Semestrální práce

z předmětu KIV/PC

Interpret podmožiny jazyka LISP

Jan Hinterholzinger
15. ledna 2021

Kapitola 1

Zadaní

Naprogramujte v ANSI C přenositelnou konzolovou aplikaci, která bude fungovat jako jednoduchý interpret podmnožiny funkcionálního programovacího jazyka Lisp. Vstupem aplikace bude seznam příkazů v jazyce Lisp. Výstupem je pak odpovídající seznam výsledků vyhodnocení každého výrazu.

Program se bude spouštět příkazem `lisp.exe [<vstupní soubor>]`. Symbor ¡vstupní soubor¿ zastupuje nepovinný parametr - název vstupního souboru se seznamem výrazů v jazyce Lisp. Není-li první parametr uveden, program bude fungovat v interaktivním módu, kdy se příkazy budou provádět přímo z konzole do interpretu.

Vámi vyvinutý program tedy bude vykonávat následující činnosti:

1. Při spuštění bez parametrů bude čekat na vstup od uživatele. Zadaný výraz vyhodnotí a bude vyžadovat další vstup dokud nebude uveden příkaz `(quit)`.

2. Při stupštění s parametrem načte zadaný vstupní soubor, každý výraz v něm uvedený vypíše na obrazovku, okamžitě vyhodnotí a výsledek vypíše na obrazovku. Po zpracování posledního výrazu dojde k ukončení programu. Proto nemusí být jako poslední výraz uveden výraz `(quit)`. Na jedné řádce v souboru může být uvedeno více samostatných výrazů a program je musí být schopen správně zpracovat.

Váš program může být během testování spuštěn například takto:
`lisp.exe text.lisp` Hotovou práci odevzdejte v jediném archivu typu ZIP prostřednictvím automatického odevzdávacího a validačního systému. Postupujte podle instrukcí uvedených na webu předmětu. Archiv nechť obsahuje všechny zdrojové soubory potřebné k přeložení programu, makefile pro Windows i Linux (pro překlad v Linuxu připravte soubor pojmenovaný `makefile`

a pro Windows `makefile.win`) a dokumentaci ve formátu PDF vytvořenou v typografickém systému TEX, resp. LATEX. Bude-li některá z částí chybět, kontrolní skript Vaši práci odmítne.

Kapitola 2

Analýza úlohy

Úloha v sobě obsahuje několik zvlášť řešitelných podproblémů, které sa každý dají řešit několika způsoby. To směřuje k tomu, že tato analýza není jediná možná a pravděpodobně není nejlepší možná.

Úlohu bych rozdělil na několik částí:

- Interpretace výrazů
- Obsluha operátorů a funkcí
- Ukládání hodnot proměnných

2.1 Interpretace výrazů

Před vyhodnocováním výrazu je vhodné výraz interpretovat v nějaké struktuře. Pro vyhodnocování Lisp výrazů se velmi vhodně nabízí implementace zásobníku, protože postup vyhodnocování a zapisování Lisp výrazů odpovídá právě chování zásobníku.

Do zásobníku se postupně načte výraz. Po načtení výrazu začínáme ze zásobníku prvky odebírat.

Lisp výrazy mají následující strukturu:

1. **otevírací závorka** – Označuje začátek výrazu
2. **operátor nebo funkce** – Prvek označující typ operace, který bude prováděn.
3. **operandy** – Prvky výrazů, se kterými se bude podle operátoru pracovat.
4. **uzavírací závorka** – Označuje konec výrazu

Na základě této struktury víme v jakém pořadí se budou data v zásobníku vyskytovat a tedy jak s nimi zacházet. Například první prvek po otevření výrazu je operátor.

Dle příkladu (Tabulka 2.1) můžeme vidět zásobník (šedé pozadí = vrchol zásobníku) s načteným výrazem. Jako první (kromě otevírací závorky) vyskočí operátor resp. znak „+“, který značí operaci sčítání. Na Základě tohoto poznatku víme, že do konce výrazu se budou veškeré operandy sčítat. Tedy postupně se budou ze zásobníku vybírat všechnyprvky, které se budou odděleně sčítat, dokud se ze zásobníku nevybere ukončovací závorka. V tu chvíly výraz končí a jeho výsledek se zapíše zpět do zásobníku.

