

# Dokumentace návrhu a implementace

Projekt implementace překladače imperativního jazyka IFJ22

Tým xpavli95

Nebyla implementována žádná rozšíření

Pavlíček Jan (xpavli95) – Vedoucí týmu 25%

Nekula Štěpán (xnekul04) 25%

Vadovič Matej (xvadov01) 25%

Soukup Martin (xsouku15) 25%

Obsah

[Dokumentace návrhu a implementace 1](#_Toc121236561)

[1. Motivace projektu 3](#_Toc121236562)

[2. Návrh a implementace 3](#_Toc121236563)

[2.1. Lexikální analýza 3](#_Toc121236564)

[2.2. Rekurzivní sestup 5](#_Toc121236565)

[2.3. Precedenční analýza 6](#_Toc121236566)

[2.4. Sémantická analýza 7](#_Toc121236567)

[2.5. Generování kódu 8](#_Toc121236568)

[3. Použité algoritmy a struktury 8](#_Toc121236569)

[3.1. String **Chyba! Záložka není definována.**](#_Toc121236570)

[3.2. Zásobníky 8](#_Toc121236571)

[3.3. Tabulka symbolů 8](#_Toc121236572)

[3.4. AST 9](#_Toc121236573)

[4. Rozdělení práce 9](#_Toc121236574)

[4.1. Podrobnější rozdělení 9](#_Toc121236575)

[5. Závěr 9](#_Toc121236576)

[Obrázek 1: Diagram konečného automatu 4](file:////Users/syo.agency/Desktop/Dokumentace%20návrhu%20a%20implementace.docx#_Toc121236577)

[Obrázek 2: LL-gramatika 5](file:////Users/syo.agency/Desktop/Dokumentace%20návrhu%20a%20implementace.docx#_Toc121236578)

[Obrázek 3: LL-tabulka 5](file:////Users/syo.agency/Desktop/Dokumentace%20návrhu%20a%20implementace.docx#_Toc121236579)

[Obrázek 4: Precedenční tabulka 6](#_Toc121236580)

[Obrázek 5: Diagram AST 7](file:////Users/syo.agency/Desktop/Dokumentace%20návrhu%20a%20implementace.docx#_Toc121236581)

# Motivace projektu

Vytvořit program v jazyce C, který načte zdrojový kód jazyka IFJ22 a přeloží jej do cílového jazyka IFJcode22. Samotný jazyk je zjednodušenou podmnožinou jazyka PHP.

# Návrh a implementace

Program byl rozdělen na pět spolu komunikujících částí.

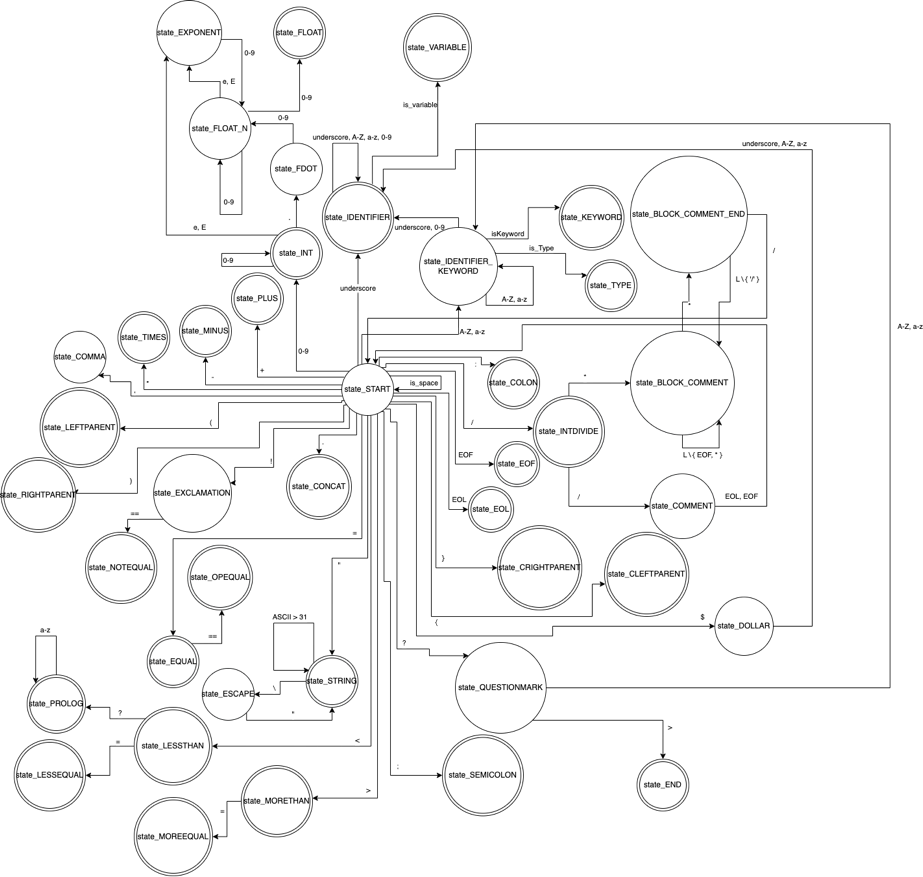
* Lexer ( scanner.c/h )
* Rekurzivní sestup ( recursive\_descent.c/h )
* Precedenční analýza ( precedent.c/h )
* Sémantická analýza ( semantic.c/h )
* Generování kódu ( code\_generator.c/h )

