Fakulta Informačních Technologií
Vysoké Učení Technické v Brně

Dokumentace IFJ 2017

Tým 112, varianta II

Rozšíření:
SCOPE
IFTHEN
FUNEXP

Autoři, rozdělení bodů:
Martin Hošala (vedoucí), xhosal00, 25 %
Vladimír Dušek, xdusek27, 25 %
Peter Kubov, xkubov06, 25 %
Tomáš Kukaň, xkukan00, 25%

1. Úvod

Tento dokument obsahuje dokumentaci k řešení projektu do předmětu IFJ, překladač z jazyka IFJ17, jenž je podmnožinou jazyka FreeBASIC, do IFJcode17.

2. Použité datové struktury

- Tabulka symbolů
- Lineárny zoznam pre dočasné uchovanie bloku trojadresného kódu.
- Symbol table is implemented as a hash table.
- Prediktívna syntaktická analýza využíva zásobnik pre terminály, neterminály a sámanticke akcie.
- Sémantické akcie využívaju zásobník pre ukladanie spracovávaných premenných a funkcii.
- Pre rozšírenie SCOPE bol implementovaný zásobník tabuliek symbolov pre možnosť vrstviť kód do blokov.

3. Tabulka symbolů

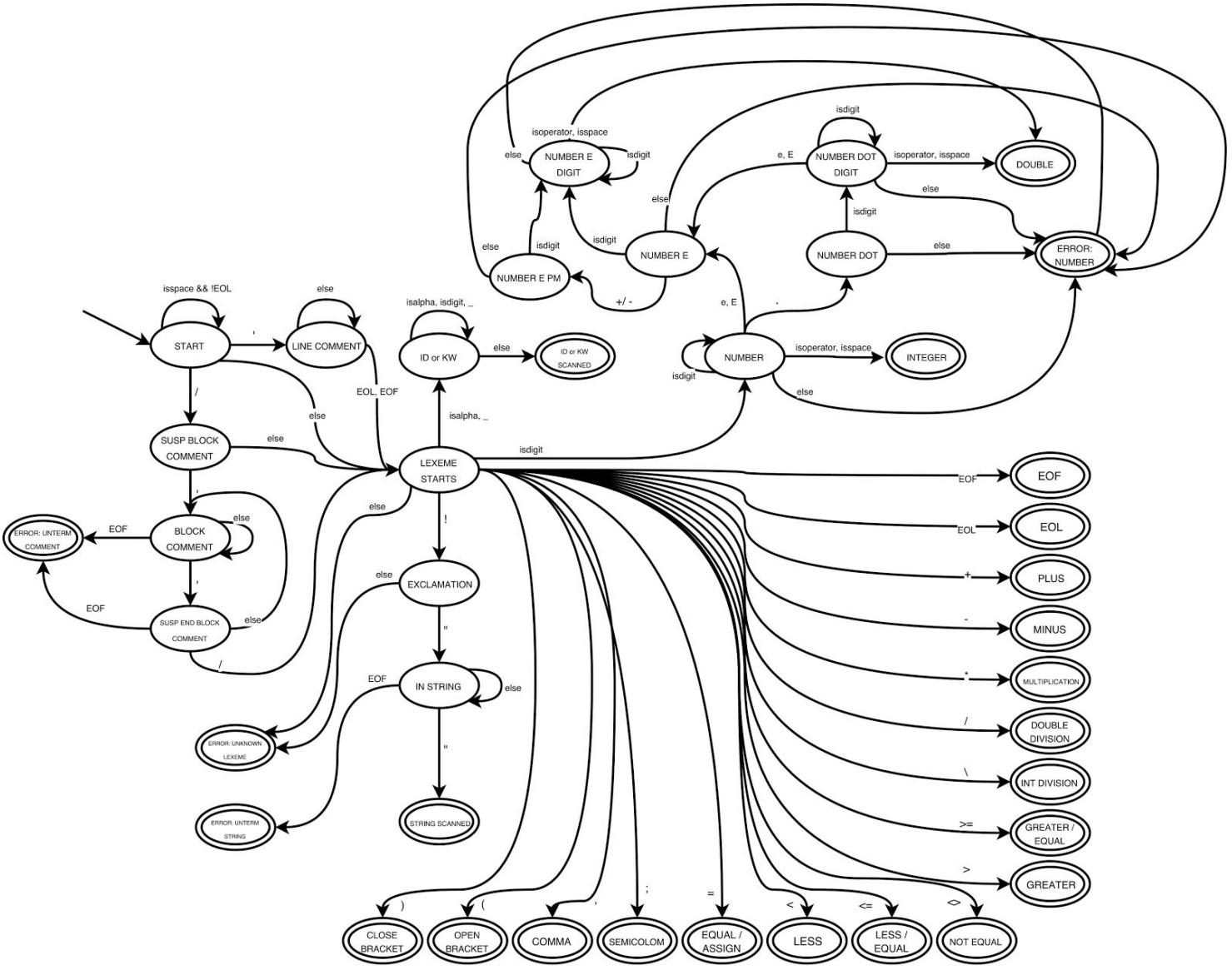
Tabulku symbolů jsme dle zadání implementovali jako tabulku s rozpýlenými hodnotami. Jako hashovací funkci jsme vybrali DJB2[1] známá svojí rychlostí, ale i přesto kvalitní distribucí. V tabulce přidáváme na konec seznamu, aby se při přidávání mohlo zkontrolovat, zda-li se nepřidává už existující položka a tím to udělat tuto tabulku více robusnou. Tabulku používáme pro ukládání všech identifikátorů včetně funkcí. Funkce má navíc jako jednu položku seznam argumentů. Tabulky se dle zanoření v programu ukládají na zásobník tabulek.

