Prezentace k projektu

Implementace překladače imperativního jazyka IFJ24

Tým: **xkalinj00**

Jan Kalina

David Krejčí

Lukáš Farkašovský

Pavel Hýža

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zdroj: Discord Icon [Online]. Dostupné z: https://logodownload.org/wp-content/uploads/2017/11/discord-logo-1-1.png
Zdroj: Visual Studio Code [Online]. Dostupné z: https://miro.medium.com/v2/resize:fit:1200/0*9xZylvsis4sqQOzb.png
Zdroj: GitHub [Online]. Dostupné z: https://th.bing.com/th/id/OIP.D_Gm8IGCvkqmOgtU2hueVwHaHS?rs=1&pid=ImgDetMain

Lexikální analýza

- „Scanner“
- Třídění vstupních znaků do 7 námi definovaných skupin:
  - Písmena
  - Číslice
  - Bílé znaky
  - Jednoduché operátory
  - Složité operátory
  - ‚EOF‘
  - Mimo jazyk IFJ24
- Efektivnější stavová logika
- Možnost využití znaků mimo základní tabulku ASCII: komentáře, `[]u8`

DEC	Colour
0-31	(U) Unprintable
35-39, 94, 96, 126, 127+	(X) Not in language
65-90, 95, 97-122	(L) Letters
48-57	(N) Numbers
40-45, 58, 59, 123-125	(S) Special Simple
33, 34, 46, 47, 60-64, 91-93	(C) Special Complex
10, 11, 12, 13, 32	(W) Whitespace
-1	(E) EOF

Tabulka rozdělení znaků pro lexikální analýzu

DEC	CHAR
-1	EOF

DEC	CHAR	DEC	CHAR	DEC	CHAR	DEC	CHAR
0	NUL	32	SP	64	@	96	`
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	'	71	G	103	g
8	BS	40	(	72	H	104	h
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	ETB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	\	124	|
29	GS	61	=	93	]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	_	127	DEL

Dynamický řetězec

- Uchování obsahu tokenů identifikátorů, znakových řetězců, číselných tokenů
- Automatické zvětšování

Tabulka symbolů trp-IZP

- Hashovací funkce – algoritmus djb2
- Indexové konflikty – řešeny lineárním průchodem
- Dynamická implementace
- Libovolně rozsáhlé programy, vysoká paměťová efektivita
- Každý rámec v parsovaném kódu – jedna tabulka symbolů

