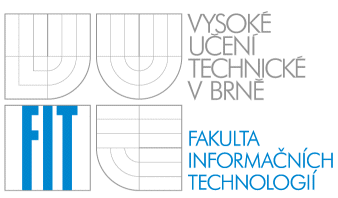
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



Dokumentace k projektu do IFJ a IAL

**Implementace překladače imperativního jazyka IFJ17**

Tým 022, varianta II

**Rozšíření:** BASE, UNARY, FUNEXP, BOOLOP, IFTHEN, CYCLES

Martin Omacht – xomach00, (vedoucí) … 34%

Zdeněk Chovanec – xchova19 … 33%

Petr Hendrych – xhendr03 … 33%

**1. Úvod**

Cílem projektu bylo vytvoření překladače pro imperativní jazyk IFJ17. Jazyk IFJ17 je podmnožinou jazyka FreeBASIC, který je založen na jazyce BASIC. Při registracích jsme si vybrali variantu II, která spočívala v řešení projektu za pomoci tabulky rozptýlených prvků.

**2. Etapy projektu**

Projekt jsme zahájili vytvořením prostředí pro vývoj. {MÍSTO PRO TVOJÍ KREATIVNÍ TVORBU ZDENDO}

**2.1 Lexikální analyzátor**

Lexikální analyzátor byl prvním krokem k vytvoření tohoto projektu. Celý lexikální analyzátor je založen na konečném automatu, který jsme navrhli tímto způsobem:

a jeho implementace koresponduje se tímto schématickým návrhem. Lexikální analyzátor je samostatná jednotka, která je podřízena syntaktickému analyzátoru. Komunikace lexikálního analyzátoru a syntaktického analyzátoru probíhá obousměrně.

**2.2 Syntaktický analyzátor**

Syntaktický analyzátor se dělí na dvě části:

## Analýza konstrukcí a příkazů

Nejdříve bylo nutné navrhnout LL gramatiku. Aplikovali jsme algoritmy, které nám byly prezentovány na přednáškách. Těmito postupy jsme sestavili množiny: *empty*, *first*, *follow* a nakonec *predict*, který nás dovedl k LL tabulce. LL tabulka je implementovaná pomocí řídké tabulky. Prediktivní syntaktickou analýzu jsme shledali jako efektivnější volbu, co se týče implementování algoritmu.

## Analýza výrazů

Analýza výrazů se prováděla pomocí precedenční tabulky a přidružených gramatických pravidel. Jelikož jsme implementovali rozšíření *UNARY*, museli jsme zde také detekovat, zda se jedná o unární mínus nebo binární mínus na základě předchozího zpracovávaného lexému.

**2.3 Sémantická analýza**

Tato část představovala největší výzvu pro náš tým. SA byla tvořena funkcemi, které byly přiřazené k určitým pravidlům gramatiky. Při použití pravidla se sémantickou funkcí se tato funkce vloží na vrchol zásobníku a následně se při zpracování lexému se předá tento lexém funkci na vrcholu tohoto zásobníku. V této sémantické funkci se pak pomocí stavového automatu zpracuje tento lexém (například kontrola typu, existence proměnné, vygenerování instrukce). Při každém volání sémantické funkce se vyhodnotí jeden stav konečného automatu.

**2.4 Generování kódu**

Původní plán byl generace abstraktního syntaktického stromu a následně z něj generace kódu, avšak neshledali jsme v tomto řešení žádnou výhodu, ale spíše časové zdržení. Proto jsme se rozhodli rovnou generovat instrukce tří adresného kódu do tří různých seznamů a to globálního seznamu, do kterého se ukládali definice globálních proměnných a jejich inicializace. Dále seznam instrukcí, kde se ukládali definice funkcí. A v neposlední řadě seznam pro instrukce hlavního těla programu. Na konci analýzy celého vstupního programu se nejdříve vypíší instrukce z globálního seznamu, poté hlavní tělo programu a nakonec funkce.

