VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Dokumentace k projektu do predmetu IFJ a IAL

Interpret imperativního jazyka IFJ14

Tým 083, varianta a/2/I

Výčet rozšíření:

MSG

BASE

Autori:

Fáber Jakub xfaber02.

Hladík Daniel xhladi21.

Marušič Marek xmarus05.

Surovčík Tomáš xsurov04.

14. prosince 2014

Obsah

1 Zadanie

Dostali sme za úlohu vyvtvoriť interpret jazyka IFJ14. Tento jazyk je podmnožinou jazyka pascal, ktorý je predovšetkým procedurálny jazyk. IFJ14 má navyše funkcie sort a find. Funkciu sort sme mali implementovať pomocou algoritmu Heap sort a na funkciu find sme mali pouziť Knuth-Moris-Prattov algoritmus. Tabuľku symbolov sme mali implementovat pomocou binarneho stromu.

2 Podrobnejší pohľad na zadanie a návrh implementácie

Našou úlohou bolo naprogramovat interpret pre jazyk IFJ14, ktorý bol vytvorený kvôli tomuto účelu. Prácu na projekte sme rozdelili na niekoľko menších častí, ktoré su podrobne popísane v nasledujúcej kapitole. Tieto časti sú veľmi úzko prepojené a bez akej koľvek časti by sme nemohli zaručiť funkčnosť projektu ako celku a výstup a správanie interpretu by sa nedali považovať za správne. Prácu sme si rozdelili na tieto základné časti:

- lexikálna analýza
- syntaktická analýza
- sémantická analýza
- interpret
- vstavané funkcie interpretu
- správne zvolené datové štruktúry
- implementácia rozšírení
- testovanie