4. Lexikální analýza

Lexikální analyzátor je implementován jako konečný automat (viz. 2.0.1), který čte zdrojový kód jazyka IFJ17. Postupně rozdělí vstupní posloupnost znaků na lexémy, ty uloží jako tokeny a na vyžádání přidělý syntaktickému analyzátoru, který celý překlad řídí. Token je reprezentován jako struktura, která obsahuje výčtový typ se stavem tokenu a union, ve kterém jsou uloženy další informace o tokenu (např. hodnota integeru, konkretní klíčové slovo, ...). Syntaktický analyzátor volá funkce z lexikální analýzy a sám si pomocí těchto funkcí alokuje paměť pro tokeny, tokeny plní a uvolňuje po nich paměť. Může také navrátit načtený tokenu k opětovnému načtení.

[1] http://www.cse.yorku.ca/~oz/hash.html

4.0.1. Diagram konečného automatu



5. Prediktívna syntaktická analýza

Pre syntaktický analyzátor sme zvolili prediktívnu syntaktickú analýzu riadenú LL-tabuľkou (3.0.2.). Pri tvorbe syntaktickej analýzy sa postupovalo výhradne na základe výkladu učiva na predmete IFJ a teda celý proces analýzy sa snaží napodobniť na prednáškach ukázaný návrh syntaktického analyzátoru zhora dole, teda číta vstupy od lexikálnej analýzy a predpovedá ďalší stav spracovávaného programu na základe zásobníka a LL-tabuľky. Pre správnu analýzu syntaxe bola navrhnutá LL-gramatika, ktorá má definované pravidlá (3.0.1.). Pri návrhu gramatiky bola vzatá v ohľad skutočnosť, že výrazy sú spracovávané osobitnou syntaktickou analýzou a preto sa skutočná implementácia líši v miestach očakávania neterminálu v gramatike označeného ako <NT_EXPPRESSION>, kde je precedentná syntaktická analýza využívaná.

