali s různými problémy.

Náš tým zvolil variantu II., která k implementaci použila tabulku s rozptýlenými položkami. Ke zhotovení projektu jsme využili několik dílčích částí, které budou popsané v následujících kapitolách. K řešení jsme využívali informace získané na přednáškách a stránce projektu, ale také jsme si sami vyhledávali potřebné informace na internetu.

* 1. Vývojové prostředí + verzování

Všichni členové týmu jsme byli zvyklí používat operační systém Linux (PopOS, Ubunutu), a tak nikde nenastával problém s kompatibilitou.

K práci jsme využívali verzovací systém GitHub, kde nám stačilo využít jednu větev, jelikož na daném souboru pracovali vždy maximálně dva lidé, a tak díky vzájemné komunikaci (hlavně pomocí aplikace Discord) nebyl problém s mergováním. Nicméně občas bylo nutné setkání ve škole např. kvůli propojení některých modulů.

Funkčnost jsme testovali vzdáleně na CentOS Merlin, a paměťové úniky pomocí programu Valgrind.

Dokumentace je napsána v programu Microsoft Word a graf vykreslen v pomocí draw.io.

## Tabulka rozdělení práce

|  |  |
| --- | --- |
| Boris Burkalo | Vedení týmu, parser |
| Jiří Hergott | Lexikální analyzátor, generování kódu |
| Jan Klusáček | Parser, dokumentace |

# Implementace

## Lexikální analýza

Hlavním modulem pro lexikální analýzu je modul scanner. Stavebním kamenem scanneru je funkce int get\_next\_token(Token \*token), která čte znak po znaku vstupní soubor a ve switch case přechází do jednotlivých stavů konečného automatu (viz obr.). V případě načtení znaku patřícího dalšímu tokenu se volá funkce ungetc a na vyžádání parseru se začíná načítat další token. Funkce get\_next\_token vrací příslušnou int hodnotu v případě neúspěchu, jinak vrací 0????. Do struktury Token předává typ daného lexému a případně jeho data. V případě, že se jedná o typ identifikátor, kontroluje se, jestli se nejedná o klíčové slovo.

Kromě posílání tokenů do parseru má scanner za úkol také odstranit z kódu nepotřebné části, jako jsou například komentáře.

Dalším modulem, který se využívá pro scanner je modul strings, ve kterém se nachází struktura a funkce pro práci s dynamickým stringem (možná napsat něco víc).

## Syntaktická a sémantická analýza

Tělem syntaktické a sémantické analýzy je modul parser, který však využívá další moduly popsané níže. Funguje na základě analýzy shora dolů a je řízen pravidly LL(1) gramatiky. Z modulu main je volána funkce prog, která se postará o vše potřebné pro spuštění funkce body, jako například spuštění scanneru, vytvoření tabulky s rozptýlenými položkami atd. Funkce body si žádá o tokeny ze vstupního souboru pomocí funkce get\_next\_token, které jsou předány ze scanneru a již prošly lexikální analýzou. Následně volá další funkce, pomocí kterých se zpracovávají celé příkazy.

Na zpracování výrazů se používá precedenční tabulka, která je implementována ve zvláštním modulu expression.

Přidat tabulku a asi o ní něco napsat.

## Generování kódu

V modulu generator se nacházejí funkce, které jsou volány za běhu parsování a postupně vytvářejí dynamycký string, obsahující výsledný kód. String je poslán na standartní výstup v případě že nenastala žádná chyba. Pomocný modul generator\_functions slouží pro generování vestavěných funkcí. Možná něco doplnit

SPEC. ALG A DAT STRUKTURY (dyn řetězec, hashovací tabulka, zásobík?)????

Závěr???

🡪

Grafy a obrázky

• Diagram konečného automatu, který specifikuje lexikální analyzátor.

19

• LL-gramatiku, LL-tabulku a precedenční tabulku, podle kterých jste implementovali

váš syntaktický analyzátor.

🡨

POUŽITÁ LITERATURA -> NĚCO BY TAM MĚLO BÝT

Textová část řešení

Součástí řešení bude dokumentace vypracovaná ve formátu PDF a uložená v jediném souboru dokumentace.pdf. Jakýkoliv jiný než předepsaný formát dokumentace bude ignorován, což povede ke ztrátě bodů za dokumentaci. Dokumentace bude vypracována v českém, slovenském nebo anglickém jazyce v rozsahu cca. 3-5 stran A4.

V dokumentaci popisujte návrh (části překladače a předávání informací mezi nimi),

implementaci (použité datové struktury, tabulku symbolů, generování kódu), vývojový cyklus, způsob práce v týmu, speciální použité techniky a algoritmy a různé odchylky od přednášené látky či tradičních přístupů. Nezapomínejte také citovat literaturu a uvádět reference na čerpané zdroje včetně správné citace převzatých částí (obrázky, magické konstanty,

vzorce). Nepopisujte záležitosti obecně známé či přednášené na naší fakultě.

Dokumentace musí povinně obsahovat (povinné tabulky a diagramy se nezapočítávají do doporučeného rozsahu):

• 1. strana: jména, příjmení a přihlašovací jména řešitelů (označení vedoucího) + údaje

o rozdělení bodů, identifikaci vaší varianty zadání ve tvaru “Tým číslo, varianta 𝑋” a

výčet identifikátorů implementovaných rozšíření.

• Rozdělení práce mezi členy týmu (uveďte kdo a jak se podílel na jednotlivých částech

projektu; povinně zdůvodněte odchylky od rovnoměrného rozdělení bodů).

• Diagram konečného automatu, který specifikuje lexikální analyzátor.

19

• LL-gramatiku, LL-tabulku a precedenční tabulku, podle kterých jste implementovali

váš syntaktický analyzátor.

Dokumentace nesmí:

• obsahovat kopii zadání či text, obrázky27 nebo diagramy, které nejsou vaše původní

(kopie z přednášek, sítě, WWW, …).

• být založena pouze na výčtu a obecném popisu jednotlivých použitých metod (jde

o váš vlastní přístup k řešení; a proto dokumentujte postup, kterým jste se při řešení

ubírali; překážkách, se kterými jste se při řešení setkali; problémech, které jste řešili

a jak jste je řešili; atd.)

V rámci dokumentace bude rovněž vzat v úvahu stav kódu jako jeho čitelnost, srozumitelnost a dostatečné, ale nikoli přehnané komentáře