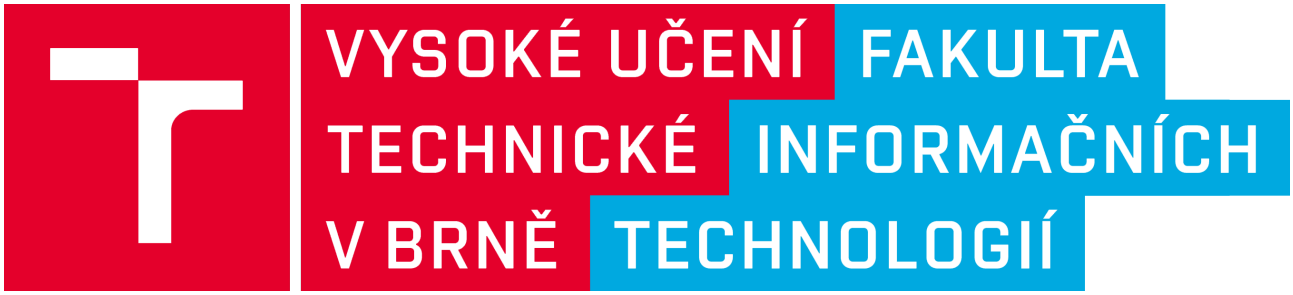


Dokumentace

Implementace překladače imperativního jazyka IFJ22

Tým xkucha28 Varianta TRP

Josef Kuchař (xkucha28) 25%
Matej Sirovatka (xsirov00) 25%
Tomáš Běhal (xbehal02) 25%
Šimon Benčík (xbenci01) 25%

CYCLES

Brno, 7. prosince 2022

Obsah

1 Tým

Josef Kuchař

- Lexikální analýza
- Rekurzivní syntaktická analýza
- Tabulka symbolů
- Generátor kódu

Matej Sirovatka

- Lexikální analýza
- Syntaktická analýza výrazů
- Rekurzivní syntaktická analýza
- Dokumentace

Tomáš Běhal

- Lexikální analýza
- Dokumentace
- Prezentace

Šimon Benčík

- Rekurzivní Syntaktická analýza
- Dokumentace
- Prezentace

1.1 Detaily o práci v týmu

Zadáním projektu bylo v jazyku C implementovat překladač jazyka IFJ22, který je podmnožinou jazyka PHP. Překladač kód přeloží do tříadresného kódu IFJ22code, který je poté interpretován.

Na projektu jsme začali dělat brzy po jeho zadání. Po složení týmu jsme se dohodli na verzovacím systému a komunikačních kanálech. Pro verzovací systém jsme se rozhodli použít Git hostovaný na platformě Github. Pro komunikaci jsme zvolili Discord. Protože někteří z nás znali strukturu překladače z vlastních projektů, tak jsme nebyli omezení čekánim na přednášky.

Rozhodli jsme se, že každý z nás bude pracovat alespoň částečně na různých částech projektu pro hlubší pochopení jednotlivých částí. K tomu jsme využívali techniky jako párové programovaní, nebo technologii Live Share ve Visual Studio Code, která nám umožňovala upravovat soubory současně.

2 Technické detaily

Jednotlivé struktury kódu jsou rozděleny do samostatných souborů, přičemž každý soubor obsahuje struktury podle jejich zařazení do struktury překladače. Deklarace těchto struktur a funkcí jsou ve stejnojmenných hlavičkových souborech. Nad každou strukturou a funkcii je uveden Doxygen komentář popisující její funkci a parametry.

2.1 Lexikální analýza

Lexikální analýza je implmentovaná pomocí stavového automatu. Stavový automat je implementován v souboru **scanner.c**, přičemž odpovídající stavový diagram je dále v této dokumentaci s legendou. Kód je převeden do struktur *Token*, které jsou implementovány v souboru **token.c**. Každý token obsahuje typ, atribut, číslo řádku a sloupce na kterém se nachází, tyto informace jsme využívali k jednoduššímu ladění. Atributy jsou typu union a typ tokenu určuje, jaký atribut je nastaven.

