VYSOKÉ UČENÍ TECHNICKÉ V BRNE
Fakulta informačních technologii

Dokumentácia k projektu z predmetu IFJ

Implementácia prekladača imperativního jazyka IFJ17

Rozšírenia :

Tím 100, varianta 2

Tomáš Lapšanský, xlapsa00

David Dejmal, xdejma00

Marek Kalabza, xkalab09

Tomáš Fedor, xfedor08

Obsah

1.Úvod

Táto dokumentácia sa zaoberá implementáciou prekladaču imperatívneho jazyka IFJ17. Dokumentácia je rozdelená do jednotlivých kapitol a ich podkapitol. V dokumentácii sa dozviete všetko o komunikácii tímu, implementácii jednotlivých častí projektu.

2. Práca v tíme

2.1 komunikácia

Náš tím mal pravidelné stretnutia jeden krát za týždeň, ale začiatky boli ťažké. Po pár stretnutiach už sme mali jasno, kto a čo bude robiť. Samozrejme naďalej sme sa stretávali jeden krát do týždňa, kde sme riešili ako budeme ďalej pokračovať na riešení projektu. Okrem stretnutí sme mali aj hlavne internetovú komunikáciu.

2.2 Správa kódu

Na to aby každí člen tímu vedel o aktuálnom stave projektu, tak na správu zdrojového kódu sme použili Github. Tento spôsob nám vyhovoval najviac, keďže každí člen tímu mohol upravovať a pridávať zdrojový kód.

2.3 Rozdelenie práce

Prácu sme si rozdelili na prvom stretnutí, takže sme nemali problém s tým kto a čo bude robiť.

3. Implementácia

3.1 Lexikálny analyzátor

Lexikálny analyzátor je implementovaný ako konečný automat, ktorý postupne načíta jednotlivé znaky zo zdrojového súboru. Podľa týchto znakov vytvára tokeny. Token sa skladá z poľa znakov, informácie o veľkosti tohto poľa znakov a identifikujúceho čísla, ktoré označuje typ tokenu. Pokiaľ sa analyzátor nedostane do požadovaného stavu, v konečnom automate vracia lexikálnu chybu.

3.2 Syntaktický analyzátor

Na implementáciu syntaktického analyzátora sme použili metódu rekurzívneho zostupu. Pri tejto metóde sme využívali aj LL gramatiku ktorej schéma je v prílohe ku dokumentácií. Pri vyhodnocovaní výrazov sme podľa zadania použili precedenčnú syntaktickú analýzu ku ktorej mame taktiež implementovanú precedenčnú tabuľku symbolov podľa zadania. Na komunikáciu medzi syntaktickou a sémantickou analýzou a tabuľku symbolov sme využili nami implementované funkcie pre vkladanie a vyhľadávanie v tabuľke.

3.3 Precedenčná syntaktický analýza

Táto časť analýzy je volaná, keď syntaktická analýza narazí na vyhodnotenie výrazu. Pre overenie správnosti výrazu je využitý dvojsmerný viazaný zoznam. Samotné vyhodnotenie je realizované pomocou statickej tabuľky prechodov vďaka ktorej určíte prioritu a asociativitu operátorov.

3.4 Tabuľka symbolov

Tabuľka symbolov a jej vnútorné usporiadanie sú implementované pomocou abstraktných dátových štruktúr. Tabuľka s rozptýlenými položkami tvorí základ tabuľky symbolov. V každej položke tejto štruktúry sa nachádza ukazateľ, ktorý slúži pre vytvorenie ďalšej vnorenej tabuľky symbolov. Táto tabuľka sa bude vytvárať za chodu programu a iba pokiaľ je to nutné. Na začiatku programu sa teda vždy vytvorí hlavná tabuľka a ďalej sa vytvorí ďalšia až podľa potreby programu. Na konci programu, ak skoční úspechom alebo neúspechom, sa všetky vytvorené tabuľky uvoľnia.

Kľúč v tabuľke zastupuje ID premenné či funkcie. Okrem ďalších premenných obsahuje položka taktiež hlavičku lineárneho jednosmerne viazaného zoznamu. Ten slúži k uloženiu parametrov funkcií v pevne danom poradí.