3 Implementácia

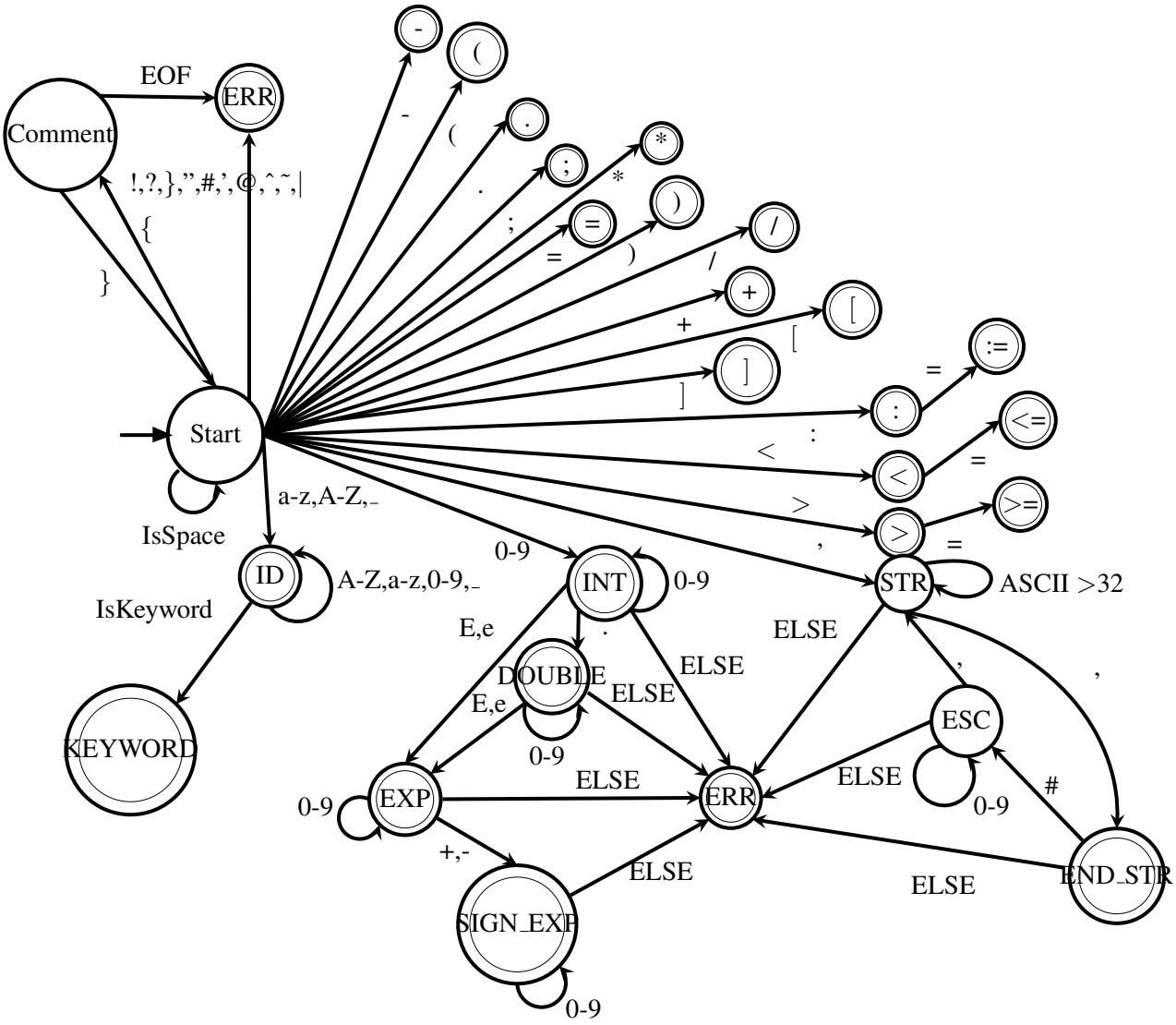
3.1 Vstavané rozšírenia

V našom projekte sme sa rozhodli implementovat nasledujúce rozšírenia:

- MSG - chybový výstup podla formátu GNU STANDARD
- Base - celočíselné a escape sekvence je možné zadávať aj dvojkovej, osmičkovej a šestnástkovej sústave

3.2 Lexikálna analýza

Lexikálny analyzátor rozdeľuje vstupný program pomocou konečného automatu na lexémy a odstraňuje biele znaky. Lexémy sú lexikálne jednotky reprezentujúce čísla, názvy identifikátorov, kľúčove slová atď. Lexémy sú následne uložené do tokenov, s rôznymi atribútmi. Token je dátovy typ ktorý v sebe nesie názov lexému, typ lexému a pozíciu na ktorom riadku a v ktorom stlpci sa nachádza. V prípade, že lexikálna analýza narazí na nepovolený znak v zdrojovom programe, musí zahlásiť chybu. Konečný automat použitý v lexikálnej analýze je podrobnejšie popísany v nasledujúcom diagrame 3.2. Z priestorových dvôvodov sú v diagrame vynechané niektoré malé detaily.


Comment
EOF
ERR
!,?,},”,#,’,@,^,~,|
{
}
Start
IsSpace
a-z,A-Z,_
ID
A-Z,a-z,0-9,_
IsKeyword
KEYWORD
0-9
INT
0-9
E,e
.
DOUBLE
E,e
0-9
ELSE
ELSE
ELSE
EXP
0-9
ELSE
+,-
SIGN_EXP
0-9
ELSE
ERR
-
(
.
;
*
=
)
/
+
[
]
:
:=
<
<=
>
>=
=
,
STR
ASCII >32
ELSE
,
,
ESC
ELSE
0-9
#
ELSE
END_STR

3.3 Syntaxou riadený preklad

Syntaktická analýza prekladá zdrojový kód na pseudoinštrukcie a generuje inštrukcie pre ich interpretáciu. Pokiaľ nastane chyba (syntaktická, sémantická alebo vnútri prekladača), interpretácia sa nespustí a vypíše sa kód chyby. Na vstupe se očakáva zdrojový kód, ktorý prejde lexikálnou analýzou. Potom syntaktická analýza pracuje s reťazcom tokenov, získaných z lexikálnej analýzy funkciou $getNextToken()$.

Na overenie syntaxe kódu sa využívá rekurzivný zostup podľa pravidiel precedenčnej a prediktívnej LL-gramatiky. Výrazy sa spracovavajú pomocou precedenčnej gramatiky metodou zdola nahor. Všetko ostatné sa srpacováva prediktivnou gramatikou zhora dolu. Inštrukcie sa generujú pomocou funkcie $add\_instruction()$. V prílohe sú priložené použité pravidlá LL gramatiky a v prílohe 1 precedenčná tabulka.