Pro načítaní tokenů, kterých atribut je delší než 1 znak, a chceme si jej ponechat, jsme si implementovali dynamické pole, jenž je implmentováno v souboru **str.c**. Jeden z atributů tokenu je právě onen dynamický pole znaků, do kterého se ukládají potřebné znaky. Ty jsou pak porovnány s tabulkou klíčových slov a případně převedeny na tokeny s odpovídajícím typem.
Jestli tento buffer obsahuje nějakou escape sekvenci anebo oktálovou či hexadecimální hodnotu, tak se převede na do korektní formy a uloží se. Do tohto bufferu sa ukládají aji číslice, které jsou následňe převedeny na odpovídající datový typ a uloženy do správneho atributu tokenu. V případě, že se vyskytne chyba, tak se vypíše chybová hláška pomocí modulu **err.c** a program se ukončí s odpovídajícím návratovým kódem.
Protože v subsetu jazyka PHP, pro který jsme implementovali překladač, je prolog a epilog fixního tvaru, proto jeho správnost kontrolujeme přímo v lexikální analýze, proto náš lexikální analyzátor začína v stavu SC_CODE_START, kde se jenom kontroluje správnost prologu.

2.2 Syntaktická analýza

Syntaktickou analýzu jsme implementovali pomocí rekurzivního sestupného parseru a precedenční analýzy pro výrazy.

2.2.1 Rekurzivní syntaktická analýza

Syntaktická analýza je implementovaná pomocí rekurzivního sestupného parseru v souboru **parser.c**. Funguje na základě pravidel. Odpovídající LL tabulka se nachází dále v této dokumentaci.
Parser je implementován jako řídicí struktura celého překladu, a proto obsahuje i strukturu scanneru a generátoru kódu, přičemž volá metodu scanneru scanner_get_next, která načte další token a takto načíta vstupní zdrojový kód
Každé pravidlo je implementováno jako funkce, která má jako parametr parser a jeho stav, přičemž ten značí, jestli se nacházíme ve smyčce nebo funkci, a také počita v jak hlubokém zanoření se nacházime.
V každé funkci jsou tokeny porovnány s pravidly pomocí pomocných funkcí na kontrolu typů. Když narazí na neshodu, vypíše se syntaktická chyba a program se ukončí.

2.2.2 Syntaktická analýza výrazů

Analýza výrazů je implementovaná pomocí precedenční tabulky, samotná analýza je implementovaná v souboru **exp.c**. Zásobník na redukce pravidel je implementován jako dynamické pole v souboru **stack.c**. Obsahuje struktury token_term_t, které obsahují operaci, typ výsledku, levý a pravý operand. Tato struktura je implementovaná v souboru **token_term.c**.
Po zavoláni funkce *rule_exp*, se na základě načteného tokenu a tokenu na vrchu zásobníku zjišťuje, která akce se má provést. Akce je implementovaná jako funkce, která má jako parametr 2 tokeny a vrací příslušnou operaci na základě precedenční tabulky.

2.3 Sémantická analýza

Protože IFJ22 je dynamicky typovaný jazyk, tak jsme se rozhodli implementovat většinu sémantické analýzy za běhu programu. Staticky analyzujeme pouze jestli je funkce nebo proměnná definovaná a jestli je volána s počtem parametrů, který odpovídá požadovanému počtu.
Jednotlivé funkce a proměnné jsou ukládány do tabulky symbolů jenž je implementovaná pomocí hashovací tabulky, toto rozhraní je implementováno v souboru **symtable.c**. Kolize při hashovaní jsou řešeny pomocí řetězených seznamů.

2.4 Generování kódu

Generování kódu jsme implementovali za pomoci funkcí pro jednotlivé klíčové části programu v souboru **gen.c**. Funkce generují instrukce, které jsou dále předávány do dynamického pole, které je implementováno v souboru **str.c**. Dále je zde implementováno i generování vestavěných funkcí v souboru **buildin.c**. Podmínky a cykly jsou realizovány pomocí návěští, které mají příslušný název ukončen s _index, kde index je počet vygenerovaných návěští, kterou počítame pomocí atributu parseru s názvem *construct_ count*. Parametry generovaných funkcí se předávají přes zásobník.

2.5 Ostatní

Na překlad jsme použili Makefile, s jehož psaním jsme měli zkušenosti z předmětů IJC a IVS. Pro komentáře jsme vybrali Doxygen a dokumentaci, co byla závěrečná část projektu jsme psali v LaTeXu. Samotné diagramy jsme generovali pomocí Graphviz. LL-gramatiku jsme kontrolovali za pomoci dostupných nástrojů na internetu.

3 Datové struktury

Při implementaci jsme se snažili ulehčit si práci využitím spravných datových struktur, některé byly předepsány zadaním, jiné jsme si vybrali sami.

