VUT\_LOGO

Projektová dokumentace

Implementace překladače pro imperativní jazyk IFJ23

Tým xblaze38

Varianta vv-BVS

DATUM JMÉNA, LOGINY, ROZDĚLENÍ

Obsah

# Úvod

Cílem projektu bylo vytvořit překladač jazyka IFJ23 do cílového kódu v jazyce IFJcode23. Jazyk IFJ23 je podmnožinou jazyka Swift.

Překladač je pouze konzolová aplikace, která dostane na vstup kód v jazyce IFJ23 a převede ho na výstup v jazyce IFJcode23, přičemž zkontroluje vstupní kód a případně vrátí chybovou hodnotu pro odpovídající chybový stav.

# Návrh

## Lexikální analýza

Lexikální analýza je jedna z prvních částí projektu. Je vhodné ji mít hotovou co nejdříve z důvodu kompatibility v dalšími částmi projektu jako je například syntaktický analyzátor.

Lexikální analýza je uložena v souborech scanner.c a scanner.h a její jedinou funkcí, kterou využívají ostatní části projektu je getToken(). Funkce getToken() čte postupně ze standartního vstupu předem neurčený počet znaků, skládá je za sebe a tím vytváří tokeny. Tyto tokeny mají svůj formát uložený ve struktuře Token a jejich obsah je podle typu načtených znaků uložen do union tokenAttrib. Lexikální analyzátor má také za úkol určit typ tokenu, který je uložen v enum tokenType a případně i v enum tokenSecondType. To dělá pomocí koncových stavů konečného automatu. Jelikož je lexikální analyzátor schopný rozlišit stav, ve kterém se nachází, je schopný podle toho i určit typ tokenu.

Náš lexikální analyzátor je implementován pomocí konečného automatu, jehož diagram je na stránce XXX. Tento konečný automat je nekonečný cyklus, který přepíná pomocí přepínače mezi stavy tohoto automatu. V případě načtení maximálního možného tokenu zkontroluje, zda se nachází v koncovém stavu. Pokud se nenachází v koncovém stavu, vrací chybovou návratovou hodnotu 1, a pokud se nachází v koncovém stavu, vrátí token pomocí parametru funkce getToken().

Další funkcí pro tento problém je funkce isKeyword(), která při načítání identifikátorů kontroluje, jestli daný identifikátor není některé z klíčových slov jazyka IFJ23. Jestliže funkce určí identifikátor jako klíčové slovo, přepíše jeho typ.

## Syntaktická analýza

## Sémantická analýza

# Implementace

## Datové struktury

V projektu je využito jen pár abstraktních datových struktur. Za zmínku stojí 2 zásobníky v souborech stack.c a stack.h.

První je ve struktuře SymtabStack, který byl původně připraven pro ukládání aktivních tabulek symbolů. Tento zásobník se bohužel nepodařilo rozumně vsadit do zbytku projektu.

Druhým zásobníkem je zásobník tokenů, který je využit v souboru parser.c pro případné uložení tokenů, které bychom mohli potřebovat při zpracovávání výrazu. Dále se používá v souboru expression.c, kde si během zpracovávání výrazů ukládáme jednotlivé tokeny na zásobník.

Oba tyto zásobníky mají základní funkce Init, Top, Pop, Push a Dispose. Zásobník na tokeny má ještě funkce tStack\_Second\_Top() a tStack\_Insert(), které slouží pro přečtení 2. prvku odshora a vložení jinam než na vrchol zásobníku.

## Tabulka symbolů

Tabulku symbolů jsme podle zadání

## Generátor cílového kódu

# Práce v týmu

## Komunikace v týmu

## Využití verzovacího systému

## Rozdělení práce

# Závěr

# Diagram konečného automatu pro lexikální analýzu

# LL – gramatika specifikující syntaktickou analýzu

# Použitá LL – tabulka

# Precedenční tabulka pro zpracování výrazů