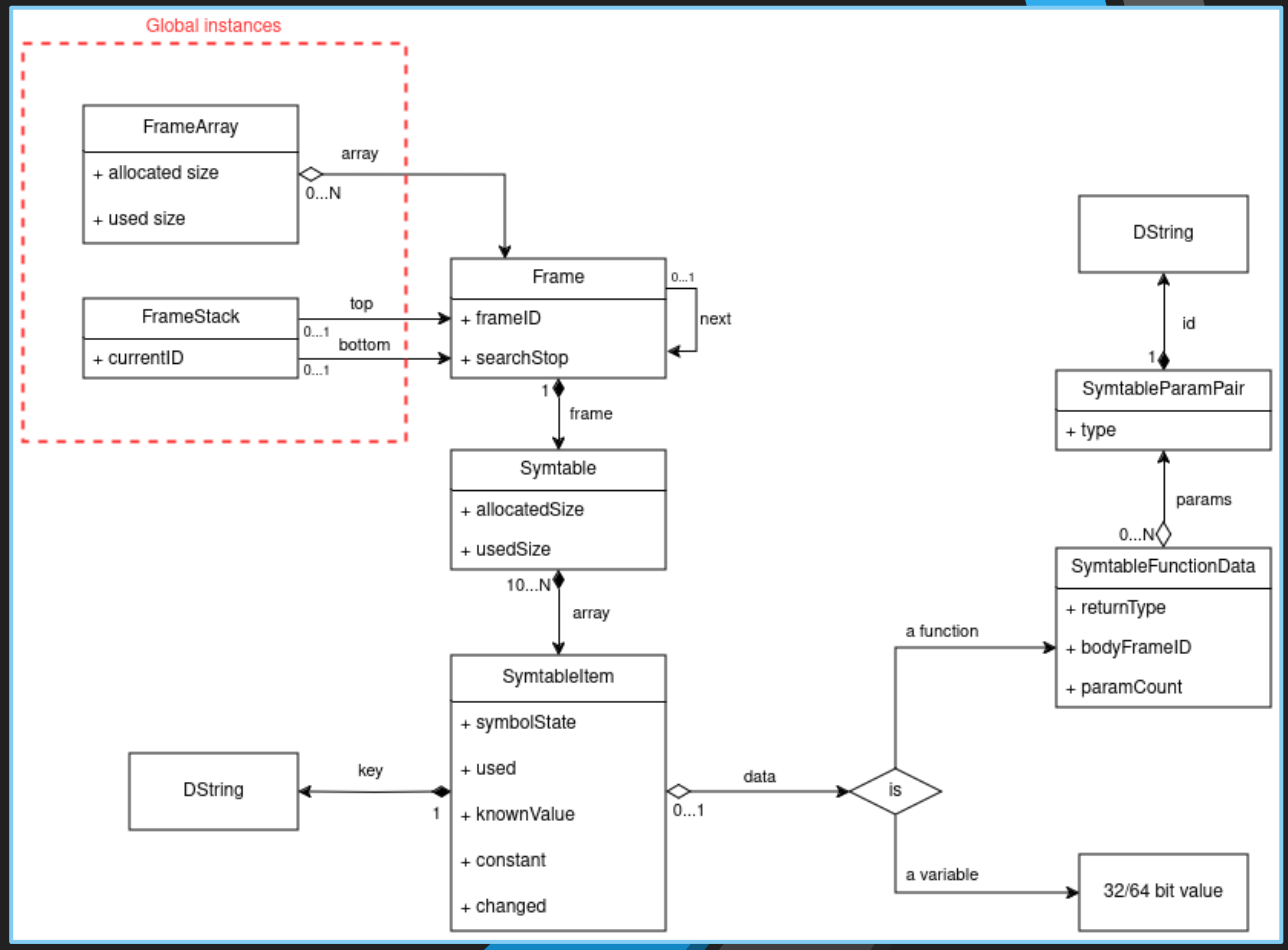


Diagram struktury tabulky symbolů

Zásobník rámců

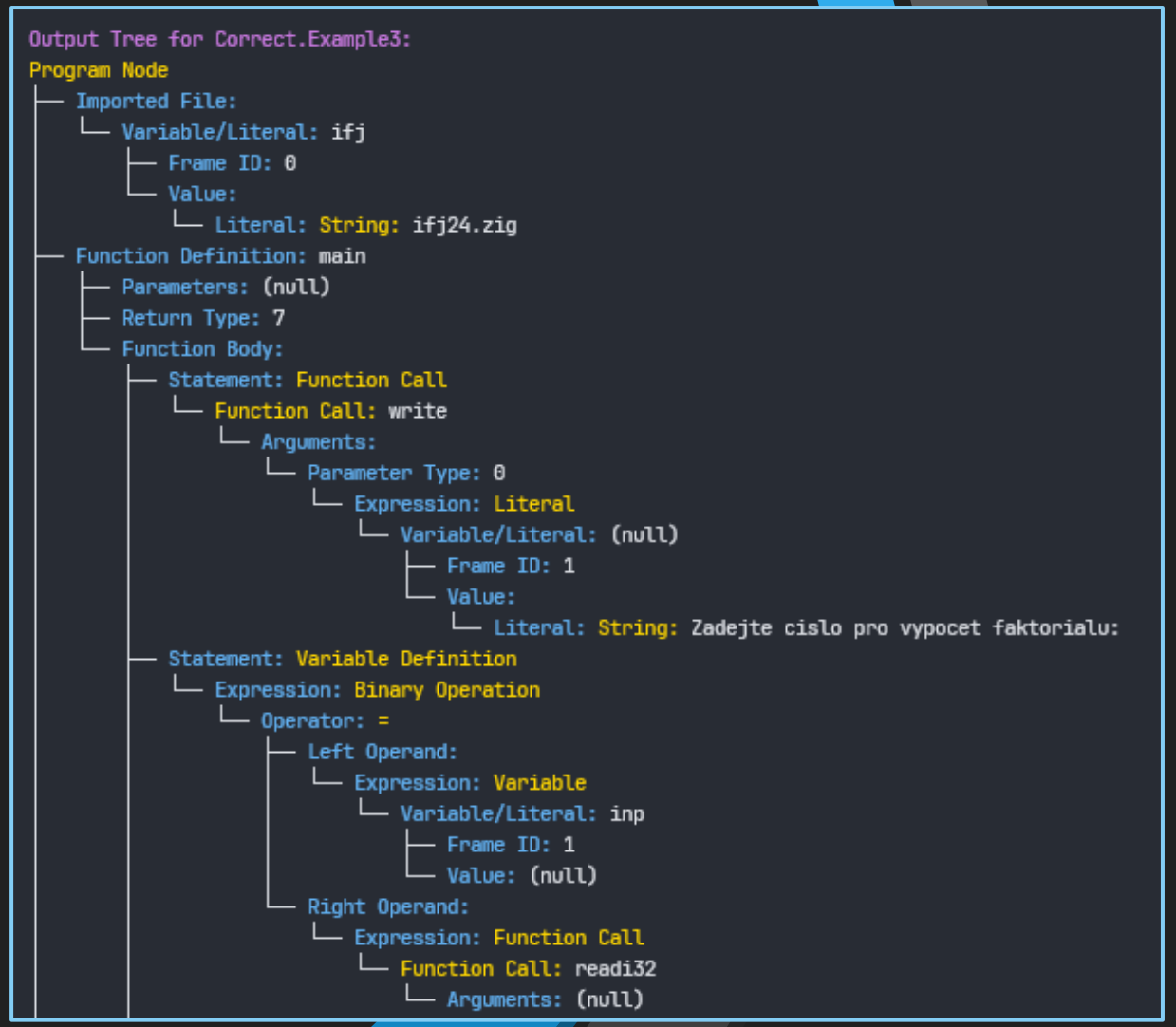
- Uchování tabulek symbolů
- Implementace – lineárně vázaný seznam
- Při vynoření je odebrán rámec ze zásobníku, ale zůstává v poli rámců pro využití během sémantické analýzy

