Vzorová řešení pro automatickou kontrolu správnosti úkolů v kurzech programování

Pavel Pastorčák

Bakalářská práce
2006

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav aplikované informatiky
akademický rok: 2005/2006

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Pavel PASTORČÁK**
Studijní program:	**B 3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Téma práce:	**Vzorová řešení pro automatickou kontrolu správnosti úkolů v kurzech programování.**

Zásady pro vypracování:

1. Vytvořte sadu vzorových programů, které budou sloužit k testování správnosti funkce studentských programů v systému Moodle.
2. Vyberte z následujících: Buble sort, Násobení matic, Násobení matic přes ukazatele, Načtení souboru do dynamického pole, Porovnání metod násobení matic, Lineární seznam, Ukazatel na funkci, Obecný binární strom, Výmaz poznámek programu v jazyce C, Převod čísel mezi soustavami, Syntéza slovního vyjádření čísel, Hádání čísel, Analýza aritmetického výrazu.

Rozsah práce:

Rozsah příloh:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

Publikace:

1. Andrews, Mark Programujeme v jazyce C++, Computer Press, 1997
2. Chalupa, Radek 1001 tipů a triků pro Visual C++, Computer Press, 2003

Internetové stránky:

3. http://www.builder.cz

Vedoucí bakalářské práce:	**Ing. Tomáš Dulík** Ústav aplikované informatiky
Datum zadání bakalářské práce:	**14. února 2006**
Termín odevzdání bakalářské práce:	**16. června 2006**

Ve Zlíně dne 14. února 2006

prof. Ing. Vladimír Vašek, CSc.
pověřený děkan

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.
1

doc. Ing. Ivan Zelinka, Ph.D.
ředitel ústavu

ABSTRAKT

Snahou bylo vytvořit vzorová řešení úkolů do kurzu Programování v jazyce C. Práce navazuje na bakalářskou práci Automatické hodnocení úkolů v kurzech programování, která řeší samotnou opravu odevzdaných úkolů. Jsou zde popsána jednotlivá řešení vzorových úkolů a také soubory se vstupními daty, které generují.

ABSTRACT

The goal of this project is to create model solutions of student exercises for the course "Programming in C language". This work continues the bachelors thesis project "Automatic assessment of assignments in programming courses", which evaluates the uploaded exercises automatically. Every model solution is described here, also every file with input data, which generates the model solution.

PROHLÁŠENÍ

Prohlašuji, že jsem svou bakalářskou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu § 60 Zákona č.121/2000 Sb. , o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

Ve Zlíně dne 15. 6. 2006 …….

podpis

OBSAH

ÚVOD

Cílem práce je vypracovat vzorová řešení úkolů pro studenty kurzu Programování v jazyku C. Na Univerzitě Tomáše Bati ve Zlíně, Fakultě aplikované informatiky se k těmto účelům používá výukový systém Moodle (http://vyuka.ft.utb.cz). Jeho prostřednictvím vyučující poskytují studentům studijní materiály, vytvářejí diskusní fóra nebo zadávají úkoly, které studenti odevzdávají opět pomocí tohoto systému. Pro každou z těchto činností je v systému Moodle vytvořen samostatný modul. Jedním z nich je modul Program, který řeší automatickou opravu úkolů. Ta se provádí interpretací odevzdaného programu a porovnáním výsledků se vzorovým zadáním. Vzhledem k tomu, že modul Program měl i své nedostatky a počítal se stejným zadáním pro každého studenta, bylo mým úkolem kromě vytvoření vzorových řešení upravit modul tak, aby každý student dostal jiná vstupní data.

Každá vypracovaná úloha obsahuje dvě části. První z nich je učitelský program, který obsahuje vzorové řešení a zároveň generuje vstupní data pro druhou část. Tou je studentský program, který čerpá právě ze vstupních dat vzorového řešení. Učitelský program se uploaduje jenom při vytváření úkolů v systému Moodle a interpretuje se s každým odevzdaným studentským řešením. Díky tomu má každý student zadání rozdílné. Vzhledem k tomu, že všechna zadání v kurzu Programování v jazyku C nejsou koncipována tak, aby proběhly bez zásahu uživatele nebo byly závislé na vstupních datech, nebylo možné vytvořit, respektive bylo zbytečné vytvářet, vzorová řešení pro všechny úkoly.