(	5
+	
2	
3	
)	

Tabulka 2.1: Zásobník s výrazem před začátkem vyhodnocování a po vyhodnocení

Tento elementární příklad pouze nastiňuje jádro řešení celého problému a je potřeba komplikovanější řešení pro to, aby bylo možné řešit i různé podvýrazy. Pro finální řešení použijeme dva zásobníky. Jeden hlavní, který bude držet vstupní výraz, jednotlivé výrazy bude nahrazovat jejich výsledky a nakonec po všech operacích se v něm bude vyskytovat pouze finální výsledek. A druhý kam se budou vkládat z hlavního zásobníku jednotlivé podvzorce, které se vyhodnotí obdobně jako z příkladu výše a výsledek se zapíše do hlavního zásobníku. Výsledek tedy tohoto parciálního výrazu se zapíše na místo do výrazu (v hlavním zásobníku), na kterém se předtím nacházel právě vypočtený výraz.

2.2 Obsluha operátorů a funkcí

Získaný výraz je potřeba samozřejmně vyhodnotit. A to na základě operátoru, podle kterého je potřeba program rozvětvit. Nasťiňuje se několik možností. Například použití konstrukce `if` (nebo `switch`), s kterou můžeme mít ošetřený každý operátor nebo funkci a samotné provedení výpočtu/funkce se provede v její kladné větvi. Tato možnost by byla sice funkčí, ale značně by prodloužila kód v hlavním cyklu a je tedy vhodné tento kód oddělit od řídícího kódu.

Mohli bychom tento kód vložit do zvlášť funkce, která se bude starat o vykonání výrazu. Každopádně i při tomto použití je i při tak malém počtu operátorů a funkcí velmi dlouhá a tedy i nepřehledná. Proto je vhodné jednotlivé operace rozdělit do jednotlivých funkcí, které přijmou vstup ze zásobníku a výsledek zapíší zpět do hlavního zásobníku. Nevýhoda tohoto postupu bude repetetivní ověřování ve všech funkcí operací.

Jedním z problémů, který komplikuje zápis výrazů a jejich vyhodnocení jsou funkce *QUOTE* a *LIST*. Jejich zápis se jeví stejně jako zápis jakéhokoli jiného operátoru. Každopádně pro další operace v výsledkem těchto funkcí je potřeba rozeznat, že se jedná o nevyhodnocovaný výraz nebo seznam a né o běžný podvýraz. Z toho důvodu a z důvodu přehlednosti jsem se rozhodl použít pro tyto prvky jiné závorky, než obyčejné kulaté. Pro *QUOTE* výraz jsem zvolil závorky hranaté ([pro otevření výrazu a] pro uzavření výrazu). Obdobně jsou použity složené závorky pro označení seznamů (**{** pro otevření výrazu a **}** pro uzavření výrazu).

2.3 Ukládání hodnot proměnných

Neodmyslytelnou částí programu je také ukládání proměnných. Ze zadání není zřejmé, zda proměnné budou pouze znaky (jednomístné řetězce např. "a"nebo "x") nebo více znakové (např. "neznama"). Rozhodl jsem se jako proměnné brát jako více znakové řetězce. Tím je problém trochu náročnější, ale ne o moc. V případě jednoznakové proměnné je řešení velice přímé. Protože máme omezený (a né příliš velký) počet znaků písmen ('a' až 'z'), tak se nabízí jednoduchá lookup tabulka, která bude obsahovat celočíselné hodnoty. Přístup do takové tabulky je v konstantním čase a je velmi jednoduchá na implementaci.

V případě víceznakového názvu proměnné se vyskytuje hned několik komplikací. Délka názvů proměnných není konstantní. Může se objevit proměnná s jednoznakovým názvem stejně jako proměnná s desetiznakovým názvem. Tento problém je ale snadno vyřešitelný, protože můžeme ukládat odkaz na

tento řetězec, který má vždy konstantní velikost. Druhý problém spočívá v struktuře, do které se budou proměnné ukládat a ze které se budou vyhledávat. Ideálním kandidátem na tuto roli je HashMapa, ale z důvodu její složité implementace a volby vhodné hashovací funkce zvolím implementaci spojového seznamu. V něm sice může mít vyhledávání v nejhorším případě až lineární složitost, ale vzhledem k předpokládanému počtu ukládaných proměnných tato nevýhoda nehraje příliš velkou roli a spokojíme se se spojovým seznamem.

Kapitola 3

Popis implementace

3.1 Implementace zásobníku

Zásobník je implementován polem s pevně danou velikostí (danou konstantou). Velikost pole se tedy nepřizpůsobuje velikosti vstupních dat. Pro práci se zásobníkovou strukturou slouží funkce v souboru `stack.c`.

