VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Formálne jazyky a prekladače – Projekt

# Prekladač pre jazyk ZIG

Tým xluptas00 varianta TRP-izp

**Bodové rozdelenie:** 25/25/25/25%
**Implementované rozšírenia:** žiadne

25. novembra 2024

**Vedúci:** Samuel Lupták (xluptas00)
Petr Nemec (xlogin00)
Lukáš (xlogin00)
Mário Klopan (xlogin00)

# 1 Návrh

Prekladač sa skladá z 3 hlavných častí a 4 pomocných dátových štruktúr.
**Hlavné časti** prekladaču (a ich podčasti):

- Lexikálny analyzátor [1]
- Dvojprechodový syntaktický a sématický analyzátor [2]
  - Analýza kódu pomocou rekurzívneho zostupu
  - Analýza výrazov pomocou precedenčnej analýzi
- Generátor výsledného kódu

**Pomocné štruktúry** použité v prekladači:

- Tabulka s rozptýlenými položkami s implicitným zreťazením položiek [3]
- Abstraktný syntaktický strom [4]
- Zásobník
- Fronta

**Využitie** jednotlivých štruktúr je nasledovné:

*Hašovacia tabulka:* Bola použítá pre implementáciu tabulky symbolov. Podmienka pre implicitné zreťazenie nám robila mierny problém, pretože teoreticky nekonečný počet identifikátorov sa nemestí do konečne velkej tabulky

*ASS:* Slúži na komunikáciu medzi parserom a generátorom kódu

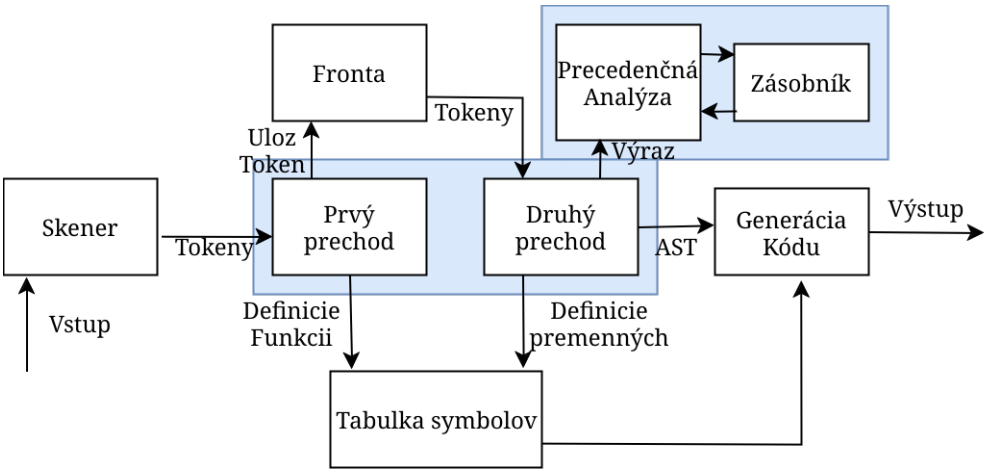
*Zásobník:* Je využitý precedenčnou analýzou, ktorá ho používa na spracovanie výrazov

*Fronta:* Má význam pri dvojprechodovej analýze ako uložisko tokenov. Pre dvojprechodovú analýzu sme sa rozhodli po zistení, že definicia funkcie nemusí lexikálne predchádzať jej volaniu.

[1] Ďalej len *skener*
[2] Ďalej len *parser*
[3] Ďalej len *hašovacia tabulka*
[4] Ďalej len *ASS*