I. TEORETICKÁ ČÁST

1 MOODLE

1.1 Základní charakteristika

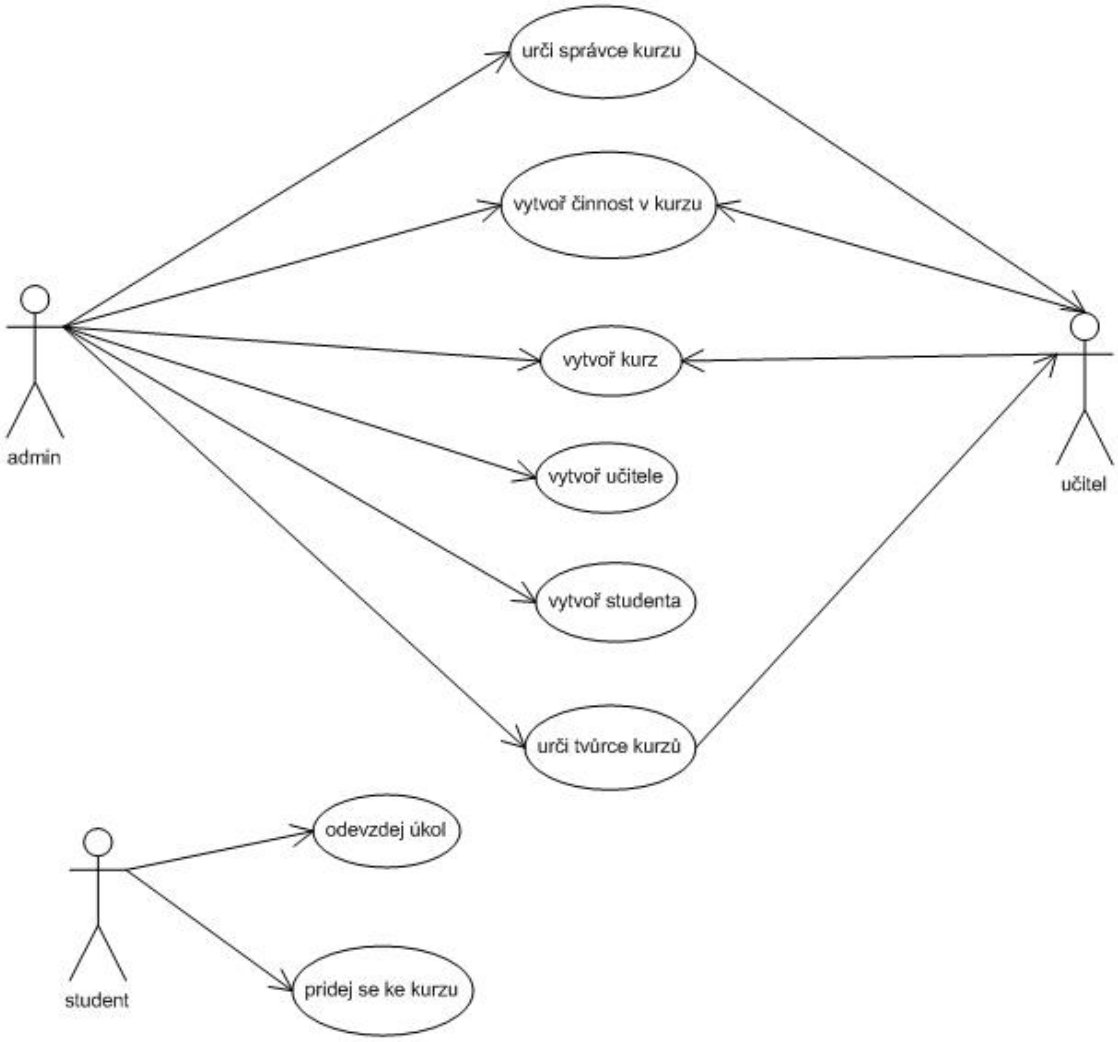
Obr. 1. Logo Moodle

Moodle je softwarový balík pro tvorbu výukových systémů a elektronických kurzů na internetu. Jedná se o neustále vyvíjející se projekt, navržený na základě sociálně konstruktivistického přístupu ke vzdělávání. Slovo Moodle bylo původně akronymem pro Modular Object-Oriented Dynamic Learning Environment (Modulární objektově orientované dynamické prostředí pro výuku); tato informace může být zajímavá především pro programátory a teoretické pedagogy. Lze ho také považovat za sloveso, které popisuje proces líného bloumání od jednoho k druhému, dělání věcí podle svého, hravost, která často vede k pochopení problému a podporuje tvořivost. V tomto smyslu se vztahuje jak k samotnému zrodu Moodlu, tak k přístupu studenta či učitele k výuce v on-line kurzech. Moodle je poskytován zdarma jako Open Source software (spadající pod obecnou veřejnou licenci GNU). První verze byla vydána 20. srpna v roce 2002. Byla určena pro intimnější prostředí menších tříd na vysokých školách a byla použita pro řadu studií, které podrobně zkoumaly spolupráci a reflexi v těchto malých skupinách dospělých účastníků. Od té doby se pravidelně objevují další verze obohacené o nové prvky a nabízející lepší škálovatelnost a vyšší výkon. Moodle nyní nepoužívají jen univerzity, ale i střední a základní školy, neziskové organizace, soukromé firmy, nezávislí učitelé a dokonce i rodiče, kteří se rozhodli vzdělávat své děti doma. Na zkvalitňování Moodlu se různým způsobem podílí čím dál víc lidí z celého světa. Důležitou součástí projektu Moodle je webová stránka moodle.org. Je zdrojem informací a místem pro diskusi a spolupráci uživatelů, mezi něž patří správci systémů, pedagogové, metodici, vědci, a samozřejmě vývojáři.