3.2 Implementace spojového seznamu

Jako implementace seznamu byla zvolena spojová struktura. Funkce pro účel spojového seznamu nachází v souboru `list.c`. Spojová struktura má udržovat názvy proměnných a jejich hodnotu. To zmamená, že všechny prvky spojového seznamu obsahují pointer do paměti, kde se nachází řetězec jako název proměnné a celočíselnou hodnotu jako uloženou hodnotu v proměnné. Také pro potřeby propojení ve spojovém seznamu je potřeba určit následující prvek. Samotná struktura si tak potřebuje udržovat pouze odkaz na první prvek seznamu, ze kterého se postupně může dostat k ostatním prvkům.

3.3 Pomocné funkce pro práci s řetězci a zásobníkem

Při konstrukci programu se vyskytly chvíle, kdy bylo potřeba implementovat některé procedury, které se často opakovaly do funkcí. Tyto funkce se zaměřovaly zejména na čtení prvků z řetězce nebo zásobníků, práce s řetězci atd. Tyto funkce jsou obsaženy v souboru `loader.c`.

Příkladem takové funkce může být například `int split_line(char *, int)`, která v případě, že ve vstupním řetězci se nachází více výrazů vrátí

indexy konce (rozdělení) těchto výrazů.

Další často používanou funkcí je `int next_operator(stack *, char **)`. Tato funkce čte znaky ze zásobníku a skládá z nich řetězec dokud nenarazí na mezeru nebo dokud zásobník nebude prázdný. Takto se tedy v kódu získávají jednotlivé operátory nebo i operandy všech funkcí nebo operací. Výstupní řetězec se alokuje v této metodě, je tedy potřeba, aby po volání této funkce byl příslušná přidělená paměť také uvolněna.

Soubor obsahuje i funkce pro převod čísla na řetězec a obráceně. Obdobně také pro jejich zapsání do hlavního zásobníku.

3.4 Obsluha vyhodnocovacích funkcí

Samotné vyhodnocení jednotlivých výrazů se provádí v souboru funkcí `operators.c`. Zde se nachází všechny funkce pro obsluhu všech operátorů a funkcí programu. Všechny tyto obsluhující funkce jsou přiřazeny k značkám (řetězcům, které ve výrazu označují daný operátor nebo funkci). Tyto dvojice jsou k sobě přiřazeny ve struktuře `calc_oper` a všechny tyto struktury jsou obsaženy v konstantě pole `OPERATORS`. Při vyhledávání konkrétní operace se toto pole prochází a vyhledá příslušnou funkci ve funkci `fcalc get_handler(char[])`.

Po nalezení správné funkce se v ní provede žádaná operace s prvky z pomocného zásobníku. Všechny operandy jsou podrobeny kontrolou, zda se nejedná o název proměnné. V tomto případě by název byl nahrazen hodnotou proměnné. Po vykonání operace se výsledek zapíše do hlavního zásobníku a ukončí průběh funkce.

3.5 Proces vyhodnocení výrazu

Implementace centrální funkce pro vyhodnocení zadaného výrazu `int evaluate(char *, ilst *)` bude stručně shrnuta v této sekci.

Funkce nejprve zadaný řádek rozdělí na jednotlivé výrazy. Ty postupně po znacích vloží do hlavního zásobníku. Současně jsou provedeny procedury pro obsluhu seznamů a výrazu *QUOTE*. Při vkládání do zásobníku pokud program narazí na znak uzavírající některý výraz nebo podvýraz, zahájí jeho vyhodnocování. Nejprve tedy z hlavního zásobníku převede veškeré znaky do pomocného zásobníku dokud opět nenarazí tentokrát na závorku otevírající výraz.

Nyní máme v pomocném zásobníku celý výraz, který je potřeba vyhodnotit. Aby jsme věděli jak ho vyhodnotit, tak potřebujeme značku operátoru. Ta se nachází na první pozici výrazu, takže získáme první „slovo“ z pomocného

zásobníku a pokusíme se pro něj získat příslušnou obsluhující funkci funkcí `get_handler(...)`. Jakmile zjistíme obsluhující funkci, tak je zavolána a tím je tento podvýraz vyhodnocen a pokračuje se dále vkládáním dalších znaků ze vstupního řetězce. Na konci této procedury, kdy jsou všechny podvýrazy a poté samotný výraz vyhodnoceny, tak v hlavním zásobníku zůstane pouze výsledek.

Pro vypsání výsledku je potřeba jej celý převést do pomocného zásobníku, aby znaky výsledku byly ve správném pořadí. Poté pouze vyjmeme znak po znaku výsledek z pomocného zásobníku a vypíšešeme je. V případě, že výsledek obsahuje námi zavedené pomocné závorky, tak je přepíše zpět na obyčejné kulaté závorky výrazu.