Syntaktický analyzátor

- „Parser“
- Abstraktní syntaktický strom (AST)
- 5 submodulů
  - LL tabulka
  - LL parser
  - Precedenční tabulka
  - Precedenční parser
  - Parser common

AST

- Namísto jednoho univerzálního uzlu máme minimální počet různých uzlů kvůli přehlednosti
- Konstrukce – zdola nahoru, během parsingu
- Zajištění vysoké volnosti při konstrukci AST
  - Struktury některých uzlů obsahují ukazatel na `void`
  - Možnost propojovat uzly
- Program pro vizualizaci AST



Příklad výstupu programu pro vizualizaci AST

LL gramatika a LL tabulka

- Neterminály pro hlavní konstrukce jazyka
- Dělení na menší, například kvůli levé rekurzi
- C++ program pro automatické generování některých množin
- Nepoužití řešení pomocí dvourozměrného pole
- Implementace vlastní struktury
  - Klíč (terminál)
  - Pole LL pravidel – binární vyhledávání

```
1:  <PROGRAM> → <PROLOG> <FUN_DEF_LIST> EOF
2:  <PROLOG> → const ifj = @import ( [prec_expr] ) ;
3:  <FUN_DEF_LIST> → <FUN_DEF> <FUN_DEF_LIST>
4:  <FUN_DEF_LIST> → ε
5:  <FUN_DEF> → pub fn id ( <PARAMETERS> ) <RETURN_TYPE> <SEQUENCE>
6:  <PARAMETERS> → <PARAM_LIST>
7:  <PARAMETERS> → ε
8:  <PARAM_LIST> → <PARAM> <PARAM_LIST_REST>
9:  <PARAM_LIST_REST> → , <PARAM_LIST>
10: <PARAM_LIST_REST> → ε
11: <PARAM> → id : <DATA_TYPE>
12: <RETURN_TYPE> → <DATA_TYPE>
```

Ukázka LL gramatiky

LL parser

- První hlavní submodul Parseru
- Syntaktická analýza rekurzivním sestupem
- Řízení Scanneru
- Pro zpracování výrazů propůjčuje řízení Precedenčnímu parseru
- Během syntaktické kontroly plní tabulku symbolů
- Kontrola zastínění
- Kontrola redefinice identifikátoru v jeho rozsahu platnosti

