VYSOKÉ UČENÍ TECHNICKÉ V BRNE

FAKULTA INFORMACNICH TECHNOLOGII

Dokumentácia projektu do predmetov IFJ a IAL

Interpret jazyka IFJ16

Tým 097, varianta a/1/I

Vedúci : Peter Grofčik: 25%

Rastislav Pôbiš : 25%

Patrik Krajč: 25%

Filip Kolesár: 25%

11.december 2016

Obsah

1 Úvod

Dokumentácia popisuje návrh a implementáciu projektu Interpret jazyka IFJ16 do predmetov IFJ (Formálne jazyky a prekladače) a IAL (Algoritmy). Rozhodli rozhodli pre variantu a/1/I , ktorá určuje, aký algoritmus máme pre daný problém využiť. Pre vyhľadávanie použijeme **Knuth-Morris-Prattuv** algoritmus, ktorý sme následne implementovali vo vstavanej funkcii **`ifj16.find`** .Pre radenie používame algoritmus **Quick sort,** ktorý sme následne implementovali vo vstavanej funkcii **ifj16.sort**. Posledné zo špecifikovaných pravidiel použitia, sme mali využiť v implementácii tabuľky symbolov **binární vyhledávací strom**.

2 Štruktúra projektu

Prekladač je rozdelený do troch hlavných celkov. Pomocou **lexikálneho analyzátoru** (scanner) sa načíta zdrojový kód a rozdelí na tokeny. **Syntaktický analyzátoru** zažiada lexikálny o token a overí syntax. Po bezchybnej kontrole sa spustí **Interpret** so sémantickou kontrolou jazyka.

3 Lexikálneho analyzátoru (scanner)

Analyzátor je implementovaný v v súboroch scanner.c a scanner.h. Lexikálny analyzátor načíta zdrojový kód po znakoch a prevedie ich na tokeny. Všetka komunikácia z ostatnými knižnicami prebieha volaním fukcie get_token, Analyzátor je implementovaný pomocou konečného automatu (príloha 1), pričom ignoruje všetky komentáre zdrojového kódu. Funkcia je volaná vždy, keď syntaktický analyzátor zažiada o token. Načítanie znaku zaisťuje funkcia fget(). Na základe načítaného lexému sa určí o aký typ tokenu sa jedná. V niektorých prípadoch je potrebné načítaný znak vrátiť do súboru a to pomocou funkcie unget().Token definuje jednotlivé prvky kódu a určuje ako sa k nemu má pri preklade pristupovať. Analyzátor taktiež pomocou funkcie compare_keywords(). zisťuje či nejde o rezervované kľúčové slovo pre špeciálnu funkciu.

4 Syntakticky analizator (parser)

Syntaktický analyzátor kontrolu syntaktickú správnosť zdrojového kódu. Postupne volá tokeny zo lexikálneho analyzátoru, ktorú sú spracováné dvoma spôsobmi a to pomocou LL-gramatiky (príloha 2) a precedenčnej syntaktickej analýzy (príloha 3), ktoré sú jadrom syntaktického analyzátoru. Pri úspešnom dokončení syntaktikej analýzy sa generuje inštrukčná páska pre interpret

5 Interpret

Poslednou časťou Interpretu je vlastne preklad, ktorý nastane iba ak zdrojový kód úspešne prejde cez lexikálny a syntaktická analyzátor. Interpretačná časť má za úlohu spracovať inštrukčnú sadu založenú na 3AC (trojadresnom kóde). Generovanie 3AC zabezpečuje parser a samotnú interpretáciu nájdeme v súboroch interpret.c a interpret.h.

