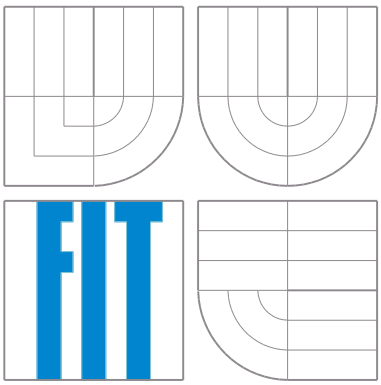


Formální jazyky a překladače

Implementace interpretu imperativního jazyka IFJ12

Tým 027, varianta b/2/I

7. prosince 2012

Rozšíření: MINUS
LOOP
LOGOP

Autor: Radek Ševčík, xsevci44@stud.fit.vutbr.cz (vedoucí) : 100%
Daniel Berek, xberek00@stud.fit.vutbr.cz : 0%
Dušan Fodor, xfodor01@stud.fit.vutbr.cz : 0%
Pavol Jurčík, xjurci06@stud.fit.vutbr.cz : 0%
Peter Prítel, xprite01@stud.fit.vutbr.cz : 0%

Fakulta Informačních Technologií
Vysoké Učení Technické v Brně

Obsah

1. Úvod

Tento dokument popisuje analýzu a implementační dokumentaci společného projektu dvou předmětů *IFJ (Formální jazyky a překladače)* a *IAL (Algoritmy)* - ***Implementace interpretu imperativního jazyka IFJ12***.

Cílem projektu je vytvořit interpret skriptovacího jazyka IFJ12[1], který byl pro tento účel stvořen a je inspirován jazykem Falcon[2].

V rámci formálních jazyků projekt popisujeme lexikální, syntaktickou a sémantickou analýzou a interpretem.

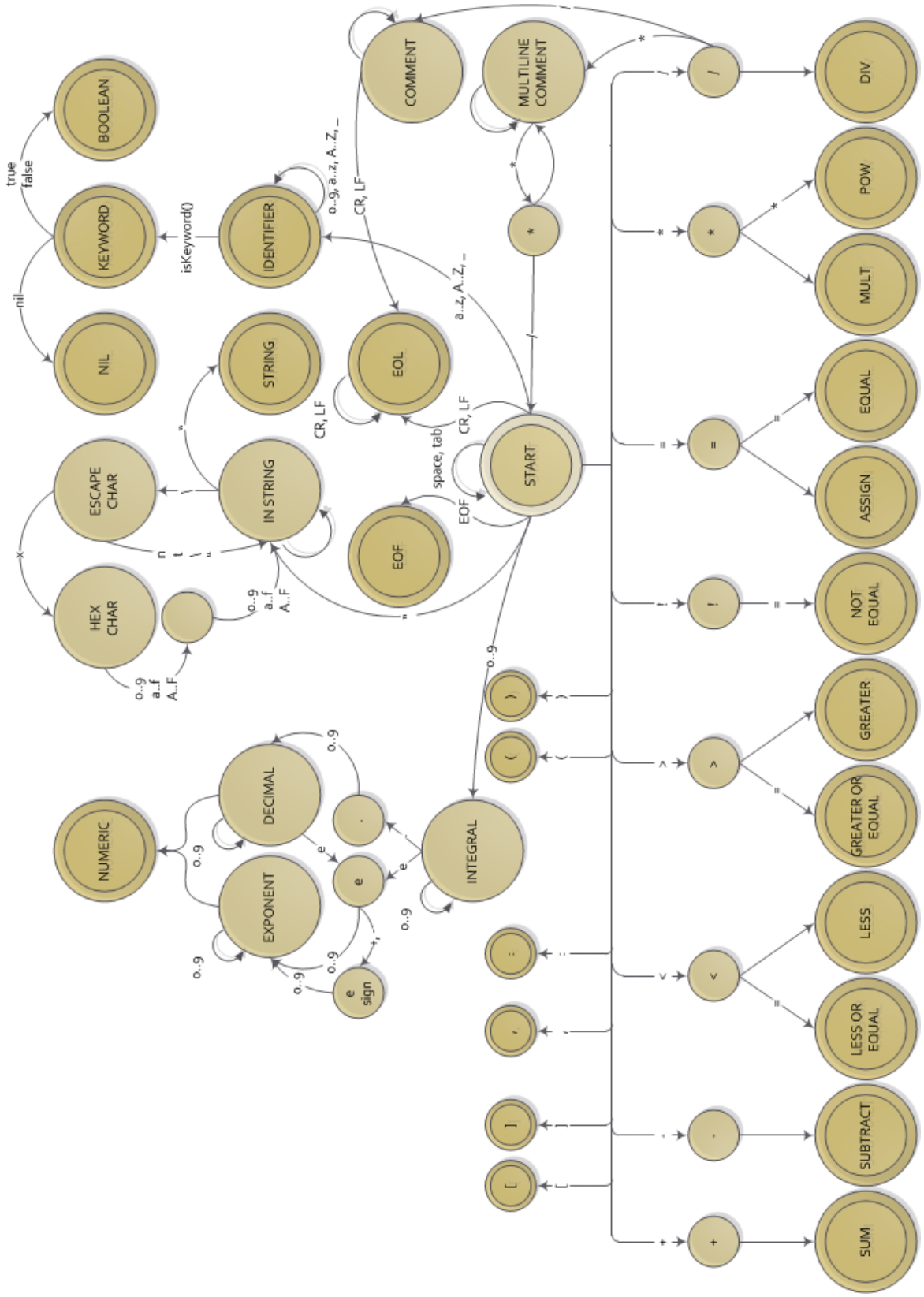
Z varianty projektu je pro vyhledávání podřetězců zadaný Boyer-Mooreův algoritmus, pro řadící funkci heap-sort a pro tabulku symbolů binární vyhledávací strom.