5.0.1. LL-gramatika G_{IFJ17}

Nech G_{IFJ17} = (N, T, P, S), kde:

- N je abeceda neterminálov
- T je abeceda terminálov
- P je konečná množina pravidiel
- S je počiatočný neterminál

Pre G_{IFJ17} platí:

N = {<NT_PROGAM>, <NT_FUNCTIONS>, <NT_DEFINE>, <NT_PARAMETERS>, <NT_NEXTPARAMETER>, <NT_DEF_PARAMETERS>, <NT_DEF_NEXTPARAMETER>, <NT_STATEMENTS>, <NT_STATEMENT>, <NT_WHILE_STATEMENTS>, <NT_IF_STATEMENTS>, <NT_ELSE_STATEMENTS>, <NT_DATATYPE>, <NT_INITIALIZATION>, <NT_NEXT_EXPR>, <NT_IDENTIFIER>, <NT_EXPRESSION>}

T = {T_AS, T_ASSIGN, T_CLOSEBRACK, T_COMMA, T_DECLARE, T_DIM, T_DO, T_DOUBLE, T_ELSE, T_ELSIF, T_END, T_EOF, T_EOL, T_FOR, T_FUNCTION, T_FUNID, T_IDENTIFIER, T_IDENTIFIER_HT, T_IF, T_INPUT, T_INTEGER, T_INTVAL, T_STRINGVAL, T_DOUBLEVAL, T_LOOP, T_OPENBRACK, T_OR, T_PRINT, T_RETURN, T_SCOPE, T_SEMICOL, T_STRING, T_THEN, T_WHILE}

S = <NT_PROGAM>

P je množina obsahujúca pravidlá:

Globálna časť programu

1. <NT_PROGAM> -> <NT_FUNCTIONS><NT_PROGRAM>
2. <NT_PROGRAM> -> (T_SCOPE)(T_EOL)<NT_STATEMENTS>(T_SCOPE)<NT_PROGRAM>
3. <NT_PROGRAM> -> (T_EOL)<NT_PROGRAM>
4. <NT_PROGRAM> -> (T_EOF)

Funkčná časť programu

5. <NT_FUNCTIONS> ->
(T_DECLARE)(T_FUNCTION)(NT_IDENTIFIER)(T_OPENBRACK)<NT_PARAMETERS>(T_AS)<NT_DATATYPE>(T_EOL)
6. <NT_FUNCTIONS> -> (T_FUNCTION)<NT_DEFINE>
7. <NT_DEFINE> ->
(NT_IDENTIFIER)(T_OPENBRACK)<NT_DEF_PARAMETERS>(T_AS)<NT_DATATYPE>(T_EOL)<NT_STATEMENTS>(T_FUNCTION)(T_EOL)
8. <NT_DEFINE> ->
(T_FUNID)(T_OPENBRACK)<NT_DEF_PARAMETER>(T_AS)<NT_DATATYPE>(T_EOL)<NT_STATEMENTS>(T_FUNCTION)(T_EOL)

Parametre funkcie

9. <NT_PARAMETERS> -> (T_CLOSEBRACK)
10. <NT_PARAMETERS> -> <NT_PARAMETER><NT_PARAMETERS>
12. <NT_PARAMETER> -> <NT_IDENTIFIER><NT_NEXTPARAMETER>
13. <NT_NEXTPARAMETER> -> (T_CLOSEBRACK)
14. <NT_NEXTPARAMETER> -> (T_COMMA)<NT_PARAMETER><NT_NEXTPARAMETER>
15. <NT_DEF_PARAMETERS> -> (T_CLOSEBRACK)
16. <NT_DEF_PARAMETERS> -> <NT_DEF_PARAMETER><NT_DEF_PARAMETER>
17. <NT_DEF_PARAMETER> -> <NT_IDENTIFIER><NT_DEF_NEXTPARAMETER>
17. <NT_DEF_NEXTPARAMETER> -> (T_CLOSEBRACK)
19. <NT_DEF_NEXTPARAMETER> -> (T_COMMA)<NT_PARAMETER><NT_DEF_NEXTPARAMETER>