**Rozdělení práce**

Martin Omacht

* vůdce,
* hlavní programátor,
* implementace syntaktické a sémantické analýzy, generace kódu,
* testy

Zdeněk Chovanec

* hlavní návrhář,
* návrh konečného automatu lexikálního analyzátoru,
* návrh gramatik pro syntaktický analyzátor,
* pomocný programátor,
* implementace syntaktické analýzy pro výrazy,
* implementace sémantických kontrol a generace kódu,
* implementace tabulky symbolů,
* testy

Petr Hendrych

* implementace lexikálního analyzátoru
* vestavěné funkce v IFJcode17
* dokumentace
* prezentace

**Rozšíření**

**GLOBAL**

Toto rozšíření přidávalo možnost definování globální a statické proměnné. Implementace globální proměnných nepředstavovala žádný problém, avšak statické proměnné si musí uchovávat hodnotu mezi jednotlivými volání funkce. Toto jsme vyřešili tak, že jsme tyto proměnné udělali globálními a přidali jsme jim prefix *S* a název funkce, ve kterých jsou definovány (aby nedošlo ke kolizím s ostatními proměnnými).

**UNARY**

Implementace operace přiřazení s aritmetickou operací se jednoduše vyřešilo provedením aritmetické operace a následní přirazení do proměnné. U unárního mínus bylo potřeba rozpoznat, zda se jedná o unární nebo binární mínus. Viz (odkaz na anal. výraz) {ZDENDO TOTO TAM NEPIŠ ALE DEJ TAM TEN ODKAZ}

**BASE**

Rozšíření BASE ovlivnilo pouze lexikální analyzátor, kde po načtení čísla v jiné soustavě se převedlo do dekadické soustavy a dále se pracovalo pouze s dekadickými čísly.

**CYCLES**

{ZDENDO TO SI NAPIŠ SÁM!}

**FUNEXP**

Toto rozšíření se implementovalo jednoduše pomoci vyhodnocování výrazů na datovém zásobníku, protože parametry funkcí předáváme datovým zásobníkem a stejně jako návratovou hodnotu funkce. Menší problém představovala implicitní konverze datového typu předávaných parametrů při volání funkce, jelikož parametry již byly vloženy na datový zásobník a nebyl k nim přístup pro konverzi. Toto bylo vyřešeno v definici funkce, kde pokud byl parametr typu INT nebo DOUBLE bylo k nim vygenerováno dynamické přetypování. To spočívalo, v zjištění datového typu pomocí instrukce TYPE a následně přeskočení konverze, pokud datový typ odpovídal definici parametru.

**IFTHEN**

Podpora IF bez větve ELSE nebyla složitá. Akorát u podpory větví ELSEIF bylo složitější generování návěstí. Což bylo vyřešeno dvěma identifikátory. Jeden pro celý IF a druhý pro aktuální větev ELSEIF. Z těchto identifikátorů se následně generovaly jednotlivé návěstí s prefixy ELSE nebo ENDIF. Přičemž ELSE může být další ELSEIF.

**BOOLOP**

{PRÝ JSI TO DĚLAL TY, ZDENDO, TAK MÁŠ OPĚT PROSTOR KE SVÝM KREACÍM}

**Komunikace v týmu**

Jelikož dva ze tří členů bydlí na kolejích a ve stejném pokoji, tak komunikace spočívala převážně s posledním členem týmu. Jako online komunikaci jsme si vybrali Messegner, kde jsme si založili skupinový chat pro snadnou komunikaci. Pokud se však objevil větší problém, tak jsme se sešli všichni osobně

Pro sdílení souborů projektu a verzování jsme zvolili verzovací systém Git hostovaný na službě Github. Github nám také umožňoval code review nově nahraných verzí nebo například přidávání issues pokud něco nefungovalo.

**Projekt v číslech**

* počet řádků:
* počet commitů:
* počet souborů:

**Shrnutí**

Projekt byl něco zcela nového pro všechny z nás a zabral opravdu velkou část semestru. Museli jsme se naučit mnohému jako například: práce v týmu a komunikace v něm, práce se systémem Git apod. Také jsme využili pokusného odevzdání, po kterém jsme mohli už pouze dolaďovat menší nedostatky a implementovat zbývající rozšíření, které jsme ještě měli v plánu.

**Čerpané zdroje**

* přednášky a slajdy IFJ a IAL
* https://www.youtube.com/watch?v=Kdx5HOtb-Zo&index=25&list=PLFB9EC7B8FE963EB8