6 Algoritmy z predmetu IAL

6.1 Radenie (Quick sort)

Radenie Quick sort sme využilie pri implementácii vstavanej funkcie ifj16.sort. Základnou myšlienkou Quicksort je rozdelenie radené postupnosti čísiel na dve približne rovnaké časti (Quicksort patrí medzi algoritmy typu rozdeľ a panuj). V jednej časti sú čísla väčšie a v druhej menšie, než nejaká zvolená hodnota (nazývaná pivot - anglicky "stred otáčania"). Ak je táto hodnota zvolená dobre, sú obe časti približne rovnako veľké. Ak budú obe časti samostatne zoradené, je zoradené aj celé pole. Obe časti sa potom rekurzívne radí rovnakým postupom, čo ale neznamená, že implementácia musí tiež použiť rekurziu.

6.2 Vyhľadávanie pod reťazca v reťazci (Knuth-Morris-Prattuv)

Podľa zadanie sme mali vstavanú funkciu ifj16.find() ktorá by využívala Knuth-Morris-Prattuv algoritmus. Spočíva v tom, že ak sa tieto informácie využijú správnym spôsobom, môže sa vzorka nad prehľadávaným textom posúvať aj o viac ako iba o jeden znak doprava. Tým sa významne skráti doba potrebná k prehľadávaniu textu. Tiež je zbytočné sa v prehľadávanom texte vracať k znakom, ktoré už boli analyzované tak ako to robí naivný algoritmus. Toto vrátenie spočíva v skutočnosti, že ak pri porovnávaní vzorky s daným textom narazím na nezhodu, vrátim sa späť na začiatok vzorky a ten posuniem o jedno miesto doprava. Táto činnosť je zrejme zbytočná, lebo ja už mám informáciu o predchádzajúcich znakoch, stačí ju len dostatočne využiť. Vrátenie sa v texte môže priniesť aj ďalší problém, ktorý nie je na prvý pohľad zrejmý. Pri spracovávaní dlhšieho textu, určite nie je tento text v pamäti počítača celý. Zo súboru sa načíta po kusoch do nejakého bufferu v pamäti, s ktorým sa potom pracuje. Funkcia je implementovaná podľa skript z predmetu IAL.

```
Funkcie KMP algoritmu:

mall_kmp()

kmp()

partition ()
```

6.3 Tabuľka symbolov

V našom zadaní sme implementoval tabuľku symbolov pomocou binárneho stromu. V našej implementácii binárny strom používa ako kľúč identifikátor premennej či funkcie a ich umiestnenie sa riadi lexikografickým porovnaním. Existujú tu globálne tabuľky a tabuľky lokálne pre každú funkciu. V nich sú obsadené premenné definované v danom kontexte. Ďalej pre každé volanie funkcie sa vytvára "inštancia" lokálne tabuľky, kde sa ukladajú hodnoty pre to konkrétne volanie. V prípade rekurzie teda bez konfliktov. Každý uzol premenné v tabuľke obsahuje jej identifikátor, dátový typ a hodnotu. U uzla funkcie potom hodnotou myslíme návratovú hodnotu a navyše tu sú informácie o type návratovej hodnoty, odkaz na list inštrukcií danej funkcie, zoznam parametrov a odkaz na tabuľku symbolov pre funkciu.

7 Testovanie

Testovanie prebiehalo priebežne . Navzájom sme si kontrolovali svoju priebežnú implementáciu a snažili sa odladiť všetky nedostatky, aby program vedel adekvátne vyhodnotiť všetky vstupy a aby správne reagoval na chybové stavy ako napríklad:

If 5 then

While(int<double)

Var;

8 Práca v tíme

Prácu sme si rozdelili približne nasledovne:

Peter Grofčik: binárny strom

Rastislav Pôbiš : interpret

Patrik Krajč: lexikálna analýza, syntaktická analýza

Filip Kolesár: syntaktická analýza

Pri riešení projektu sme sme postupovali spoločne, rozdelenie práce bolo často krát ovplinené vyťažením členov tímu. Pri pravidelných návštevách sme porovnávali a testovali naše aktuálne implementácie programu. Všetku našu prácu organizoval náš vedúci.