Štruktúra príkazov

20. <NT_STATEMENTS> -> (T_END)
21. <NT_STATEMENTS> -> <NT_STATEMENT><NT_STATEMENTS>

Príkaz podrobne

22. <NT_STATEMENT> -> (T_EOL)
23. <NT_STATEMENT> ->
(T_DIM)(T_IDENTIFIER)(T_AS)<NT_DATATYPE><NT_INITIALIZATION>
24. <NT_STATEMENT> -> (T_IDENTIFIER_HT)(T_ASSIGN)<NT_EXPRESSION>(T_EOL)
25. <NT_STATEMENT> -> (T_FUNID)(T_EOL)
26. <NT_STATEMENT> -> (T_SCOPE)(T_EOL)<NT_STATEMENTS>(T_SCOPE)(T_EOL)
27. <NT_STATEMENT> -> (T_INPUT)(T_IDENTIFIER_HT)(T_EOL)
28. <NT_STATEMENT> -> (T_PRINT)<NT_EXPRESSION>(T_SEMMICOL)<NT_NEXT_EXPR>
29. <NT_STATEMENT> ->
(T_DO)(T_WHILE)<NT_EXPRESSION>(T_EOL)<NT_WHILE_STATEMENTS>(T_EOL)
30. <NT_STATEMENT> -> (T_RETURN)<NT_EXPRESSION>(T_EOL)
31. <NT_STATEMENT> ->
(T_IF)<NT_EXPRESSION>(T_THEN)<NT_IF_STATEMENT>(T_IF)(T_EOL)

Príkaz v cykle while

32. <NT_WHILE_STATEMENTS> -> (T_LOOP)
33. <NT_WHILE_STATEMENTS> -> <NT_STATEMENT><NT_WHILE_STATEMENTS>

Podmienený príkaz

35. <NT_IF_STATEMENTS> -> (T_END)
36. <NT_IF_STATEMENTS> -> <NT_STATEMENT><NT_IF_STATEMENTS>

37. <NT_IF_STATEMENTS> ->
(T_ELSIF)<NT_EXPRESSION>(T_THEN)(T_EOL)<NT_IF_STATEMENTS>
38. <NT_IF_STATEMENTS> -> (T_ELSE)(T_EOL)<NT_ELSE_STATEMENTS>
39. <NT_ELSE_STATEMENTS> -> (T_END)
40. <NT_ELSE_STATEMENTS> -> <NT_STATEMENT><NT_ELSE_STATEMENTS>

Datové typy

41. <NT_DATATYPE> -> (T_STRINGVAL)
42. <NT_DATATYPE> -> (T_INTVAL)
43. <NT_DATATYPE> -> (T_DOUBLEVAL)

Inicializácia premennej

44. <NT_INITIALIZATION> -> (T_EOL)
45. <NT_INITIALIZATION> -> (T_ASSIGN)<NT_EXPRESSION>(T_EOL)

Príkaz print

46. <NT_NEXT_EXPR> -> (T_EOL)
47. <NT_NEXT_EXPR> -> <NT_EXPRESSION>(T_SEMICOL)<NT_NEXT_EXPR>

Identifikátor

48. <NT_IDENTIFIER> -> (T_IDENTIFIER)
49. <NT_IDENTIFIER> -> (T_IDENTIFIER_HT)

