Dokumentace k projektu pro předměty IFJ a IAL

Implementace interpretu imperativního jazyka IFJ14

Tým 085, varianta a/3/I

10. prosince 2014

Autoři: Vojtěch Bastl, xbastl03

Zdeněk Březina, xbrezi15

Marek Dokulil, xdokul03

Jiří Fňukal, xfnuka01

Lukáš Hubl, xhubll00

Fakulta Informačních Technologií

Vysoké Učení Technické v Brně

Obsah

1 Úvod 1

2 Analýza problému ..... Chyba! Záložka není definována.

2.1 Zadání projektu ..... **Chyba! Záložka není definována.**

2.2 Iterační metoda výpočtu ..... **Chyba! Záložka není definována.**

2.3 Doba trvání programu ..... **Chyba! Záložka není definována.**

2.4 Analýza vstupních dat 3

2.5 Přesnost výstupních dat 3

3 Řešení problémů 5

3.1 Řešení problémů vstupních dat 5

3.2 Jednotlivé řady a vzorce pro výpočet funkcí 5

3.3 Druhá odmocnina 6

3.4 Arcus sinus 6

3.5 Výpočet vnitřních úhlů v trojúhelníku 6

4 Specifikace testů 7

5 Popis řešení 8

5.1 Ovládání programu 8

5.2 Vlastní implementace 8

5.3 Optimalizace 9

6 Závěr 10

7 Reference 11

Příloha A 12

Úvod

Tématem této práce je implementace překladače pro jazyk IFJ14. Překladač byl rozdělen na pět samostatných problémů, tak aby mohla být implementace rozdělena mezi členy týmu.

Překladač byl po vzájemné domluvě rozdělen na tyto části: Lexikální analýza, Syntaktická analýza bez zpracování výrazů, Syntaktická analýza pro výrazy, složitější struktury a interpret

Vzhledem k tomu, že tyto podproblémy jsou spolu úzce spjaty, implementace vyžadovala neustálou spolupráci a komunikaci v týmu. Jako výjimku lze požadovat lexikální analýzu, která zpracovává samotný vstupní soubor, a proto musí být implementována před řešením ostatních problémů. Následně byla využita při řešení zbylých částí.

1. Implementace

V této kapitole se zaměříme na řešení problémů jednotlivých částí překladače a zároveň implementace zadaných algoritmů pro vyhledávání a řazení v řetězci.

1.1 Lexikální analýza

Lexikální analýza byla implementovaná jako konečný automat v souboru scanner.c a je volaná jako funkce get_token, která načítá jednotlivé znaky ze zdrojového souboru a vrací typ tokenu, které jsou spolu se stavy konečného automatu definované v souboru scanner.h. Schéma automatu je vyobrazeno v příloze.

1.2 Knuth-Morris-Prattův algoritmus (KMP)

Tento algoritmus ke své práci využívá konečný automat. Algoritmus se dá rozdělit na dvě části. V první části algoritmus prochází vzorek, který vyhledáváme a do pole Fail ukládá jednotlivé indexy, ke každé pozici ve vzorku pouze jeden, které udávají, na kterou pozici se máme vrátit, dojde-li k neshodě na aktuální pozici. V druhé části algoritmus prochází vzorek a zároveň řetězec, v kterém vyhledáváme. Pokud dojde k neshodě, posuneme se ve vzorku na pozici, která je v poli Fail pro aktuální pozici a znova porovnáváme vyhledávaný vzorek se zadaným řetězcem. Algoritmus tedy proběhne m+n krát, přičemž m je délka řetězce v kterém vyhledáváme a n je délka vzorku, který vyhledáváme. V nejhorším případě se může m rovnat n a algoritmus vlastně proběhne 2n krát a má tedy lineární složitost. Algoritmus byl implementován jako jedna funkce v souboru ial.c a byl pojmenován find. Algoritmus nám vrací pozici kde došlo ke shodě vzorku s řetězcem a při neshodě vrací 0.

1.3 Shell sort – řazení se snižujícím se přírůstkem

Výše zmíněný algoritmus si rozděluje řetězec na podřetězce složené se znaků vzdálených od sebe o krok. Shell sort začíná s krokem rovným polovině délky řetězce. Na jednotlivé podřetězce aplikuje Booble sort, to znamená že, každý podřetězec prochází od počátku do konce a vyměňuje prvky, tak aby prvek s větší ordinální hodnotou byl vždy vpravo. Booble sort skončí, až když při průchodu nedojde k výměně žádné dvojice prvků. Po seřazení všech podřetězců se krok zmenší na polovinu a znovu se na podřetězce aplikuje Booble sort. Algoritmus skončí, pokud se krok rovná 0. Algoritmus byl implementován jako funkce sort v souboru ial.c a vrací seřazený řetězec. Shell sort je nestabilní metoda která pracuje in-situ,

to znamená, že nemusí respektovat pořadí položek se stejným klíčem a k seřazení nepotřebuje další pomocné pole.

