



**Projektová dokumentace**  
**Implementace překladače jazyka IFJ23**  
Tým xhalam16, varianta TRP-izp

28. listopadu 2023

<b>Marek Halamka</b>	<b>(xhalam16)</b>	TBA, %
Šimon Motl	(xmotls00)	TBA %
Richard Juřica	(xjuric31)	TBA %
Jan Kroutil	(xkrout04)	TBA %

# Obsah

<b>1</b>	<b>Úvod</b>	<b>2</b>
<b>2</b>	<b>Implementace</b>	<b>2</b>
2.1	Pomocné moduly a rozhraní . . . . .	2
2.1.1	Error . . . . .	2
2.1.2	DynamicBuffer . . . . .	2
2.1.3	Token . . . . .	3
2.1.4	Tabulka symbolů . . . . .	3
2.1.5	Parameter list . . . . .	3
2.1.6	Stack . . . . .	4
2.2	Lexikální analýza . . . . .	4
2.3	Syntaktická analýza . . . . .	4
2.4	Sémantická analýza . . . . .	4
2.5	Generování cílového kódu . . . . .	4

# 1 Úvod

Překladač jazyka IFJ23 je projekt vytvořený v rámci předmětů IFJ a IAL na FIT VUT v Brně.

Cílem projektu je vytvořit překladač jazyka IFJ23, který bude překládat zdrojový kód napsaný v tomto jazyce do cílového jazyka IFJcode23 a vrací příslušný návratový kód.

Program je implementován jako konzolová aplikace, která na standardní vstup přijímá zdrojový kód jazyka IFJ23 a na standardní výstup vypisuje cílový kód jazyka IFJcode23.

Překladač je implementován v jazyce C dle normy C11<sup>1</sup> a je rozdělen do několika modulů. Každý modul má svůj hlavičkový soubor, který obsahuje deklarace funkcí a struktur definovaných v daném modulu.

## 2 Implementace

Zvolená metoda implementace je jednoduruchodový **syntaxí řízený překlad** a skládá se z částí, které jsou popsány v následujících podkapitolách.

### 2.1 Pomocné moduly a rozhraní

Program mimo hlavní funkce obsahuje několik důležitých modulů a rozhraní využívaných v rámci celého projektu. Jejich výčet a popis se nachází v následujících podkapitolách.

#### 2.1.1 Error

Toto rozhraní, definované v hlavičkovém souboru `error.h`, obsahuje výčtový typ chybových stavů, které mohou nastat při běhu programu.

#### 2.1.2 DynamicBuffer

Tento modul definovaný v souboru `dynamic_buffer.c` má za úkol uchovávat řetězce proměnné délky. Korespondující hlavičkový soubor `dynamic_buffer.h` obsahuje deklarace funkcí a struktur definovaných v tomto modulu.

Rozhraní obsahuje strukturu `dynamic_buffer`, která obsahuje **ukazatel** na alokovanou paměť, **kapacitu** alokované paměti a **velikost** obsazené paměti.

Modul obsahuje funkce pro inicializaci, uvolnění, realokaci a práci s řetězci. Buffer se v případě naplnění automaticky realokuje na dvojnásobek své původní kapacity.

Pomocí výše zmíněných funkcí jsme definovali abstraktní datový typ `DynamicBuffer`, který je využíván v dalších částech projektu.

---

<sup>1</sup>ISO/IEC 9899:2011, viz. <https://www.iso.org/standard/57853.html>

### 2.1.3 Token

Hlavičkový soubor `token.h` obsahuje definici struktury `token`, která reprezentuje token.

Struktura obsahuje typ tokenu, hodnotu tokenu a textovou reprezentaci tokenu ve zdrojovém souboru.

**Typ tokenu** je definován výčtovým typem `token_type`, který obsahuje všechny typy tokenů jazyka IFJ23.

**Hodnota tokenu** je typu `union`, a nabývá buď hodnoty odpovídající číselné hodnotě tokenu, nebo ukazatele na dříve definovanou strukturu `DynamicBuffer`, která reprezentuje řetězec. Typ `union` byl zvolen z důvodu úspory paměti, protože token může obsahovat pouze jednu z těchto hodnot.

**Textová reprezentace tokenu** ve zdrojovém souboru je uložena jako ukazatel na strukturu `DynamicBuffer`, který obsahuje řetězec. Je uchovávána z důvodu implementace funkce `peek_token`, která je podrobněji popsána v sekci 2.2.

### 2.1.4 Tabulka symbolů

Tabulka symbolů slouží k uložení informací o proměnných a funkcích.

Deklarace proměnné či funkce odpovídá vytvoření záznamu v tabulce symbolů, kde klíčem je **identifikátor** proměnné.

Dle zvolené varianty zadání je tabulka symbolů implementována jako **TRP s otevřenou adresací**.

Implicitní rozptýlení využívá **lineární** určení kroku při výpočtu dalšího volného indexu.

Tabulka symbolů je implementována v souboru `symtable.c` s rozhraním v souboru `symtable.h`.

Rozhraní obsahuje funkce pro inicializaci, uvolnění, vložení a vyhledání položky v tabulce symbolů či vytvoření nové položky.

Kromě těchto funkcí obsahuje i signaturní **hashovací funkci**, která je použita při transformaci klíče, na index do tabulky symbolů.

Tabulka se v případě naplnění automaticky **realokuje** na dvojnásobek své původní kapacity.

V programu rozlišujeme mezi GLOBÁLNÍ a LOKÁLNÍ tabulkou symbolů. Globální tabulka symbolů je vytvořena při inicializaci programu a je uvolněna při jeho ukončení. Lokální tabulka symbolů je vytvořena při vstupu do bloku a je uvolněna při jeho opuštění.

Tyto typy tabulek se liší mimo jiné i v datech, které uchovávají. Funkce pro práci s tabulkou symbolů jsou implementovány tak, aby bylo možné používat stejné funkce pro oba typy tabulek. Docíleno je to pomocí ukazatele typu `void`, který je přetypován na konkrétní typ tabulky symbolů v závislosti na tom, zda se jedná o GLOBÁLNÍ nebo LOKÁLNÍ tabulku symbolů.

### 2.1.5 Parameter list

Parametry funkcí jsou uchovávány v seznamu, který je implementován jako **jednosměrně vázaný seznam**.

Seznam je implementován v souboru `symtable.c` s rozhraním v souboru `symtable.h`.

Rozhraní obsahuje funkce pro inicializaci, uvolnění, vložení a vyhledání položky v seznamu.

Implementací výše zmíněných funkcí je definován abstraktní datový typ `parameter_list_t`, který je využíván v dalších částech projektu.

#### **2.1.6 Stack**

### **2.2 Lexikální analýza**

### **2.3 Syntaktická analýza**

### **2.4 Sémantická analýza**

### **2.5 Generování cílového kódu**