5.0.2. LL-Tabuľka

	T_DECLARE	T_FUNCTION	T_SCOPE	T_EOL	T_EOF	T_IDENTIFIER	T_FUNID	T_CLOSEBRACK	T_IDENTIFIER_HT	T_COMMA	T_PRINT	T_DO	T_RETURN
<NT_PROGRAM>	1	1	2	3	4	-	-	-	-	-	-	-	-
<NT_FUNCTIONS>	5	6	-	-	-	-	-	-	-	-	-	-	-
<NT_DEFINE>	-	-	-	-	-	7	8	-	-	-	-	-	-
<NT_PARAMETERS>	-	-	-	-	-	10	-	9	10	-	-	-	-
<NT_PARAMETER>	-	-	-	-	-	12	-	-	12	-	-	-	-
<NT_NEXTPARAMETER>	-	-	-	-	-	-	-	13	-	14	-	-	-
<NT_DEF_PARAMETERS>	-	-	-	-	-	16	-	15	16	-	-	-	-
<NT_DEF_PARAMETER>	-	-	-	-	-	18	-	-	18	-	-	-	-
<NT_DEF_NEXT_PARAMETER>	-	-	-	-	-	-	-	18	-	19	-	-	-
<NT_STATEMENTS>	-	-	21	21	-	21	21	-	21		21	21	21
<NT_WHILE_STATEMENTS>	-	-	33	33	-	33	33	-	33		33	33	33
<NT_IF_STATEMENTS>	-	-	36	36	-	36	36	-	36		36	36	36
<NT_ELSE_STATEMENTS>	-	-	40	40	-	40	40	-	40		40	40	40
<NT_STATEMENT>	-	-	26	22	-	-	25	-	24		28	29	30
<NT_DATATYPE>	-	-	-	-	-	-	-	-	-	-	-	-	-
<NT_INITIALIZATION>	-	-	-	44	-	-	-	-	-	-	-	-	-
<NT_NEXT_EXPR>	-	-	-	46	-	-	47	-	47	-	-	-	-
<NT_IDENTIFIER>	-	-	-	-	-	48	-	-	49	-	-	-	-

	T_IF	T_ELSEIF	T_ELSE	T_END	T_LOOP	T_STRING	T_INT	T_DOUBLE	T_ASSIGN	T_STRINGVAL	T_DOUBLEVAL	T_INTVAL
<NT_PROGRAM>	-	-	-	-	-	-	-	-	-	-	-	-
<NT_FUNCTIONS>	-	-	-	-	-	-	-	-	-	-	-	-
<NT_DEFINE>	-	-	-	-	-	-	-	-	-	-	-	-
<NT_PARAMETERS>	-	-	-	-	-	-	-	-	-	-	-	-
<NT_PARAMETER>	-	-	-	-	-	-	-	-	-	-	-	-
<NT_NEXTPARAMETER>	-	-	-	-	-	-	-	-	-	-	-	-
<NT_DEF_PARAMETERS>	-	-	-	-	-	-	-	-	-	-	-	-
<NT_DEF_PARAMETER>	-	-	-	-	-	-	-	-	-	-	-	-
<NT_DEF_NEXT_PARAMETER>	-	-	-	-	-	-	-	-	-	-	-	-
<NT_STATEMENTS>	21	-	-	20	-	-	-	-	-	-	-	-
<NT_WHILE_STATEMENTS>	33	-	-	-	32	-	-	-	-	-	-	-
<NT_IF_STATEMENTS>	36	37	38	35	-	-	-	-	-	-	-	-
<NT_ELSE_STATEMENTS>	40	-	-	39	-	-	-	-	-	-	-	-
<NT_STATEMENT>	31	-	-	-	-	-	-	-	-	-	-	-
<NT_DATATYPE>	-	-	-	-	-	41	42	43	-	-	-	-
<NT_INITIALIZATION>	-	-	-	-	-	-	-	-	45	-	-	-
<NT_NEXT_EXPR>	-	-	-	-	-	-	-	-	-	47	47	47
<NT_IDENTIFIER>	-	-	-	-	-	-	-	-	-	-	-	-

5.1. Sémantická analýza

Sémantická analýza na úrovni spracovania programu funguje na princípe sémantických akcii ukladaných na zásobník analyzátoru syntaxe pri predpovedaní následujúcej syntaxe programu. Sémantické akcie je možné pridávať na syntaktický zásobník a vykonávať ich vďaka schopnosti prediktívnej syntaktickej analýzy predpovedať vstupy, ktoré prídu od lexikálnej analýzy. Sémantické akcie majú rôznorodú štruktúru, napríklad kontrola datových typov, kontrola správnosti definície funkcie, generovanie trojadresného kódu, atď...