3.1 Tabulka s rozptýlenými položkami

Tato struktura byla využita jako lokální a globální tabulka symbolů. Při implementaci jsme se řídili znalostmi z předmětu IAL, a také IJC kde jsme ji implmentovali. Tuto implementaci [1] jsme převzali a upravili pro potřeby tohoto projektu. Klíč tabulky je ukazatel na char a hodnota je implementována jako struktura *htab_item*, která obsahuje ukazatel na další položku v řeťezeném seznamu, který byl použit na řešení kolizí. Dále obsahuje *htab_pair*, která obsahuje klíč a data, přičemž data jsou implementována jako union, která určuje jestli je položka funkce nebo proměnná. Na základě toho se pak určují další atributy, jako počet parametrů, typy parametrů, typ výsledku, datový typ atd.
Pro práci s tabulkou jsou implementovány různé pomocné funkce. Např.: funkci pro vložení funkce, proměnné, vyhledání položky, rozšíření tabulky atd.

3.2 Dynamické pole znaků

Pro lexikální a syntaktickou analýzu jsme implementovali dynamické pole znaků, které je implementováno v souboru **str.c**. Tato struktura má atributy: pole znaků, současnou délku a také kapacitu. Pole znaků je alokováno dynamicky a jeho kapacita se zvětšuje automaticky při volaní pomocných funkcí na vložení znaku, resp. řetězce. Napsali jsme také různé pomocné funkce, jako konkatenace dvou dynamických polí, vložení C řetězce nebo znaku. Všechny tyto funkce vyhodnocují, jestli je potřeba zvětšit kapacitu pole či nikoli. Také jsou implementovány funkce pro výpis pole za účelem ladění.

3.3 Zásobník

Pro syntaktickou analýzu výrazů jsme implementovali zásobník, jenž je implementován jako dynamické pole v souboru **stack.c**. Má atributy pole ukazatelů na struktury *token_term_t*, délku a kapacitu. Pole ukazatelů je alokováno dynamicky, jeho kapacita se zvětšuje při operaci push, pokud je potřeba. Dále jsou implementovány pomocné funkce, jako pop, top atd.
Take jsme potřebovali funkce pro vložení na zásobník za vrchní terminál, nebo vrácení vrchního terminálu. Pro účel ladění jsme také implementovali funkci *pprint_stack()*, která vypíše obsah zásobníku.