1.4 Analýza vstupních dat

U vstupních dat musíme pamatovat na různorodost jejich zadaní a ošetřovat je k jednotlivým funkcím a zohlednit obor hodnot jednotlivých funkcí, aby program nezkolaboval nebo nepředával nesmyslné hodnoty. Další častí ošetření je počet parametrů, který je u jednotlivých funkcí různý a musíme to v programu zohlednit.U trojúhelníku potom musíme ověřovat, zda se vůbec o trojúhelník jedná.

1.5 Přesnost výstupních dat

U výstupních dat zase musíme počítat s nepřesností některých výsledků. Program nemusí pracovat správě pro mezní hodnoty ve funkciarcus sinus. Na toto jsme museli při psaní programu myslet a najít vhodný vztah pro přepočet v mezních hodnotách.

2 Syntaktická analýza

Zkoumání, zda byl vstupní program správně syntakticky napsán, musíme rozdělit na dvě části. První část je kontrola syntaxe neobsahující výrazy. Výrazy poté řešíme pomocí druhé části. Tyto dvě části spolu musí vzájemně komunikovat a musí být vhodně použity na jednotlivé části vstupního souboru.

2.1 Syntaktická analýza neobsahující výrazy

Problém správnosti syntaxe je řešen pomocí vhodné LL gramatiky a následně její implementace pomocí rekurzivního sestupu. LL gramatika je navržena vzhledem k zadání a její každý jednotlivý neterminál je charakterizován pomocí funkce. Pokud k jednomu neterminál přísluší více LL pravidel, jejich rozlišení je zajištěno pomocí větvení. Pokud se zjistí, že musí být zpracováván výraz, program zavolá modul na zpracování výrazu a následně se znovu vrátí na rekurzivní sestup LL gramatikou. Tato technika je uplatněna i na volání funkcí ve vstupním programu, kdy nejprve syntaktická analýza zjistí, zda daná funkce je deklarována a následně při každém parametru zavolá zpracování výrazu. Syntaktická analýza zároveň pokrývá část typové kontroly konkrétně přiřazení a parametry volání funkcí včetně jejich návratového typu. Samotné znění LL gramatiky je přibaleno v příloze.

3 Řešení problémů

3.1 Řešení problémů vstupních dat

Vstupní data můžeme dostat velmi různorodá, máme totiž funkce bez parametru, funkce s jedním parametrem a se šesti parametry. Nejprve si tedy zjistíme, kterou z funkci budeme volat a do pomocné proměnné si uložíme její příznak. Podle příznaku dále ošetřujeme, kolik parametrů bylo zadáno, jestli bylizadány přijatelně. U všech funkci kromě funkce help, máme alespoň jeden parametr, který je číslo. Tento převod musíme ošetřit, zda vůbec lze parametr na číslo převést a dále u jednotlivých funkcí musíme zjišťovat, jestli zadaný parametr funkce leží v jejím oboru hodnot. U funkce my_triangle navíc musíme ošetřit, zda se jedná o trojúhelník. Toto řešíme pomocí jednoduchého pravidla: Součet dvou stran v trojúhelníků musí být vždy delší než třetí strana.

3.2 Jednotlivé řady a vzorce pro výpočet funkcí

- Druhá odmocnina (Newtonova metoda)

  $y_0 = x\ ;\ y_{i+1} = \frac{1}{2}\left(\frac{x}{y_i} + \ y_i\right)$; kde x je vstupní hodnota (1)

- Arcus sinus (Taylorova věta)

  $\sin^{-1} = x + \ \frac{1 \cdot x^3}{2 \cdot 3} + \frac{1 \cdot 3 \cdot x^5}{2 \cdot 4 \cdot 5} + \frac{1 \cdot 3 \cdot 5 \cdot x^5}{2 \cdot 4 \cdot 6 \cdot 7} + \ \ldots;$ kde x je vstupní hodnota (2)

- Vztah pro výpočet následujícího členu arcus sinus

  $y_0 = x;\ y_{i+1} = \frac{y_i \cdot x^2 \cdot (2+i \cdot 2)^2}{(3+i \cdot 2) \cdot (4+i \cdot 2)}$; kde x je vstupní hodnota (3)