6. Spracovanie výrazov

6.1. Precedenčná syntaktická analýza

Na spracovanie výrazov bola v projekte podľa zadania použitá precedentná syntaktická analýza, ktorá pracuje so zásobníkovým automatom a je riadená precedentnou tabulkou (4.1.1.). Pri návrhu a implementácii precedentnej analýzy sa takisto postupovalo výhradne na základe znalostí získaných v predmetoch IFJ či IAL a teda postupuje zdola nahor. Keď prediktívna analýza narazí na miesto, kde čaká výraz, predá “riadenie” precedentnej analýze, ktorá následne číta tokeny zo vstupnej pásky a vykonáva príslušné akcie.Pokiaľ sa na konci vstupu, nájdu pravidlá pre opakovanú redukciu, až kým na zásobníku nenachádza práve jeden neterminál a nič iné, tak je výraz považovaný za spracovaný, čiže je syntakticky aj sématicky správne a taktiež je už vygenerovaný 3AK obsahujúci inštrukcie pre jeho “vyčíslenie”.

6.2. Sémantická analýza

Sémantická analýza výrazov prebieha zároveň s ich syntaktickou analýzou a to počas aplikovania syntaktických pravidiel. V prípade sémantickej správnosti postupnosti terminálov a neterminálov na ktorú je pravidlo aplikované, je generovaný 3AK obsahujúci inštrukcie prislúchajúce danému pravidlu a prípadnému nutnému implicitnému pretypovaniu.

6.3. Použité dátové štruktúry a rozšírenia

Kvôli odľahčeniu a väčšej prehľadnosti diania v zásobníkovom automate je zásobník implementovaný veľmi špecificky na danú problematiku a disponuje celkom prívetivým rozhraním, ktoré poskytuje veľmi pohodlné vykonávanie vyššie zmienených akcií a takisto jednoduchý prístup k terminálu ako aj neterminálu najbližšie vrcholu zásobníka. Celý zásobník je taktiež obojsmerne viazaný zoznam terminálov a neterminálov, pre uľahčenie “odtrhnutia” vrcholu zásobníku, až po najbližší terminál *hold,* v prípade redukcie.
Keďže je súčasťou našej implementátie taktiež rozšírenie FUNEXP, návrh syntaktických pravidiel, precedentnej tabuľky a takisto implementácia od začiatku toto rozšírenie zahrňovali. Pre spracovanie funkčných volaní sa používa veľmi jednoduchý pomocný zásobník obsahujúci informácie o práve spracovávanej funkcii (počet parametrov, počet skontrolovaných parametrov, atď.).
Precedentná tabulka je pre ľahkosť vyhľadávania implementovaná ako dvojrozmerné pole, kde vstupný terminál aj terminál najbližšie vrcholu zásobníku sú zároveň indexami akcie, ktorá danej kombinácii prislúcha

6.1.1. Precedentná tabulka

Akcie:

- **< (shift)**
- **> (reduce)**
- **= (push)**
- **X (none)**

	p1	p2	p3	REL	(	)	ID	FCE	,	$
p1	>	<	<	>	<	>	<	<	>	>
p2	>	>	<	>	<	>	<	<	>	>
p3	>	>	>	>	<	>	<	<	>	>
REL	<	<	<	>	<	>	<	<	>	>
(	<	<	<	<	<	=	<	<	=	X
)	>	>	>	>	X	>	X	X	>	>
ID	>	>	>	>	X	>	X	X	>	>
FCE	X	X	X	X	=	X	X	X	X	X
,	<	<	<	<	<	=	<	<	=	X
$	<	<	<	<	<	X	<	<	X	X