3.6 Spouštěcí funkce programu

V hlavní funkci programu se vytvoří seznam pro ukládání proměnných a načtou se řádky souboru pomocí funkce `FILE fopen(...)` a ty postupně vyhodnotí. V případě, že argumenty neobsahují název souboru, tak se spustí nekonečná smyčka, která bude vyzívat uživatele k zadání výrazu dokud nezadá ukončující příkaz.

Kapitola 4

Uživatelská příručka

4.1 Přeložení a sestavení programu

4.1.1 Na operačním systému Windows s MinGW

Pro přeložení na OS Windows je zapotřebí přejmenovat soubor `MakeFile.win` na `makefile`. Pokud již takový soubor existuje, tak jej prosím smažte nebo přejmenujte.

Po předešlé akci můžete spustit příkaz `mingw32-make` a nechat překladač a linker sestavit celý program. Program se sestaví do spustitelného souboru `lisp.exe`.

4.1.2 Na operačních systémech Linux

Pro překlad a spuštění na operačním systému Linux je zapotřebí spustit příkaz `make`. Překladač a linker program přeloží a poté složí do spustitelného souboru `lisp`.

4.2 Spuštění programu

Program lze spustit se vstupním textovým souborem, který obsahuje seznam příkazů (výrazů), které se mají provést. Při spuštění je potřeba cestu k tomuto souboru přiložit do prvního spouštěcího argumentu. Program se se vstupním souborem spouští zadání následovného příkazu do příkazové řádky na OS Windows:
`lisp.exe <textový soubor>`,
kde místo `<textový soubor>` nahradíme cestou k vstupnímu souboru.
A pro systém Linux:

`./lisp <textový soubor>`,
kde rovněž `<textový soubor>` nahradíme cestou k vstupnímu souboru.

Program lze spustit i bez tohoto vstupního souboru. Tím se uživatel dostane přímo k zadávání výrazů do příkazové řádky. Program pro tento mód spustíme bez parametrů.
To znamená pro systém Windows příkaz
`lisp.exe`
a pro systém Linux příkaz
`./lisp`.

4.3 Ovládání programu

Ovládání programu je myšleno ruční zadávání výrazů do příkazové řádky. To je umožněno v případě spuštění programu bez parametrů, nebo s neplatnou cestou ke vstupnímu souboru.

Program akceptuje všechny výrazy dané přílohou v zadání **Odkaz na zadání**. Stručný výpis podporovaných příkazů můžeme nalézt v tabulce 4.1. Program lze ukončit zadáním výrazu `(quit)`.

Popis	Značka operátoru	Příklad
Sečtení všech operandů	+	(+ 1 2 5)
Odečtení všech operandů od prvního operandu	-	(- 5 3)
Roznásobení všech operandů	*	(* 6 2)
Celočíselné vydělení prvního operandu všemi operandy	/	(/ 8 2)
Kontrola na rovnost	=	(= 2 2)
Kontrola na nerovnost sousedních prvků	/=	(/= 3 2 2)
Kontrola na „menší než“ u sousedních prvků	<	(< 2 2 2 4)
Kontrola na „větší než“ u sousedních prvků	>	(> 2 2 2 4)
Kontrola na „menší než“ nebo rovnost u sousedních prvků	<=	(<= 2 2 2 4)
Kontrola na „větší než“ nebo rovnost u sousedních prvků	>=	(>= 2 2 2 4)
Zkrácený zápis quote výrazu	’	’(1 2 a)
Výraz nebude vyhodnocen	quote	(quote A)
Zadané operandy jsou seznam	list	(list 1 2 3 4)
Ukončení programu	quit	(quit)
Vrácení prvního prvku seznamu	car	(car 1 2 3 4)
Vrácení seznamu bez prvního prvku	cdr	(cdr 1 2 3 4)
Nastavení hodnoty proměnné	set	(set ’a 4)

Tabulka 4.1: Tabulka operátorů a funkcí

Kapitola 5

Závěr

Program funguje podle požadavků daným zadáním. Tedy přijímá výrazy jazyka Lisp a vyhodnocuje je. Podporuje minimální seznam operátorů a vyhodnocuje je dle očekávání. Program vypisuje chybovou hlášku na události, které zadání zvolilo a všechna alokovaná paměť se dle programu Valgrind uvolňuje.

Vhodnost použití některých struktur nebo jejich implementace je sporná, ale vede ke zdárnému výsledku a pro potřeby semestrální práce splňuje svůj účel.

Veškeré zdrojové soubory i s funkcemi jsou okomentovány a jsou použity konstanty preprocesoru.