- Vztah mezi arcus sinus a arcus cosinus

  $\cos^{-1} x = \frac{\pi}{2} - \ \sin^{-1} x$; kde x je vstupní hodnota (4)

- Vztah pro optimalizaci mezních hodnot

  $\sin^{-1} x = -\frac{\pi}{2} + \sin^{-1}\left(\sqrt{1 - x^2}\right); pro\ x < 0$; kde x je vstupní hodnota (5)

  $\sin^{-1} x = \frac{\pi}{2} - \sin^{-1}\left(\sqrt{1 - x^2}\right); pro\ x > 0$; kde x je vstupní hodnota (6)

- Cosinová věta

  $c^2 = a^2 + b^2 - a \cdot b \cdot \cos\gamma$ (7)

- Délka strany

  $\sqrt{a} = (Cx - Bx)^2 + (Cy - By)^2$ (8)

3.3 Druhá odmocnina

Druhou odmocninu implementujeme pomoci Newtonovy metody (1), kdy výsledek dopočítáváme pomocí předchozího výsledku. Pro vstupní hodnotu 0 vracíme, bez průběhu funkcí, hodnotu 0, pro urychlení běhu programu.

3.4 Arcus sinus

Arcus sinus implementujeme pomoci Taylorovy věty (2), z které jsme si vyjádřili vzorec pro výpočet následujícího členu rozvoje, pomocí členu předchozího (3). Funkce tedy počítá následující člen rozvoje a vždy ho přičte k výsledku. Předchozí člen a výsledek si musíme pamatovat. Funkci optimalizujeme pro kladné mezní hodnoty pomocí vzorce (5) a pro záporné mezní hodnoty pomocí vzorce (6).

3.5 Výpočet vnitřních úhlů v trojúhelníku

K výpočtu vnitřních úhlů v trojúhelníku nejprve potřebujeme délky stran. Ty dopočítáme podle vzorce (8) a poté je doplníme do cosinové věty (7). Poté už jen stačí přepočítat arcus cosinus na arcus sinus pomocí vzorce (4) a můžeme zavolat funkci pro výpočet arcus sinus, která nám vrátí hodnotu úhlu v radiánech. Toto zopakujeme pro všechny strany.

4 Specifikace testů

Z analýzy problémů vyplívá několik vstupních dat, které musíme testovat - nevhodný počet parametrů(neodpovídá přesné syntaxi), hodnoty mimo definiční obor funkce, body netvoří trojúhelník, chybně zadaný první argument(neodpovídá přesné syntaxi), argument není číslo, mezní hodnoty pro funkci arcus sinus.

Test 1: Druhá odmocnina

Spuštění s parametry	Očekávaný výstup
--sqrt 4	2.0000000000e+000
--sqrt 3	1.7320508079e+000
--sqrt 12	3.4641016151e+000
--sqrt	Spatnezadane argumenty
--sqrt -1	nan
--sqrt 0	0.0000000000e+000
	Není zadan argument.
sqrt	Spatnezadane argumenty.

Test 2: Arcus sinus

Spuštění s parametry	Výstup
--asin 0.5	5.2359877560e-001
--asin 0.333	3.3948337815e-001
--asin 12	Zadanecislonelezi v Definicnim oboru ktery je <-1;1>.
--asin 0.9999	1.5566540733e+000
--asin -0.9999	1.5566540733e+000
--asin	Spatnezadane argumenty.
	Není zadan argument.

Test 3: Trojúhelník

Spuštění s parametry	Výstup 1	Výstup 2	Výstup 3
--triangle 1 2 4 6 3 1	1.3909428270e+000	4.4610554894e-001	1.3045442776e+000
--triangle 5 9 5 2 4 7	4.6364760904e-001	1.9739555985e-001	2.4805494847e+000
--triangle -9 1 2 8 -3 6	1.2800905867e-001	1.8622284041e-001	2.8273607545e+000
--triangle 0 2 4 5	Nevhodnypocet parametru.		
--triangle 0 2 4 9 c6 9	Spatnezadanasouradnice x bodu C.		
--triangle 0 0 2 2 4 4	Body A, B, C netvoritrojuhelnik.		

5 Popis řešení

5.1 Ovládání programu

Program je řešený jako konzolová aplikace. Do samotné konzole se nic nepíše, vše je zadáno v parametrech při spuštění. V konzoli se tedy dozvíme jen výsledek dané operace.