Precedenční parser

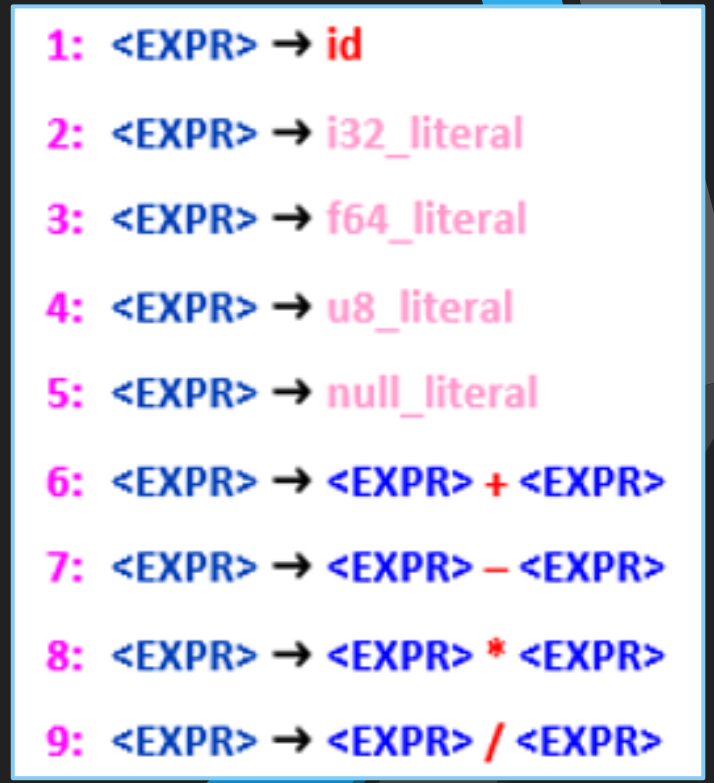
- Druhý hlavní submodul parseru
- LL parser předává řízení pro zpracování výrazů nebo argumentů v rámci rozšíření **FUNEXP**
- Precedenční tabulka priorit operátorů a tabulka redukčních pravidel tvořených sekvencí symbolů

Precedenční zásobník

- „Zásobník precedenčních zásobníků“ při parsování argumentů volání funkcí
- Shoda mezi sekvencí symbolů na vrcholu zásobníku a sekvencí redukčního pravidla – aplikace redukčního pravidla, jinak chyba

Precedenční tabulka

- Obsahuje také priority symbolů tvořících volání funkcí, ty mohou vystupovat ve výrazech a argumentech volání funkcí v rámci rozšíření **FUNEXP**



Ukázka redukčních pravidel

Parser common

- Třetí hlavní submodul Parseru
- Minimalizace duplicitního kódu v submodulech Parseru
- Jednotné zázemí pro LL i precedenční parser
- Speciální funkce `parser_errorWatcher()`
  - Statické proměnné pro zaznamenání chybových stavů
  - Propagace chyb zpět rekurzivním sestupem