Syntaktický analyzátor je pri syntakticky riadenom preklade kľúčovou jednotkou, pričom lexikálny, sémantický analyzátor a generátor trojadresného kódu sú mu podriadené. Od lexikálneho analyzátoru požiada token, ktorý spracováva a pridáva ho do syntaktického stromu a zároveň riadi sémantickú analýzu a generovanie kódu.

Tabulka 1: **Precedenčná tabuľka**

	+	**-**	*****	**/**	<	>	<=	>=	=	<>	(	)	**id**	**mn**	**$**
+	>	>	<	<	>	>	>	>	>	>	<	>	<	<	>
-	>	>	<	<	>	>	>	>	>	>	<	>	<	<	>
*****	>	>	>	>	>	>	>	>	>	>	<	>	<	<	>
/	>	>	>	>	>	>	>	>	>	>	<	>	<	<	>
<	<	<	<	<	>	>	>	>	>	>	<	>	<	<	>
>	<	<	<	<	>	>	>	>	>	>	<	>	<	<	>
<=	<	<	<	<	>	>	>	>	>	>	<	>	<	<	>
>=	<	<	<	<	>	>	>	>	>	>	<	>	<	<	>
=	<	<	<	<	>	>	>	>	>	>	<	>	<	<	>
<>	<	<	<	<	>	>	>	>	>	>	<	>	<	<	>
(	<	<	<	<	<	<	<	<	<	<	<	=	<	<	
)	>	>	>	>	>	>	>	>	>	>		>			>
id	>	>	>	>	>	>	>	>	>	>		>			>
mn	>	>	>	>	>	>	>	>	>	>		>			>
$	<	<	<	<	<	<	<	<	<	<	<		<	<	

3.4 Tabuľka symbolov

Podľa zadania je tabuľka symbolov implementovaná ako binárny vyhľadávací strom, kde kľúčom, podľa ktorého sa vyhľadáva je jedinečný názov identifikátoru. Kazdý symbol v strome obsahuje dodatočné informácie ako

3.5 Tabuľka funkcií

Existuje jedna globálna tabuľka funkcií, kde každá funkcia ma vlastnú tabuľku symbolov. Okrem odkazu na tabuľku symbolov obsahuje jedinečný identifikátor, adresu funkcie,

3.6 Interpret

Interpret vykonáva interpretáciu trojadresného kódu, ktorý je generovaný syntatickou analýzou. Trojadresný kód je ukladaný do instrukčného listu, ktorý je implementovaný pomocou jednosmerne viazaného lineárneho zoznamu.

3.7 Knuth-Morris-Prattov algoritmus

Algoritmus využívá pomocné pole, kde sa najskôr vypočíta prekrytie (overlap), ktoré se využíva pri vyhľadávaní. Pri vyhľadavání sa porovnává písmenko po písmenku a pri nezhode nám pomocné pole povie ako moc sa vrátiť. Zložitosť algoritmu je maximálne $O(m + n)$.

3.8 Heapsort

Heapsort je jeden z najlepších algoritmov zoraďovania. Základnou myšlienkou je využitie dátovej štruktúry halda (heap). Pred začiatkom zoradenia je najskôr treba haldu pripravit. Potom vždy pri každom priechode zmeníme prvý a posledný prvok a následne "pripravíme" haldu na ďaľší priechod. Aj ked môže byť pomalší než quicksort, je jeho časová náročnost $O(NlogN)$.

3.9 Testovanie

Na testovanie našeho interpretu sme použili automatizované testovacie skripty, ktoré mali za úlohu porovnávať očakávané výstupy s reálnymi výstupmi našeho interpretu. Tieto testy sa snažili odhaliť hlavne logické chyby a nedostatky. Po riadnom testovaní a odladení chýb sme projekt znova prestestovali na školskom servere merlin.

4 Záver

V tomto projekte sme sa stretli so všetkými nástrahami a problémami, ktoré sa vyskytujú pri každom rozsiahlejšom softvérovom projekte. Hlavným cielom tohoto projektu bolo pochopit a vyskúšať si ako funguje proces prekladu programov a pochopiť formálne jazyky. Neoddeliteľnou súčasťou bola práca v týme, kde sme si vyskúšali ako pracovať v tíme a ako z jednotlivých menších častí poskladať rozsiahly program.