3.5 Mapovanie funkcie tabuľky symbolov

Pri implementácií tabuľky symbolov sme vychádzali z projektu z predmetu IAL (zadanie c016) a preto je mapovacia funkcia identická. Veľkosť tabuľky sme nastavali na 19 prvkov. Tu sa nachádza veľký priestor pre optimalizáciu. Pokiaľ by sme počet prvkov zmenšili ušetrili by sme pamäť pri tvorbe tabuľky. Na druhú stranu hrozí dlhší prístup k položkám pri väčšom počte premenných a funkcií. Pokiaľ veľkosť tabuľky zväčšíme moc je možné že viac prvkov tabuľky zostane prázdnych a budú zbytočne zaťažovať pamäť. Preto je nutné aby veľkosť tabuľky bola čo najviac možno rovná počtu vložených prvkov.

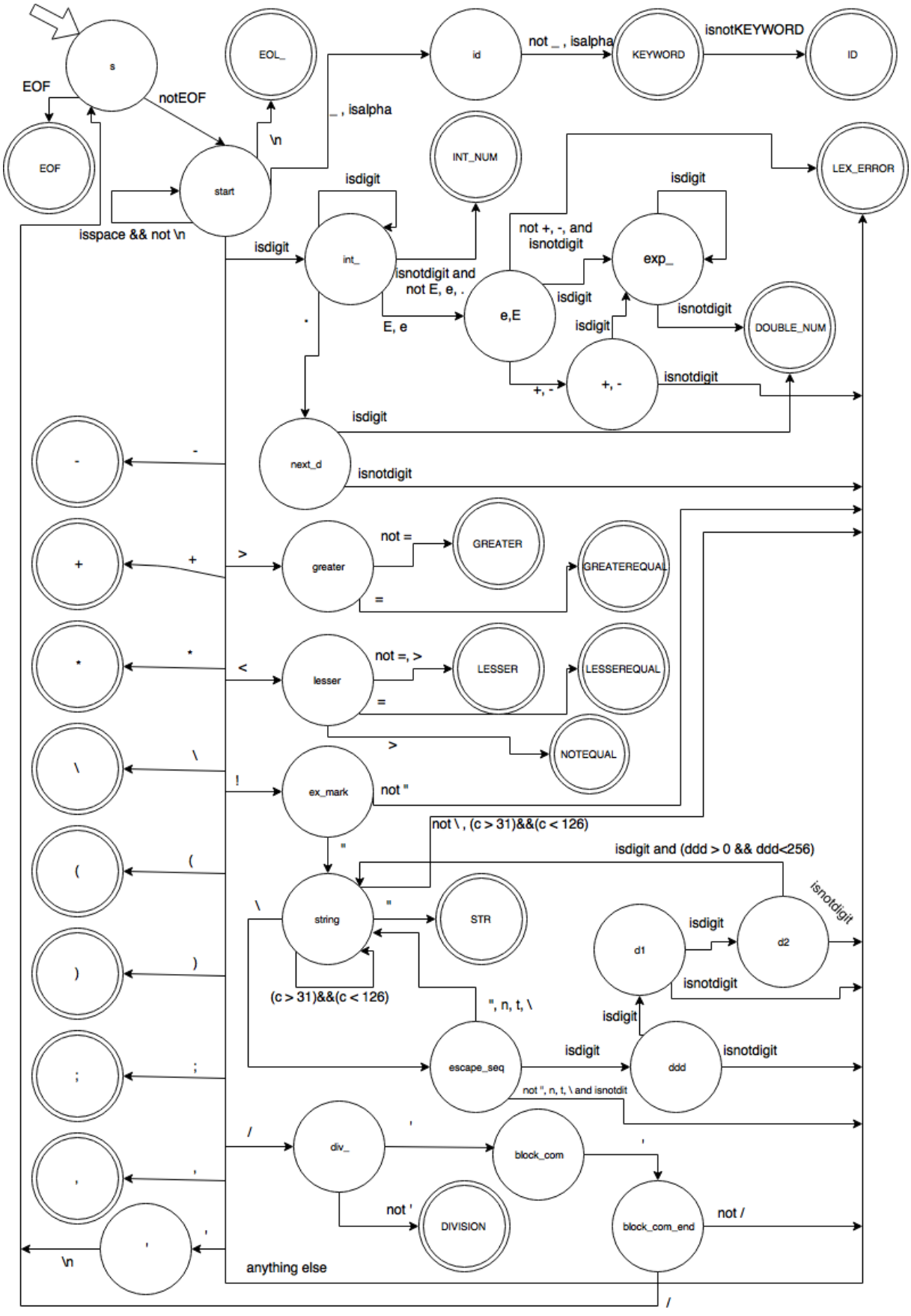
3.6 Mapovacia funkcia

```
int hashCode ( char* key ) {
        int retval = 1;
        int keylen = strlen(key);
        for ( int i=0; i<keylen; i++ )
                retval += key[i];
        return ( retval % HTSIZE );
}
```