Z možností rozšíření interpretu se budeme věnovat unárnímu operátoru mínus, cyklu loop a logickým operátorům.

[1] https://wis.fit.vutbr.cz/FIT/st/course-files-st.php/course/IFJ-IT/projects/ifj2012.pdf
[2] http://falconpl.org/

2. Lexikální analýza

Činností lexikálního analyzátoru je přečíst znak po znaku ze vstupního soubor a vytvořit z nich posloupnost tokenů, které jsou definovány typem a atributy. Implementaci tvoří konečný automat.



3. Syntaktická analýza

Syntaktická analýza úzce pracuje s lexikálním analyzátorem. Na vyžádání dostane právě jeden token a na základě gramatiky ověřuje posloupnosti těchto tokenů a vytváří vstup pro sémantickou analýzu – abstraktní syntaktický strom. Pro rozebrání příkazů použijeme LL analýzu a pro výrazy precedenční analýzu.

3.1. LL analýza

LL analýza konstruuje nejlevější derivaci pro vstup zleva, je to analýza typu shora-dolů. Naimplementována je rekurzivně, kde pro každý neterminál existuje zpracovávající funkce. Pro neterminál <expr> je zavolána precedenční analýza.

Nadefinujeme si její gramatiku:

1 <program> → <function-or-stmt-list>

2 <opt-expr> → <expr>
3 <opt-expr> → ε

4 <end-stmt> → end EOL
5 <end-stmt-opt-expr> → end <opt-expr> EOL

6 <if-stmt> → if <expr> EOL <stmt-list>
7 <else-stmt> → else EOL <stmt-list>
8 <if-else-stmt> → <if-stmt> <else-stmt> <end-stmt>

9 <loop-stmt> → loop EOL <stmt-list> <end-stmt-opt-expr>
10 <while-stmt> → while <expr> EOL <stmt-list> <end-stmt>
11 <return-stmt> → return <expr> EOL
12 <assign-stmt> → id = <expr> EOL

13 <stmt> → <if-else-stmt>
14 <stmt> → <loop-stmt>
15 <stmt> → <while-stmt>
16 <stmt> → <return-stmt>
17 <stmt> → <assign-stmt>

18 <function-or-stmt-list> → <stmt> <function-or-stmt-list>
19 <function-or-stmt-list> → <function> <function-or-stmt-list>
20 <function-or-stmt-list> → ε
21 <stmt-list> → <stmt> <stmt-list>
22 <stmt-list> → ε

23 <function> → function id (<param-list>) EOL <stmt-list> <end-stmt>
24 <param-n> → , id <param-n>
25 <param-n> → ε
26 <param-list> → id <param-n>
27 <param-list> → ε

3.2. Precedenční analýza

Precedenční analýza je jednoduchá analýza zdola-nahoru implementovaná pomocí stacku a precedenční tabulky.

Z LL gramatiky je jasné, že veškeré výrazy jsou zpracovány precedenční analýzou, tudíž i volání funkcí a výběr podřetězce. Toto rozhodnutí nám umožňuje mít právě tyto dvě zmíněné funkce jako součást výrazu. Mimo jiné mohou být také výrazy součástí parametrů funkce.

Pro zjednodušení parsování podřetězců však jejich analýzu přenecháme LL gramatikám.

<opt-num> → numeric
<opt-num> → ε
<substr> → i [<opt-num> : <opt-num>]

Uvedeme si gramatiku, precedenci a asociativitu použitých operátorů.