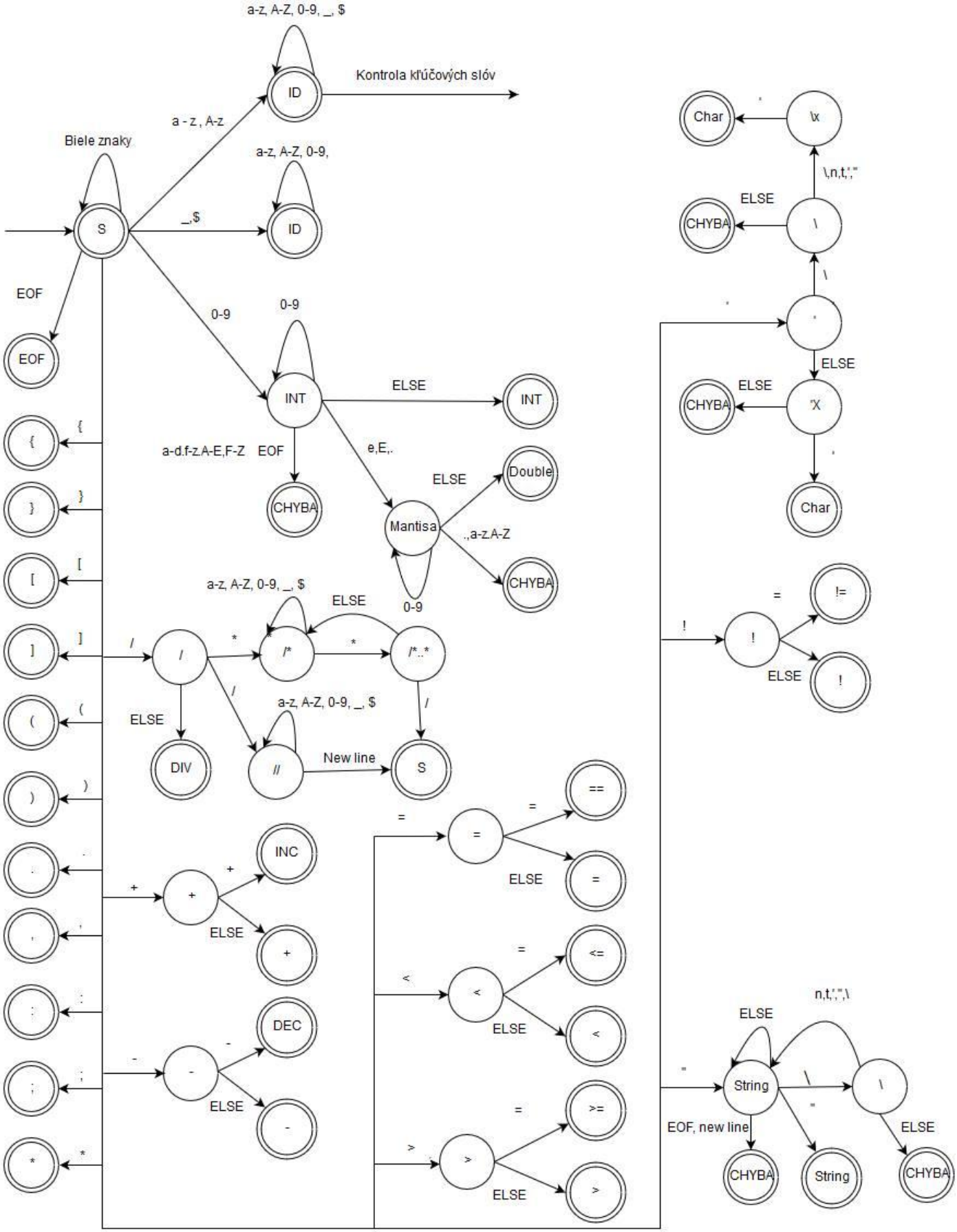
9 Záver

Projekt bol vyvíjaný po dobu jedeného mesiaca z toho posledné dva týždne intenzívnejšie aby program spĺňal všetky požiadavky dané zadaním.