Reference

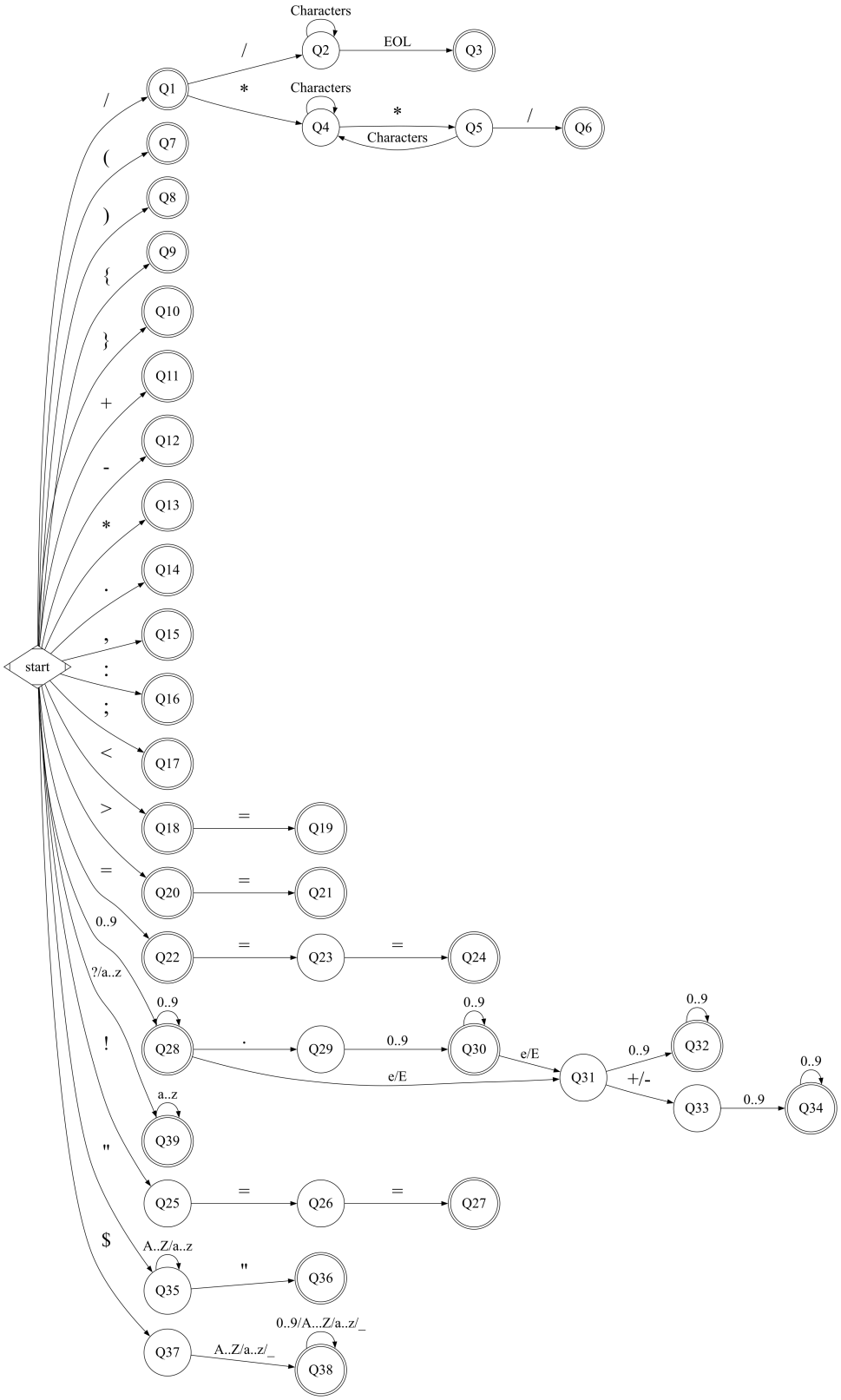
[1] KUCHAŘ, Josef. IJC Tabulka s rozptýlenými položkami [online]. 2022 [cit. 2022-12-05]. Dostupné z: https://github.com/JosefKuchar/ijc-hw2.

4 Závěr

Projekt jsme si napříč jeho složitosti užili, naučili jsme se hodně o tom, jak fungují části překladače, přičemž pro některé členy týmu to byla první zkušenost s programem takového rozsahu. Při řešení projektu jsme narazili na různé problémy, ale vždy se nám je společnými silami podařilo vyřešit. Proto jsme se také rozhodli ať se každý ze členů týmu podílí na všech částech projektu. Průběžne jsme projekt testovali automatickými testy, a jejich správnost nám potvrdilo pokusné odevzdání projektu. Po dobrém výsledku jsme se ještě rozhodli implementovat rozšíření CYCLES, které obsahuje podporu pro for cyklus a příkazy break a continue. S časem jsme díky dobrému rozdělení práce problém neměli. Zásluhu na tom mělo i to, že jsme počáteční části projektu dokázali implementovat ještě předtím jak byla daná látka probírána na přednáškách.

5 FSM

5.1 Diagram



5.2 Legenda ke koncovým stavům

1. Q1 Děleno
2. Q3 Komentář
3. Q6 Více řádkový komentář
4. Q7 Levá kulatá závorka
5. Q8 Pravá kulatá závorka
6. Q9 Levá složená závorka
7. Q10 Pravá složená závorka
8. Q11 Plus
9. Q12 Mínus
10. Q13 Krát
11. Q14 Konkatenace řetězců
12. Q15 Čárka
13. Q16 Dvojtečka
14. Q17 Středník
15. Q18 Menší než
16. Q19 Menší nebo rovno
17. Q20 Větší než
18. Q21 Větší nebo rovno
19. Q22 Přiřazení
20. Q24 Rovná se
21. Q27 Nerovná se
22. Q28 Celé číslo
23. Q30 Desetinné číslo
24. Q32 Desetinné číslo s exponentem
25. Q34 Desetinné číslo s kladným/negativním exponentem
26. Q36 Řeťezec
27. Q38 Proměnná
28. Q39 Datový typ

6 LL

6.1 LL-Gramatika

1. <Program> → function <FunId> (<Param>) : <ReturnType> { <StatementList> } <Program>
2. <Program> → <Statement> <Program>
3. <Program> → ε