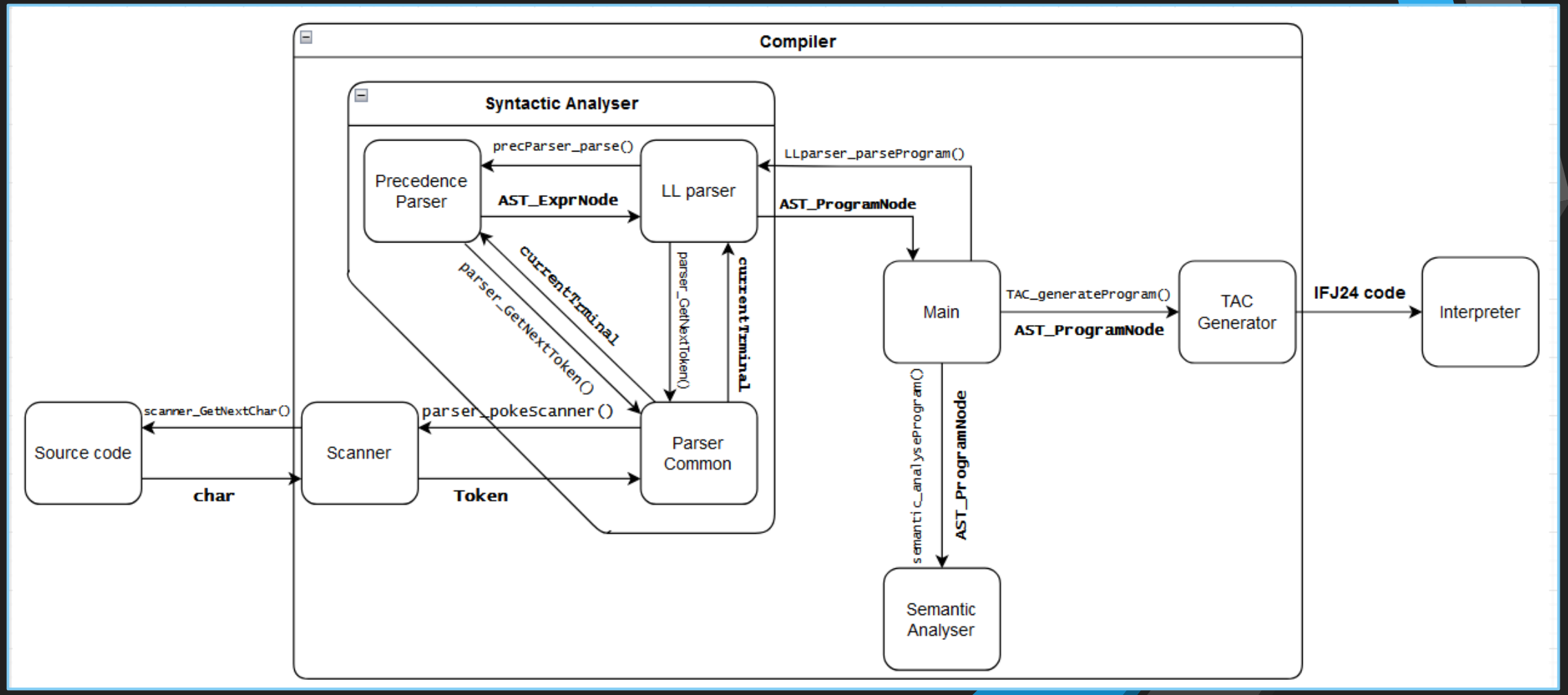
Sémantický analyzátor

- 4 fáze
  - Během parsingu: přidávání položek do rámců, kontrola nedefinovaných a vícekrát definovaných proměnných
  - Po parsingu: kontrola přítomnosti funkce `main()`, správnost prologu
  - Kontrola těl funkcí, jejich příkazů a výrazů, rekurzivní průchod stromem
  - Prohledání všech rámců programu, kontrola využití proměnných a neměnnosti konstant
- Naše rozšíření
  - Příkaz `return` se vyskytuje v každé větvi těla dané funkce
  - Detekce mrtvého kódu za příkazem `return`
  - Implicitní konverze při přiřazení
  - Vyhodnocování výrazů se známou hodnotou při překladu

Generátor kódu

- Prochází AST podobně jako sémantický analyzátor
- Zajištění unikátních identifikátorů – proměnným v mezikódu je přidáván sufix s číslem jejich rámce
- Při zanoření do cyklu se nejprve definují všechny jeho proměnné, až poté se generuje podmínka s tělem cyklu, a to již bez definic
- Vyhodnocování výrazů – výsledná hodnota předána na vrchol zásobníku, všechny operace dále používají zásobníkovou verzi, pokud je to možné
- Optimalizace
  - Vkládání sekvencí instrukcí vestavěných funkcí přímo do těla volající funkce, snížení počtu volání funkcí a výpočetního času
  - Instrukce `PUSHS` a `POPS` se předávají do optimalizačního bufferu, který se je pokusí nahradit instrukcí `MOVE`, redukce instrukcí o cca 15 %

Diagram překladače



Závěr

Testování

- Vlastní testy pro každou oddělenou část překladače
- Integrační testy pro chod překladače jako celku

Vlastní rozšíření

- Vlastní sémantické kontroly
- Optimalizace mezikódu
- Dynamická tabulka symbolů
- Možnost využívat znaky mimo základní ASCII v komentářích nebo řetězcích
- Kontrola meze přetečení datových typů integer a double během konstrukce AST
- Přes 150 unit testů GoogleTest framework
- C++ program pro generování množin `EMPTY`, `FIRST`, `FOLLOW`, `PREDICT`
- Možnost generování detailní Doxygen dokumentace
- Propagace chybových stavů parserem v rámci lex. a syntax. analýzy při návratu z rekurze
- „Godlike Makefile“