

FAKULTA INFORMAČNÝCH TECHNOLOGIÍ VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Projektová dokumentácia Implementácia prekladača imperatívneho jazyka IFJ19

Tým 033, varianta I

Maroš Geffert	<xgeffe00>	25 %
Patrik Tomov	<xtomov00>	25 %
Martin Valach	<xvalac14>	25 %
Andrej Pavlovič	<xpavlo00>	25 %

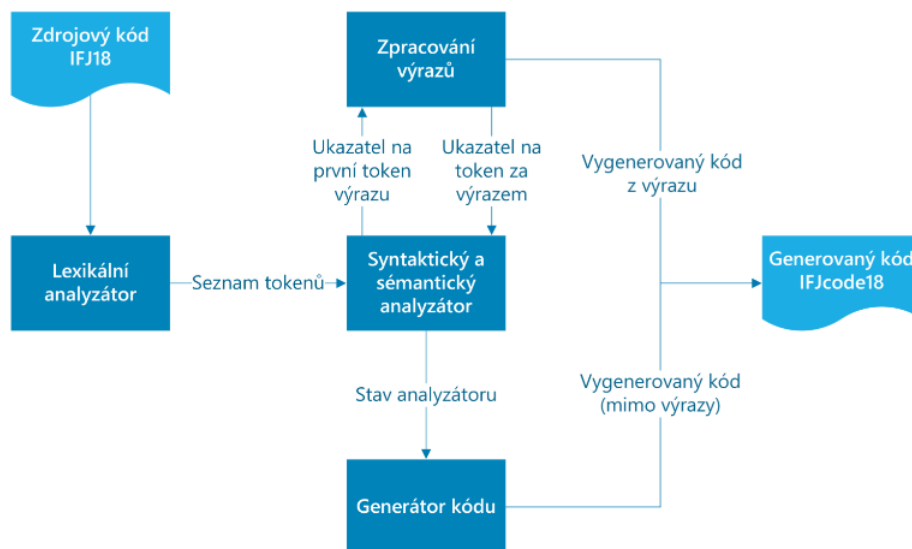
25. listopadu 2019

Obsah

1 Úvod

Cieľom nášho projektu bolo vytvoriť prekladač v jazyku C, ktorý načíta zdrojový súbor zapísaný v zdrojovom jazyku IFJ19, ktorý reprezentuje podmnožinu jazyka Python 3, a preloží ho do cieľového jazyka IFJcode19. V prípade chyby vracia odpovedajúci chybový kód.

2 Implementácia



2.1 Lexikální analyzátor

Pri programovaní prekladača, sme začínali s tvorbou lexikálneho analyzátoru. Základnou funkciou tejto časti je funkcia `get_token`, vďaka ktorej je možné načítavať lexémy¹ zo zdrojového súboru a prevádzajú sa do podoby takzvanej token. token je štruktúra ktorá obsahuje (atribúty, typy, v prípade identifikátoru si uchováva aj jednotlivý string atď.). Typy tokenov môžu byť kľúčové slová, identifikátor, rôzne operátory, EOL, EOF a všetko čo jazyk IFJ19 podporuje. Jednotlivé atribúty sa k tokenom priradzujú podľa situácie (napr. pre identifikátor, alebo čísla (float, integer)).

Lexikálny analyzátor funguje ako deterministický konečný automat. Tento konečný automat sme implementovali ako opakujúci sa switch, kde každý case odpovedá nejakému stavu z automatu. Ak sa načítaný lexem nezhoduje so žiadnym odpovedajúcim stavom určeným jazykom IFJ19 tak vraciame chybu 1. Lexikálna analýza končí načítaním posledného znaku zo zdrojového súboru, ktorým je EOF (Prípadne skončí v chybovom stave pri internej chybe).

¹Lexéma je základná jednotka lexikálnej (čiže slovnej) zásoby jazyka, t.j. základná jazyková jednotka, ktorá je nositeľom vecného významu.

2.2 Syntaktická analýza

Najpodstatnejšou časťou prekladača je syntaktická analýza tzv (parser). Parser je založený na rekurzívnom zostupe z hora dolu. Jeho úlohou je kontrola, či reťazec tokenov reprezentuje syntakticky správne napísaný program. Pokiaľ dojde pri jeho činnosti k chybe, vypíše sa kód chyby. Parser žiada tokeny od scanneru, kde potom na základe tokenov pracuje s tabuľkou symbolov.