4. <StatementList> → <Statement> <StatementList>
5. <StatementList> → ε

6. <Statement> → if (<Exp>) { <StatementList> } else { <StatementList> }
7. <Statement> → while (<Exp>) { <StatementList> }
8. <Statement> → for (<ExpFor> ; <OptionalExp>; <ExpFor>) {<StatementList>}
9. <Statement> → break ;
10. <Statement> → continue ;
11. <Statement> → return <OptionalExp> ;
12. <Statement> → <FunId> (<CallParam>) ;
13. <Statement> → <Exp> ;
14. <Statement> → <VarId> = <Value> ;

15. <Value> → <Exp>
16. <Value> → <FunId> (<CallParam>)

17. <Param> → <DataType> <VarId> <AdditionalParams>
18. <Param> → ε

19. <AdditionalParams> → , <DataType> <VarId> <AdditionalParams>
20. <AdditionalParams> → ε

21. <CallParam> → <Term> <AdditionalCallParam>
22. <CallParam> → ε

23. <AdditionalCallParam> → , <Term> <AdditionalCallParam>
24. <AdditionalCallParam> → ε

25. <Term> → <VarId>
26. <Term> → <Literal>

27. <OptionalExp> → <Exp>
28. <OptionalExp> → ε

29. <ExpFor> -> <VarId> = <Value> ;
30. <ExpFor> -> ε

6.2 LL-Tabulka

	function	<FunId>	(	):	<ReturnType>	{	}	if	<Exp>	)	else	;
<Program>	1	2						2	2			
<StatementList>		4					5	4	4			
<Statement>		12						6	13			
<Value>		16							15			
<Param>				18								
<AdditionalParams>				20								
<CallParam>										22		
<AdditionalCallParam>										24		
<Term>												
<OptionalExp>									27			28
<ExpFor>										30		30

	for	while	break	continue	return	<VarId>	=	<DataType>	,	<Literal>	$
<Program>	2	2	2	2	2	2					3
<StatementList>	4	4	4	4	4	4					
<Statement>	8	7	9	10	11	14					
<Value>											
<Param>								17			
<AdditionalParams>									19		
<CallParam>						21				21	
<AdditionalCallParam>									21		
<Term>						25				26	
<OptionalExp>											
<ExpFor>						29					

7 Precedenční tabulka

7.1

	*	/	+	-	.	<	<=	>	>=	===	!==	(	)	ID	IN	FL	ST	NI	$
*	R	R	R	R	R	R	R	R	R	R	R	L	R	L	L	L	L	L	R
/	R	R	R	R	R	R	R	R	R	R	R	L	R	L	L	L	L	L	R
+	L	L	R	R	R	R	R	R	R	R	R	L	R	L	L	L	L	L	R
-	L	L	R	R	R	R	R	R	R	R	R	L	R	L	L	L	L	L	R
.	L	L	R	R	R	R	R	R	R	R	R	L	R	L	L	L	L	L	R
<	L	L	L	L	L	X	X	X	X	X	X	L	R	L	L	L	L	L	R
<=	L	L	L	L	L	X	X	X	X	X	X	L	R	L	L	L	L	L	R
>	L	L	L	L	L	X	X	X	X	X	X	L	R	L	L	L	L	L	R
>=	L	L	L	L	L	X	X	X	X	X	X	L	R	L	L	L	L	L	R
===	L	L	L	L	L	X	X	X	X	X	X	L	R	L	L	L	L	L	R
!==	L	L	L	L	L	X	X	X	X	X	X	L	R	L	L	L	L	L	R
(	L	L	L	L	L	L	L	L	L	L	L	L	E	L	L	L	L	L	R
)	R	R	R	R	R	R	R	R	R	R	R	X	R	X	X	X	X	X	R
ID	R	R	R	R	R	R	R	R	R	R	R	X	R	X	X	X	X	X	R
IN	R	R	R	R	R	R	R	R	R	R	R	X	R	X	X	X	X	X	R
FL	R	R	R	R	R	R	R	R	R	R	R	X	R	X	X	X	X	X	R
ST	R	R	R	R	R	R	R	R	R	R	R	X	R	X	X	X	X	X	R
NI	R	R	R	R	R	R	R	R	R	R	R	X	R	X	X	X	X	X	R
$	L	L	L	L	L	L	L	L	L	L	L	L	X	L	L	L	L	L	R

Pozn.: L : <, R : >, E : =, X : Invalid