E → i()
E → i(E)
E → i(L)
L → L, E
L → E, E
E → (E)
E → E + E
E → E - E
E → E * E
E → E / E
E → E ** E
E → E and E
E → E or E
E → -E
E → not E
E → E > E
E → E >= E
E → E < E
E → E <= E
E → E == E
E → E != E
E → E in E
E → E notin E
E → i

priorita	operace	popis	asociativita
1	()	závorky	zleva
2	()	volání funkce	
3	-	unární mínus	zprava
	not	logická negace	
4	**	umocnění	zleva
5	*	násobení	zleva
	/	dělení	
6	+	sčítaní	
	-	odčítání	
7	<	menší	
	<=	menší, rovno	
	>	vetší	
	>=	vetší, rovno	
	in	je v	
	notin	není v	
8	==	rovná se	
	!=	nerovná se	
9	and	logický and	
10	or	logický or	
11	,	operátor čárka	

pozn.: Obvykle má v matematice operace umocňování prioritu zprava.
Unární mínus je vyhodnocen způsobem, kdy tokenu mínus nepředchází zakončovací závorka anebo term.
Oproti zadání má operace =, ≠ menší prioritu, než operace < ≤ > ≥
Operace in a notin je u řetězců „syntaktický cukr“ pro vestavěnou funkci find.

Precedenční tabulka

STACK \ VSTUPNÍ TOKEN	$	+	-	×	÷	^	i	(	)	,	and	or	not	in	notin	<	≤	>	≥	≠	=	-i
$	$	<	<	<	<	<	<	<			<	<	<	<	<	<	<	<	<	<	<	<
+	>	>	>	<	<	<	<	<	>	>	>	>	<	>	>	>	>	>	>	>	>	<
-	>	>	>	<	<	<	<	<	>	>	>	>	<	>	>	>	>	>	>	>	>	<
×	>	>	>	>	>	<	<	<	>	>	>	>	<	>	>	>	>	>	>	>	>	<
÷	>	>	>	>	>	<	<	<	>	>	>	>	<	>	>	>	>	>	>	>	>	<
^	>	>	>	>	>	>	<	<	>	>	>	>	<	>	>	>	>	>	>	>	>	<
i	>	>	>	>	>	>		=	>	>	>	>	>	>	>	>	>	>	>	>	>	>
(		<	<	<	<	<	<	<	=	<	<	<	<	<	<	<	<	<	<	<	<	<
)	>	>	>	>	>	>			>	>	>	>	>	>	>	>	>	>	>	>	>	>
,		>	>	>	>	>	<	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>
and	>	<	<	<	<	<	<	<	>	>	>	>	<	<	<	<	<	<	<	<	<	<
or	>	<	<	<	<	<	<	<	>	>	<	>	<	<	<	<	<	<	<	<	<	<
not	>	>	>	>	>	>	<	<	>	>	>	>	<	>	>	>	>	>	>	>	>	<
in	>	<	<	<	<	<	<	<	>	>	>	>	<	>	>	>	>	>	>	>	>	<
notin	>	<	<	<	<	<	<	<	>	>	>	>	<	>	>	>	>	>	>	>	>	<
<	>	<	<	<	<	<	<	<	>	>	>	>	<	>	>	>	>	>	>	>	>	<
≤	>	<	<	<	<	<	<	<	>	>	>	>	<	>	>	>	>	>	>	>	>	<
>	>	<	<	<	<	<	<	<	>	>	>	>	<	>	>	>	>	>	>	>	>	<
≥	>	<	<	<	<	<	<	<	>	>	>	>	<	>	>	>	>	>	>	>	>	<
≠	>	<	<	<	<	<	<	<	>	>	>	>	<	<	<	<	<	<	<	>	>	<
=	>	<	<	<	<	<	<	<	>	>	>	>	<	<	<	<	<	<	<	>	>	<
-i	>	>	>	>	>	>	<	<	>	>	>	>	<	>	>	>	>	>	>	>	>	<

4. Sémentická analýza

Veškerá sémantická kontrola je prováděna až za běhu programu, tj. v interpretu. Tato kontrola testuje existenci proměnných, funkcí, použití názvu proměnné jako název funkce, kontrolu operací nad datovými typy, aj.

5. Interpret

Vstupem interpretu je abstraktní syntaktický strom, nemusíme proto vytvářet a generovat mezikód ani díky rekurzivnímu volání používat stack. Po sobě jdoucí příkazy jsou v AST ve struktuře lineárně vázaného seznamu, které jsou procházeny a vykonávány v cyklu. Narazí-li interpret na příkaz větvení nebo cyklu, zavolá se funkce eval, která vrátí boolean hodnotu složeného výrazu, podle které se rozhodne kterou větví pokračovat.

5.1. Správa paměti

Interpret si vytvoří lineárně vázané seznamy pro tabulky symbolů a hodnoty proměnných. Jakékoliv mezivýpočty (návratové hodnoty, konkatenace řetězců, aj.) se ukládají do tabulky hodnot proměnných a uvolní se teprve až na konci programu.