10 Literatúra

Prof. Ing.Jan M Honzík,CSc. Algoritmy a datové struktury verze:16D

11 Prílohy


a-z, A-Z, 0-9, _, $
ID
Kontrola kľúčových slov
a - z , A-z
Biele znaky
a-z, A-Z, 0-9,
_,$
S
ID
EOF
EOF
0-9
0-9
INT
ELSE
INT
a-d.f-z.A-E,F-Z
EOF
e,E,.
CHYBA
ELSE
Double
Mantisa
.,a-z.A-Z
CHYBA
0-9
{
}
[
]
(
)
.
,
:
;
*
a-z, A-Z, 0-9, _, $
ELSE
/
/*
*
/*..*
/
ELSE
DIV
/
a-z, A-Z, 0-9, _, $
//
New line
S
+
INC
ELSE
+
-
DEC
ELSE
-
=
==
ELSE
=
<
<=
ELSE
<
>
>=
ELSE
>
Char
\x
\,n,t,',"
ELSE
CHYBA
\
\
'
ELSE
ELSE
CHYBA
'X
'
Char
!
=
!=
ELSE
!
n,t,',",\
ELSE
"
String
\
\
"
EOF, new line
ELSE
CHYBA
String
CHYBA

Príloha 1:Návrh konečného automatu

root> kw_class kw_main/is_id <class_body> EOF

<class_body> 1. static kw_int/kw_string/kw_double is_id char_bod_ciarka/char_rovnasa

2. static kw_int/kw_string/kw_double/kw_void is_is char_LZatvorka <func_arg> char_PZatvorka char_PMZatvorka <func_body>

3. char_PMZatvorka

<func_arg> 1. kw_int/kw_string/kw_double is_id

2. char_ciarka

3. char_PZatvorka

<func body> 1. kw_int/kw_string/kw_double is_id char_rovnasa/char_bod_ciarka

2. is_id char_rovnasa

2. is_id char_bodka is_id char_bod_ciarka/(char_rovnasa <expression_solve>)/(char_LZatvorka <func_params> char_bod_ciarka)

2. "ifj16" char_bodka kw_readInt/kw_readDouble/kw_length/kw_readString/kw_print/kw_substr/kw_compare/kw_find/kw_sort char_LZatvorka <func_params> char_bod_ciarka

2. is_id char_LZatvorka <func_params> char_bod_ciarka

3. kw_while char_LZatvorka <bool_expr> char_LMZatvorka <func_body>

3. kw_if char_LZatvorka <bool_expr> char_LMZatvorka <func_body> kw_else charLMZatvorka

4. kw_return

5. char_PMZatvorka

Príloha 3: LL gramatika

	+	-	*	/	(	)	<	>	<=	>=	==	!=	i	$
+	>	>	<	<	<	>	>	>	>	>	>	>	<	>
-	>	<	<	>	>	>	>	>	>	>	>	>	<	>
*	>	>	>	>	<	>	>	>	>	>	>	>	<	>
/	>	>	>	>	<	>	>	>	>	>	>	>	<	>
(	<	<	<	<	<	=	<	<	<	<	<	<	<	
)	>	>	>	>		>	>	>	>	>	>	>		>
<	<	<	<	<	<	>	>	>	>	>	>	>	<	>
>	<	<	<	<	<	>	>	>	>	>	>	>	<	>
<=	<	<	<	<	<	>	>	>	>	>	>	>	<	>
>=	<	<	<	<	<	>	>	>	>	>	>	>	<	>
==	<	<	<	<	<	>	<	<	<	<	<	<	>	>
!=	<	<	<	<	<	>	<	<	<	<	<	<	>	>
i	>	>	>	>		>	>	>	>	>	>	>		>
$	<	<	<	<	<		<	<	<	<	<	<	<	

Príloha 3: Tabuľka pre precedenčnú analýzu