Funkce můžeme volat s těmito parametry:

První argument	**Další argumenty**
--sqrt	x
--asin	x
--triangle	AxAyBx By CxCy
--help	

Argument --sqrt vrací druhou odmocninu z x.
Argument --asin vrací uhel v radianech.
Argument --triangle vrací vnitřní úhly v trojúhelníku A, B, C.
Argument -- help vypíše nápovědu.

5.2 Vlastní implementace

Program jako první vyhodnotí, jestli byl zadán argument. Pokud ne, vypíše chybovou hlášku a skončí.

Pokud byl zadaný argument voláme funkci vyhodnoceni, která vyhodnotí, který z možných argumentů byl zadán a uloží si příznak argumentu. U jednoargumentových funkcí zároveň převede druhý argument. Pokud nastane chyba v převodu, nebo argument není rovný s některým s přípustných, nebo druhý argument není zadán , nastavíme si příznak pro chybu. Na konci příznak pro chybu vyhodnotíme a vypíšeme příslušnou chybovou hlášku a jako návratovou hodnotu nastavíme 1, to znamená že nastala chyba a ukončíme funkci vyhodnoceni.

Dále testujeme ve funkci main návratovou hodnotu funkce vyhodnoceni, pokud nastala chyba, ukončíme program.

V poslední části programu zjišťujeme příznak prvního argumentu a podle něho voláme jednu z funkcí my_sqrt, my_asin, my_triangle nebo help.

V trojúhelníku navíc, zjistíme jestli je počet argumentů odpovídající syntaxi volání funkce pro trojúhelník. Pokud počet parametru neodpovídá, vypíšeme chybu a ukončíme program. Jako další funkci voláme my_length, která vrací délky stran a zároveň ošetřuje, zda jsou argumenty dobře zadány a jestli body tvoří trojúhelník. Pokud nastala nějaká chyba, funkce ji vypíše a vrátí chybovou hodnotu a ukončíme funkci. Pokud nenastala chyba, zavoláme funkci my_triangle s délkami stran trojúhelníku, která vrací vnitřní úhly v trojúhelníku.

Všechny výsledky vypisujeme ve formátu .10e.

5.3 Optimalizace

Funkci arcus sinus optimalizujeme pomocí vzorce (5) a (6), podle toho jestli je zadaná hodnota kladná nebo záporná. Optimalizaci provádíme ve dvou krocích. Na začátku zjišťujeme, jestli je vůbec optimalizace nutná. Pokud ano, nastavíme si příznak, podle kterého na konci zjistíme, která optimalizace nastala. Také si změníme hodnotu x, na hodnotu ze vzorce (5) nebo (6) s kterou budeme dále počítat. Po vypočítání zjistíme jestli byla funkce optimalizovaná a jestli ano, přejdeme k druhé části optimalizace, podle toho jestli byla optimalizovaná pro kladnou nebo zápornou hodnotu přepočítámevýsledek.

Tato optimalizace nám ušetří volání funkce rekurzně.

Pro výpočet funkce my_sqrt používáme relativní přesnost, proto musíme ošetřit nulový vstup. Relativní přesnost je totiž brána jako přesnost na několik platných číslic a nula není braná jako platná číslice a program vypisuje chybný výsledek.

Dále byly optimalizovány výpočty, které by byly v cyklu vypočítávány stále dokola.

6 Závěr

Problematiku vstupních dat se nám povedlo vyřešit a program byl ošetřen pro všechny nevyhovující vstupy, což nám umožňuje bezpečnější a jednodušší práci s výsledným programem.

Všechny použité algoritmy byly voleny a psány co nejefektivněji. Mezní hodnoty pro funkci arcus sinus se nám povedlo optimalizovat a díky těmto faktům byl celý program maximálně optimalizován. Na základě toho bylo dosaženo velmi uspokojivé rychlosti výpočtu. Časovou náročnost iteračních výpočtů jsme tedy potlačili.

Program byl úspěšně testován na platformách Linux a Microsoft Windows a všechny testované výsledky odpovídají formátem a přesností výpočtu výsledkům, které nám byly poskytnuty.

7 Reference

WEISSTEIN, Eric. Inverse Sine. MathWorld [online]. 1999 [cit. 2013-11-30]. Dostupné z: http://mathworld.wolfram.com/InverseSine.html

Příloha A

Metriky kódu

Počet funkcí: 7

Počet řádků zdrojového textu: 242

Velikost kódu programu: 12 913B

Velikost statických dat: 520B