4. Záver

Projekt sme sa snažili vypracovať podľa zadania, čo si myslím, že sa nám podarilo. Vďaka tomuto projektu sme nadobudli veľa nových skúseností. Projekt bol síce náročný, ale stihli sme vypracovať funkčnú časť do pokusného odovzdania.

5. Konečný automat



6. Precedenčná tabuľka symbolov

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

7. LL gramatika

```
<p_start>               <p_define> <p_scope>
<p_scope>               Scope <p_body> End Scope EOF
<p_define>              ε
<p_define>              Declare Function ID (<p_declareparameter> As <p_type> EOL <p_define>
<p_define>              Function ID (<p_parameter> As <p_type> EOL <p_body> End Function <p_define>
<p_define>              Function ID (<p_declareparameter> As <p_type> EOL <p_body> End Function <p_def:

<p_body>                ε
<p_body>                <p_prikaz> EOL <p_body>

<p_type>                Integer
<p_type>                Double
<p_type>                String
<p_type>                Boolean_
<p_type>                Int_num
<p_type>                Double_num
<p_type>                Str
<p_type>                BL

<p_parameter>           ε)
<p_parameter>           ID As <p_type>)
<p_parameter>           ID As <p_type>, <p_nextparameter>

<p_nextparameter>       ID As <p_type>)
<p_nextparameter>       ID As <p_type>, <p_nextparameter>

<p_declare_parameter>   ε)
<p_declare_parameter>   ID As <p_type>)
<p_declare_parameter>   ID As <p_type>, <p_declare_nextparameter>

<p_declare_nextparameter>       ID As <p_type>)
<p_declare_nextparameter>       ID As <p_type>, <p_declare_nextparameter>

<p_vparameter>          ε)
<p_vparameter>          ID)
<p_vparameter>          ID, <p_vnextparameter>

<p_vnextparameter>      ID)
<p_vnextparameter>      ID, <p_vnextparameter>

<p_prikaz>              ε
<p_prikaz>              Dim ID As <p_type>
<p_prikaz>              ID = <p_priradenie>
<p_prikaz>              Input ID
<p_prikaz>              Print <p_print>
<p_prikaz>              If <p_vyraz> Then EOL <p_body> Else EOL <p_body> End If
<p_prikaz>              Do While <p_vyraz> EOL <p_body> Loop
<p_prikaz>              Return <p_vyraz>

<p_priradenie>          F_ID(<p_vparameter>
<p_priradenie>          <p_vyraz>

<p_print>               <p_vyraz>; <p_nextprint>
<p_print>               String; <p_nextprint>

<p_nextprint>           ε
<p_nextprint>           <p_vyraz>; <p_nextprint>
<p_nextprint>           String; <p_nextprint>
```

8. LL tabuľka

	Scope	End	EOF	ε	Integer	Double	String	Boolean	Int_num	Double_num	Str	BL	ID	As	Dim	Input	If	Then	Else	EOL	End	Do	While	Loop	Return	F_ID	:	String	Function	Declare	Print	=
p_start																																
p_scope	2	2	2																													
p_define				3									4	4						4	5							4	4			
p_body				7																8												
p_type					9	10	11	12	13	14	15	16																				
p_parameter				17									18	18																		
p_nextparameter													20	20																		
p_declare_parameter				22									23	23																		
p_declare_nextparameter													25	25																		
p_vparameter				27									28																			
p_vnextparameter													30																			
p_prikaz				32									33	33	33	35	37	37	37	37	37	38	38	38	39						36	34
p_printdemic																										40						
p_print																											42	43				
p_nextprint				44																							45	46				