Hlavním tělo programu se nachází v souboru main.c a volá rekurzivní sestup, který poté řídí celý průběh překladu.

## Lexikální analýza

Lexikální analýza (dále Scanner) je implementována pomocí konečného stavový automatu. Hlavní funkce getToken přebírá jako parametry strukturu Token a soubor se zdrojovým kódem načítaný ze standardního vstupu.

Scanner postupně prochází znaky a podle vlastností jazyka IFJ22 tvoří lexémy. Jakmile se scanner dostane do konečného stavu a lexém má správný tvar, přiřadí se mu typ konečného stavu. Pokud scanner odhalí lexikální chybu (neplatný znak, špatný tvar lexému), skončí s návratovou hodnotou 1. Speciální případy nastanou, pokud je špatně zapsán prolog zdrojového kódu nebo uzavírací značka, kdy scanner končí s návratovou hodnotou 2.



Obrázek : Diagram konečného automatu

## Rekurzivní sestup

Obsah obrázku stůl

Popis byl vytvořen automatickyRekurzivní sestup je základní část syntaktické analýzy, provádí syntaktickou kontrolu jednotlivých částí zdrojového kódu, provádí postupné vytváření AST, který poté předává do sémantické analýzy na další modifikování. Ke svému průchodu volá postupně scanner a jednotlivé tokeny ze scanneru ukládá do zásobníku, s tímto zásobníkem tokenů poté pracuje po celou dobu rekurzivního sestupu. V případě, že narazí na výraz, volá precedenční analýzu. Pravidla pro LL-gramatiku jsou tvořena ručně, ke generování LL-tabulky byl využit online nástroj.

Obrázek : LL-gramatika

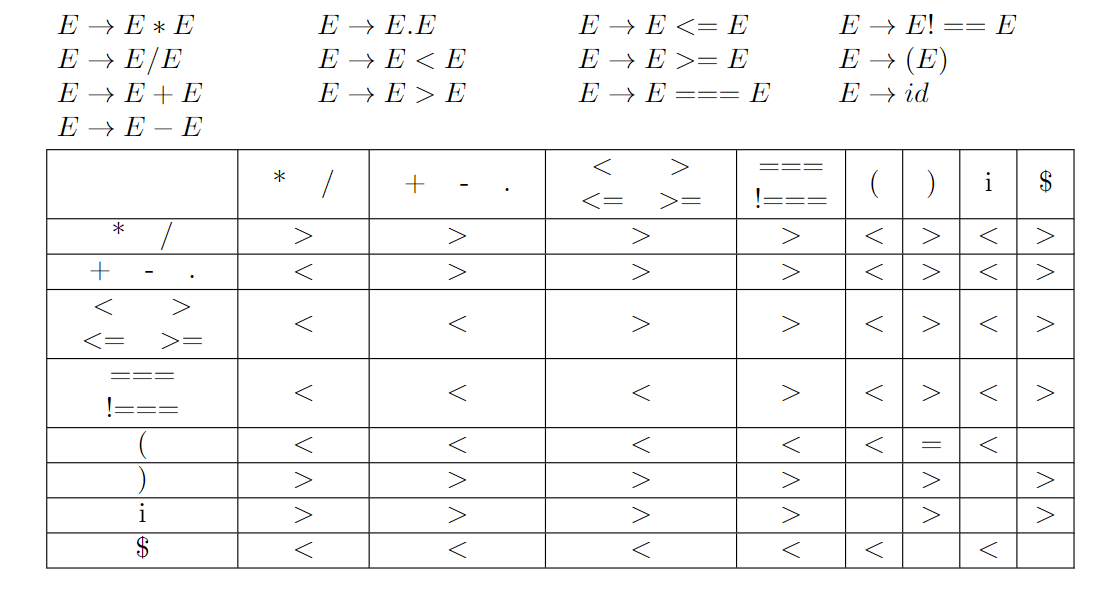
Obsah obrázku text, japonské posuvné dveře