5.2. Volání funkcí

První tabulka symbolů je interně „globální“, protože obsahuje odkazy do AST, avšak neobsahuje žádné globální proměnné. Pokud výraz obsahuje volání funkce, vyhledá její „deklaraci“ v globální tabulce, funkcí eval vypočítá předávané parametry a vloží je do nově vytvořené tabulky symbolů. Následně započne rekurzivní interpretace instrukcí ve funkci. Pokud obsahuje funkce příkaz return, vloží se hodnota výrazu do tabulky symbolů a interpretace skončí. Implicitně funkce vrací nil.

5.3. Předdefinované funkce

Tzv. „built-in“ funkce jsou také definovány v globální tabulce symbolů, avšak nemají žádnou adresu. Jejich rozpoznání proběhne těsně před voláním. V počátku byly built-in funkce rozpoznány už v lexikální analýze, avšak z tohoto návrhu sešlo, kvůli komplikacím se sémantickou kontrolou.

6. Algoritmy

Hojně je v programu využíván stack s daty proměnné délky, samorealokovatelný řetězec, lineárně vázané seznamy a nakonec binární vyhledávací strom.

Avšak heap-sort a vyhledání podřetězce naimplementován není, namísto toho jsou použity funkce qsort a strstr knihovny jazyka C.

7. Rozšíření

MINUS – unární mínus je vysvětleno v kap. precedenční analýzy (str. 6).
LOOP – jediný rozdíl mezi while a loop je v možnosti vynechat výraz (implicitně true) a také, že se výraz vyhodnotí až na konci.
LOGOP – priorita a asociativita operátorů AND, OR, NOT je v kap. precedenční analýzy (str. 6).

8. Závěr

Týmová spolupráce

POUŽITÉ PROSTŘEDKY:

- Verzovací systém GIT na github.com (po domluvě možnost nahlédnutí do repositáře)
- Jabber na vutbr.cz, školní mail
- Osobní setkání

Po emailu bylo doporučeno pracovat ve dvojicích, byl nastaven interní deadline týden před odevzdáním, pro možnost dopsat dokumentaci a odladit chyby. Snahou bylo odevzdat v předtermínu s některými rozšířeními, proto již začátkem října byl založen repositář a členové týmu byli obeznámeni se systémem git (spousta odkazů na nastavení účtu na githubu, odkaz na pěkně zpracovanou knížku v češtině popisující základní příkazy gitu, jak použít ssh-keygen, který klient stáhnout pokud používají windows, aj.)

ROZDĚLENÍ PRÁCE:

- Dušan Fodor, xfodor01 – stack, parser, sort
- Pavol Jurčík, xjurci06 – tab. symbolů, find, interpret
- Peter Prítel, xprite01 – tab. symbolů, find, interpret
- Radek Ševčík, xsevci44 – scanner, main, ladící funkce, rozhraní pro moduly, analýza

VÝSLEDEK:

- Radek Ševčík, xsevci44 – scanner, parser, interpret, dokumentace, aj.

Neimplementované algoritmy

Jelikož mám předmět IAL absolvovaný, po domluvě s vedením nemusím implementovat algoritmy sort a find a mohu použít funkce z knihovny jazyka C.

Co mi projekt přinesl

Naučil jsem se pro mě dosud neznámého syntakt. cukru jazyka C99 i preprocesoru.
Nahlédl jsem do problémů s organizací vícečlenných týmů.
Pochopil jsem, že čím více se deadline blíží, tím více jsou komentáře zapotřebí.
Zde získané zkušenosti mohu využít i v jiných projektech, zejména ve strukturovaných konfiguračních souborech nebo při jednoduché interpretaci příkazů jako součásti většího programu, kde je např. připojení plnohodnotného interpretu lua plýtváním prostředků.
Bohatá praktická příprava na semestrální zkoušku ☺

Co jsem udělal za chyby

Zejména nedomyšleným ukládáním tokenů na zásobník (zapomněl jsem atributy) v precedenční analýze jsem se připravil o možnost přidat rozšíření FUNEXP, výsledkem je spousta práce s neúplnou nápravou tzn., že volání funkce může být součástí výrazu, ale ne obráceně.
Kvůli práci na „poslední chvíli“ není projekt moc rozšiřitelný o nové vlastnosti.

Metriky kódu
Počet řádků: 5419
Počet zdrojových souborů: 25

Literatura

- Přednášky předmětu IFJ
- Materiály k demonstračnímu cvičení předmětu IFJ
- Wikipedia contributors, *Operators in C and C++,* Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/wiki/Operators_in_C_and_C%2B%2B
- T.Niemann, *Operator Precedence Parsing*, ePaper Press http://epaperpress.com/oper/download/oper.pdf