- **Terminál na vstupe**
- **Terminál najbližšie vrcholu zásobníku**

OZNAČENIE	TERMINÁL
p1	operátor s najnižšou prioritou (sčítanie, odčítanie)
p2	operátor so strednou prioritou (celočíselné delenie)
p3	operátor s najvyššou prioritou (násobenie, delenie)
REL	relačný operátor (väčší, menší, rovný...)
(	ľavá zátvorka
)	pravá zátvorka
ID	identifikátor premennej
FCE	identifikátor funkcie
,	čiarka
$	koniec vstupu / žiadny terminál

7. Generování kódu

Generátor je modul, který na základě zavolání od precedenční nebo sématické analýzy generuje výsledný kód IFJcode17. Generování se provádí už během překladu, aby se souběžně s vynořováním mohli uvolňovat tabulky symbolů.
Výše zmíněné moduly, které generátor používájí, vytváří lineární seznam instrukcí trojadresného kódu, který pak pošlou na vstup generátoru.
Tříadresný kód se skládá z požadované instrukce, která říká jaká logika se má vypsat, a třech ukazatelů na položky v tabulce a zároveň mohou ukazovat na NULL adresu, pokud jsou nevyužité. Ty předáváme proto, aby instrukce mohli být naprosto univerzální a mohlo se v nich předávat cokoliv.

8. Vývojový cyklus

1. Riešenie problému lexikálnej analýzy a návrhu konečných automatov.
2. Implementácia a testovanie lexikálnej analýzy.
3. Návrh tabuľky symbolov a operácii nad danou tabuľkou.
4. Návrh LL-gramatiky.
5. Návrh syntaktickej, sémantickej anlýzy a generátoru kódu.
6. Vytvorenie LL-tabuľky, precedentnej tabuľky.
7. Implementácia prediktívnej a precedentnej syntaktickej analýzy.
8. Implementácia generátoru cieľového kódu, definovanie sémantických akcií.
9. Implementácia sémantických akcií a integrácia modulov.
10. Vytvorenie regresných testov.
11. Testovanie celkového systému.

- *Návrh a implementácia rozšírení prebiehali zároveň s návrhom a implementáciou základných požiadaviek.*

9. Tímová spolupráca

9.1 Spôsob práce v tíme

V rámci zvýšenia produktivity boli zavedené pravidelné schôdze tímu kde boli prediskutovávané aktuálne problémy a ich riešenia a zvyšovalo sa tak povedomie o projekte ako celku. S blížiacim sa termínom odovzdania projektu sa interval medzi schôdzami zúžoval a zaviedli sa aj nové typy schôdzok, na ktorých členovia týmu spoločne programovali. Úlohy si každý vyberal dobrovoľne, na základe časti, ktorá mu najviac vyhovovala, však nedá sa povedať, že jeden člen tímu robil na jednej časti samostatne nakoľko bolo neustále nutné riešiť kompatibilitu modulov a s tým súvisiace problémy a počas skupinového programovania si samozrejme všetci členovia pomáhali ako vedeli.

9.2 Rozdelenie práce v týme

Martin Hošala - precedentná syntaktická a sémantická analýza výrazov
Peter Kubov - prediktívna syntaktická analýza a sémantická analýza
Tomáš Kukaň - generovanie kódu IFJ17, tabulka symbolů
Vladimír Dušek - lexikální analýza

10. Špeciálne použité techniky

V rámci vývoja projektu pre IFJ 2017 boli experimentálne využité rôzne techniky programovania. Využíval sa systém code review pre udržanie čistoty v hlavnom repozitári a pri náročnejších úloach bola snaha o pair coding, pričom členom tímu sa v priebehu semestra stala aj debugging kačička Steve.