# 2 Popis komunikácie

## 2.1 Diagram




## 2.2 Stručný popis

Bolo by vhodné začať tým, že parser (modrá časť diagramu) inicializuje všetky operácie v prekladači. Preklad začína vložením programu v jazyku zig na vstup. Skener (Na zavolanie) postupne fragmentuje vstup na jednotlivé lexémi a posiela ich vo forme tokenov do parseru. Ako bolo už spomenuté, tak parser je dvojprechodový. Tokeny idú najprv cez prvý prechod, ktorý kontroluje syntax a sematiku iba pre hlavičky funkcií, ktoré následne ukladá do tabulky symbolov, aby informácie o nich boli dostupné v druhom prechode. Prvý prechod ukladá všetky prečítané tokeny do fronty. Z fronty si tokeny po jednom berie druhý prechod, ktorý kontroluje syntax a sématiku pre ostatok kódu. V prípade že v kóde sa nachádza výraz, zavolá sa precedenčná analýza ktorá tento výraz s pomocov zásobníku spracuje. Druhý prechod zároveň pridáva definované premenné do tabulky symbolov (Samotnú tabulku symbolov však aj využíva, napr. pre kontrolu redefinície). Počas druhého prechodu sa zároveň vytvára ASS ktorý po úspešnom dokončení analýzi slúži ako výstup a zároveň vstup do generátoru kódu. Generátor kódu s pomocou tabulky symbolou generuje cieľový kód.

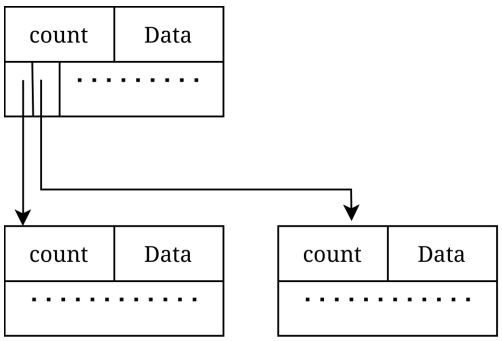
# 3 Implementácia

**Skener:** *lexer.*, token.h*(xlogins)
**Parser:** *syntax.*, queue_fill.*, precedence.** (xluptas00, xlogins)
**Generátor kódu:** *code_gen.**(xlogins)
**Hašovacia tabulka:** *symtable.**(xluptas00)
**ASS:** *tree.**(xlogins)
**Zásobník:** *stack.**(xlogins)
**Fronta:** *queue.** (xluptas00)
**Ostatné časti:** *error.** (xluptas00)

Implementácia dátových štruktúr sa nachádza v jednotlitvých súboroch pomenovaných podla danej štruktúry.

Implementácia častí samotného prekladača spočíva v súboroch skeneru, parseru, generátoru a súboru error.*, ktorý implementuje základné pracovanie s chybami. Prvý prechod je implementovaný v súboroch queue_fill.* a druhý prechod je implementovaný v súboroch syntax.*. syntax.c zároveň obsahuje funkciu Main. Prílohy A, B, C, D ukazujú využitú teóriu, ktorá slúžila ako podklad pre jednotlivé časti prekladaču.

## 3.1 Strom




Pre kompletnosť sme sa rozhodli vizualizovať ASS, ktorý sme navrhli pre tento prekladač. Ako je vidno na diagrame, tak každý uzol obsahuje 3 hlavné časti a to sú: *count, data, children*. Kde *count* určuje počet detí ktoré daný uzol má. Hlavná časť, *data*, v sebe uchováva data potrebné na správne generovanie kódu (viac viď. tree.*). Posledná časť *children* je pole ukazovať elov na deti. Pre konkrétne prípady ako sa strom využíva viď. syntax.c alebo precedence.c

## 3.2 Neštandardné riešenia

**Veľkosť tabulky symbolov:** Jediná implementačná anomália v rámci dátových štruktúr je statické obmedzenie veľkosti tabulky symbolov na 1000 položiek z dôvodu požadovanej implementácie pomocou implicitného zreťazenia položiek. Veľkosť 1000 bola zvolená na základe informácií, že viac ako 1000 položiek sa v žiadom teste nebude nachádzať. Uvedomujeme si že omnoho lepšie riešenie by bola tabulka symbolov s explicitným zreťazením položiek.
**Úroveň zanorenia v tabulke symbolov:**

# 4 Práca v tíme

# 5 PRÍLOHA A - Konečný automat

# 6 PRÍLOHA B - LL1 gramatika

# 7 PRÍLOHA C - LL1 tabulka

# 8 PRÍLOHA D - Precedenčná tabulka