Popis byl vytvořen automaticky

Obrázek : LL-tabulka

## Precedenční analýza

Precedenční analýza je druhá část syntaktické analýzy. Volá se v případě, kdy rekurzivní sestup dojde k výrazu. Precedenční analýza poté postupně kontroluje syntaxi zadaného výrazu. Vytváří postfixovou notaci výrazu, pokud není nalezena žádná chyba ve výrazu, precedenční analýza vytvoří kus abstraktního syntaktického stromu odpovídající danému výrazu.



Obrázek : Precedenční tabulka

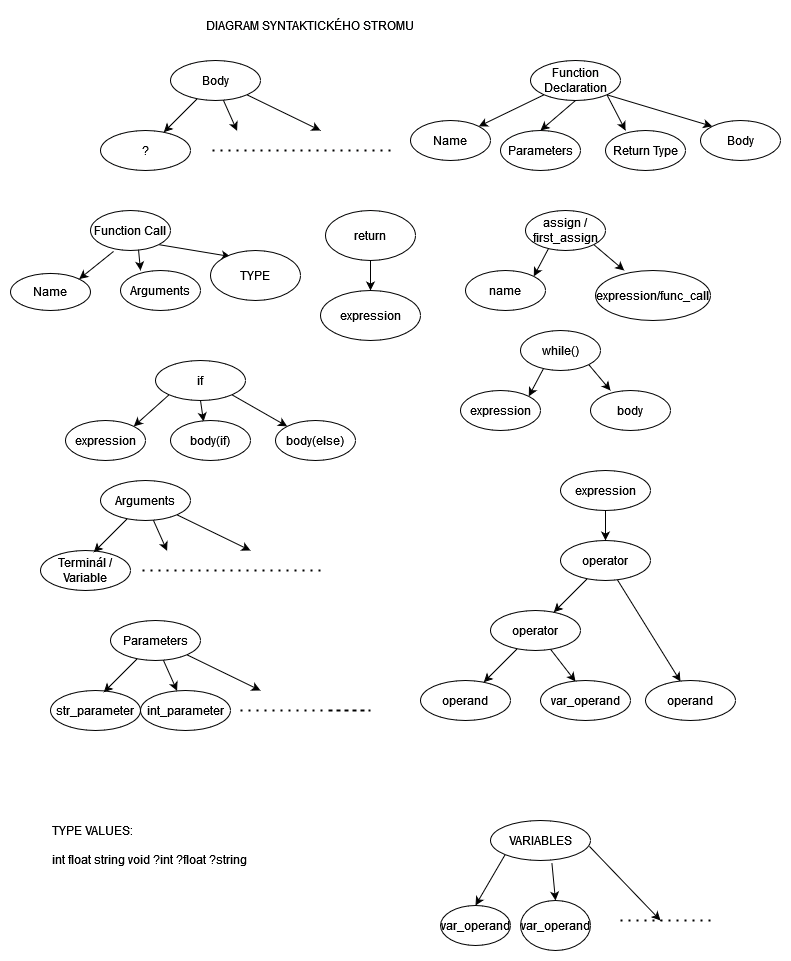
## Sémantická analýza

Sémantická analýza je rozdělena do dvou částí.

První část kontroluje nedefinované funkce a pokus o redefinici funkcí, kontrolu počtu parametrů při volání funkce a počet výrazů při návratu z funkce.

Řešení je použití tabulky symbolů. Analýza používá dva zásobníky tabulek symbolů. První ze zásobníků slouží pro uložení záznamů o definicích funkcí (návratový typ, počet a typ parametrů). Naplnění tabulky funkcí se vykoná v prvním průchodu abstraktního syntaktického stromu (dále AST).

Druhá tabulka symbolů slouží pro záznam proměnných v jednotlivých rámcích zdrojového kódu. Při druhém průchodu AST sémantická analýza vytvoří rámce tabulky symbolů, do kterých se ukládají záznamy o použitých proměnných. Informace o proměnných jsou poté uloženy do AST. První rámec zaznamenává hlavní tělo programu, další rámce jsou lokální pro dané funkce. Lokální rámec existuje jen dokud je zpracovávané tělo funkce, při návratu do hlavního těla je uvolněn a aktivním rámce je rámec hlavního těla.