Dvojprechodová

- Naplnenie tabuľky symbolov funkciami
- Samotná analýza kódu

Rekurzívny zostup

- Derivácia neterminálu na zásobník
- Precedenčná LL(1) tabuľka

2.3 Spracovanie výrazov

Dalšou časťou prekladača je sémantický analyzátor. Pri sémantickej analýze sa kontrolujú operácie nad dátovými typmi a práca s nimi (napr. ich pretypovanie). Jedná sa buď o implicitnú konverziu alebo ošetrovanie nepovoľeného pretypovania. Všetky kontroly sa robia na základe dát, uložených v derivačnom strome, a to pri prechode stromom zdola horu.

2.4 Generovanie kódu

Generovanie kódu pre nás znamená vytváranie medzikódu IFJcode19. Kód je generovaný na štandardný výstup. Po dokončení všetkých analýz, pričom jednotlivé časti sú generované v priebehu analýz do internej pamäti programu. Generátor vnútorného kódu na základe naplnenej tabuľky symbolov generuje trojadresný vnútorný kód k spracovaniu interpretom. Tento trojadresný kód sa generuje kvôli zjednodušeniu ďalšej práce a zjednodušeniu činnosti interpretu.

2.5 Interpret

Interpret vytvára interpretáciu trojadresného kódu, generovaného syntaktickým analyzátorom. Trojadresný kód je uložený v instrukčnom liste, ktorý je implementovaný pomocou jednosmerného viazaného lineárneho zoznamu. Pre každú inštrukciu je tento kód reprezentovaný typom inštrukcie, adresou výsledku a adresami prvého a druhého operandu. Pri spracovávaní aritmeticko-relačných operácií robí interpret sémantickú kontrolu.

3 Algoritmy

3.1 Hash table

Hašovacia tabuľka je údajová štruktúra, ktorá asociuje kľúče s hodnotami. Primárna efektívne podporovaná operácia je vyhľadávanie.

3.2 Dynamický string

Ďalší algoritmus, ktorý sme použili je `Lexem_string` pre operácie s reťazmi rôznej dĺžky. Štruktúra má v sebe uložené ukazateľ na reťazec, dĺžku reťazca a aj kolko pamäte je vyhradené pre reťazec. Implementované operácie : inicializácia (Pri inicializácii sa vyhradí pamäť práve pre určitý počet znakov a ak je potrebné tak sa realokuje), uvoľnenie dát, pridanie znaku alebo stringu do reťazca atd.

3.3 Tabuľka priorít

3.4 Tabuľka symbolov

Tabuľka symbolov predstavuje strom funkcií a ukazateľ na práve používanú funkciu. Každá funkcia obsahuje strom premenných, zoznam konštánt a zoznam inštrukcií.

4 Práca v tíme

Náš tím sa stretol po zadaní projektu. Vedúcim tímu boli hneď aj rozdelené úlohy. Tím väčšinou medzi sebou komunikoval osobne alebo cez discord.

Najprv sme si stanovili štruktúru celého projektu. V priebehu prvých šiestich týždňov sa toho veľmi neurobilo, implementoval sa lexikálny analyzátor, členovia študovali a zbierali informácie ohľadom svojej časti programu, ktorá im bola zadaná vedúcim tímu. V priebehu ďalších týždňov sme implementovali celý program. Testovali sme jednotlivé časti osobitne, ale neskôr sme to už testovali spolu ako celok. K zdieľaniu kódu sa používal repozitár git.

Meno	Login	Rozdelenie práce
Maroš Geffert	xgeffe00	Lexikálna analýza, dokumentácia, testy
Patrik Tomov	xtomov02	Syntaktická analýza
Martin Valach	xvalac14	Sémantická analýza
Andrej Pavlovič	xpavlo00	Generátor kódu

5 Záver

Projekt nás až tak nezarazil, vzhľadom nato, že sme sa naňho začínali pripravovať v celku dosť skoro. Tím sme si zostavili veľmi rýchlo a tým, že sme na rovnakom internáte tak nebol problém s akoukoľvek komunikáciou. Jednotlivé časti programu sme riešili z väčšej časti individuálne. Správnosť projektu sme si overili automatickými testami a pokusným odovzdaním, vďaka ktorému sme boli schopní projekt ešte viac doladiť.

Na tomto projekte sme si vyskúšali implementáciu niektorých zaujímavých dátových štruktúr, algoritmov, teóriu formálnych jazykov v praxi a spoluprácu v tíme.

6 Digramy

6.1 Deterministický konečný automat popisující lexikální analýzu