Druhá část zabezpečuje kontrolu nedefinovaných proměnných, správných typů argumentů funkcí a typů návratových hodnot. Protože tato část nemůže kontrolovat věci za běhu programu, je tato povinnost přenesena na modul generování kódu.

Obrázek : Diagram AST

## Generování kódu

Generátor kódu je samostatný modul, jenž vytvoří zdrojový kód na základě AST, který je předán syntaktickou analýzou. Při generování kódu je již předpokládáno, že AST je validní až na chyby kontrolované za běhu programu.

* Proměnné – na začátku hlavního těla jsou definovány, kvůli dynamické kontrole inicializace před použitím ve výrazu. Uživatelské proměnné jsou definovány jako proměnné momentálního lokálního rámce. Pro potřeby překladu bylo vytvořeno několik pomocných globálních proměnných.
* Výrazy – implementace výrazů pomocí postupného postorder procházení AST. Operandy jsou přidány do zásobníku a operátory použity v zásobníkovém tvaru.
* Funkce – na začátku funkce je vytvořen vlastní lokální rámec pro lokální proměnné. Vestavěné funkce jsou některé předdefinovány na začátku programu, u funkcí, kde byla možnost, jsou implementovány stylem volání inline funkce jazyka C. Argumenty funkcí jsou předávány přes zásobník.
* Cykly a podmínky – pro potřeby zajištění unikátnosti názvu návěští má každý výskyt podmínky nebo cyklu přiřazen unikátní číslo, které je součástí jejich návěští.

# Použité algoritmy a struktury

## Dynamický řetězec

Struktura řetězců (strings.c/h) obsahuje pole znaků, délku pole a velikost pole v paměti. Struktura je použita pro práci s lexémy, kdy v modulu Scanner skládá znaky za sebe a tím tvoří konečnou hodnotu Tokenu.

## Zásobníky

Překladač používá pro překlad zásobníky (token\_stack.c/h) tokenů a zásobník tabulek symbolů (symtable.c/h) .

Zásobníky tokenů se používají pro průchod rekurzivním sestupem, kdy se tokeny skládají za sebe pro jednoduší kontrolu syntaxe, do stejného zásobníku se poté ukládají i výrazy v postfixové notaci.

Zásobníky tabulek symbolů jsou použity v rámci sémantické analýzy.

## Tabulka symbolů

Tabulka symbolů (symtable.c/h) slouží pro ukládání informací o funkcích a proměnných v sémantické kontrole. Je implementována jako tabulka rozptýlených prvků. Pro práci s tabulkami symbolů je tvořen zásobník ukazatelů na jednotlivé tabulky. Tabulka sestává z mapovacího pole a zřetězených listů záznamu. Každý záznam obsahuje odkaz na data a odkaz na další záznam ve zřetězeném listu. Struktura dat sestává z klíče a odkazu na strukturu, obsahující informace spojené s proměnnou nebo funkcí. O funkcích se ukládá počet a typ parametrů a typ návratové hodnoty. O proměnných jejich datový typ, což není využito z důvodu kontroly typu za běhu programu. Tabulka symbolů nám slouží pro kontrolu nedefinovaných/redefinovaných funkcí, nedefinovaných proměnných a kontrolu počtu argumentů při volání funkce.

## AST

Abstraktní syntaktický strom (syntax\_tree.c/h) se vytváří v průběhu syntaktické analýzy, hlavní část se tvoří v rámci rekurzivního sestupu, výrazy poté v rámci precedenční analýzy v postfixové notaci.

# Rozdělení práce

Podle jednotlivých částí byla rozdělena práce na:

* Lexikální analýza – xsouku15
* Syntaktická analýza – xpavli95
* Sémantická analýza – xvadov01
* Generátor kódu – xnekul04

## Podrobnější rozdělení

* xsouku15 – lexikální analýza, Strings, dokumentace
* xpavli95 – syntaktická analýza, vedení týmu, správa repozitáře
* xvadov01 – sémantická analýza, symtable
* xnekul04 – generování kódu, symtable, struktura AST

# Závěr

V rámci řešení projektu se dbalo hodně na komunikaci. Pravidelné scházení online a téměř denní konverzace o správnosti našeho řešení. Rozdělení práce proběhlo už při první schůzce, takže všichni věděli ihned co mají dělat. V rámci verzování byl použit GitHub. Verzování probíhalo přes žádosti o změny, každá žádost byla zkontrolován před spojením s hlavní větví.

# Zdroje

Pro generování LL-tabulky byl použit nástroj <https://jsmachines.sourceforge.net/machines/ll1.html>