1.2 Struktura systému Moodle

Systém Moodle spravuje administrátor, který je určen během instalace. Administrátor může vytvářet kurzy, přiřazovat k nim učitele a studenty. Dále učitelé, kteří jsou stanoveni jako správci kurzu mohou vytvářet a editovat různé činnosti (moduly) např. úkoly, chat, hlaso-

vání, fórum, deník, test, studijní materiály, dotazníky, workshopy atd. Administrátor může také stanovit tvůrce kurzů. Ten je pak oprávněn vytvářet kurzy a určit k nim učitele. Učitel může pro každý kurz stanovit "klíč k zápisu", aby do něj měli přístup pouze oprávnění studenti. Tento klíč jim pak sdělí osobně, soukromým e-mailem apod. V případě potřeby mohou učitelé zapsat studenty do kurzu i ručně, nebo je ručně vyřadit.



Obr. 2. Use Case diagram

1.3 Modul Program

Modul Program byl vytvořen jako Bakalářská práce s názvem Automatické hodnocení úkolů v kurzech programování, autorem je Veronika Vašková. Důvod k vytvoření tohoto modulu byl prostý. Moduly, které obsahuje instalační balík, jsou vytvářeny pro obecné použití na všech školách a institucích a nemohou vyhovět individuálním požadavkům. Modul Program je téměř shodný s modulem Úkol. Rozdíl spočívá v automatické opravě odevzdaných prací, která je součástí modulu Program. Modul je vytvořen výhradně pro kurzy programování, ať už jde o jazyk C, C++ a jiné. Jak vyplývá ze samotného názvu Bakalářské práce, bylo mým úkolem vytvořit vzorová řešení úkolů do kurzu Programování v jazyce C. Z toho vyplývá, že jde o vzorová řešení pro modul Program. Při testování uploadování učitelských programů a následném odevzdávání studentských programů jsem odhalil jednu malou, ale podstatnou chybu. Při uploadu učitelského programu se program zároveň interpretuje, avšak při odevzdání studentského programu se učitelský program již neinterpretuje. Z toho vyplívá, že se vygeneruje vstup pro studentský program jenom jednou, tzn. všichni studenti budou mít zadání stejné. Toto řešení mi přišlo dost nešťastné a z toho důvodu jsem modul upravil tak, aby se učitelský program interpretoval s každým odevzdaným studentským programem.

2 VZOROVÁ ŘEŠENÍ

2.1 Struktura vzorových řešení

Jednotlivé programy jsou psány tak, aby odpovídaly zadáním úloh v kurzu Programování v jazyce C. Během testování byly odstraněny všechny chyby. Každé vzorové řešení nejprve generuje vstupní data odpovídající danému zadání a uloží je do souboru s názvem „vstup.txt“. Dál už se chová stejně jako program studentský. Vzhledem k tomu, že všechna zadání nejsou koncipována tak, aby bylo možné vycházet z určitého vstupního souboru, nebylo potřeba vzorová řešení pro všechny. Například úloha Úvod do pointrů. Tato úloha se snaží nastínit rozdíl mezi indexací polí pomocí indexů a pomocí pointrů. Výstupem je zde porovnání časů při indexaci rozměrných polí. Další úlohy podobného charakteru jsou Porovnání metod násobení matic, Hádání čísel.

2.1.1 Realizovaná zadání

Vzorová řešení jsou napsána pro následující zadání:

Řazení prvků pole – bubble sort, Násobení matic, Násobení matic přes ukazatele, Načtení souboru do dynamického pole, Lineární seznam, Ukazatel na funkci, Obecný binární strom, Výmaz poznámek v programu v jazyce C, Syntéza slovního vyjádření čísel, Analýza aritmetického výrazu.

Součástí každého vzorového řešení je studentský program.

2.1.2 Studentské programy

Studentské programy pracují podle následujícího principu. Načtou vstupní data ze souboru, jehož název je součástí příkazového řádku. Po zpracování dat podle určitých algoritmů se výstup vytiskne na monitor. Modul Program pracuje tak, že vše co se tiskne na monitor se ukládá do souboru. Tyto soubory po interpretaci porovná pomocí příkazu „diff“ a díky tomu zjistí, zda jsou výstupy shodné či nikoliv.

2.1.3 Struktura souboru se vstupními daty

Vstupní data uložená v souboru „vstup.txt“ by měla být takového charakteru, aby otestovala všechny možné mezní stavy. A to z toho důvodu, aby učitel měl jistotu, že studenti při

vývoji algoritmu počítali se všemi možnými stavy, které mohou na vstupu nastat a podle nich svůj program ošetřili. Zároveň je to vhodná možnost jak zjistit, jestli studenti své programy tvořili poctivě a snažili se přemýšlet o nejlepším řešení.

II. PRAKTICKÁ ČÁST

3 ÚPRAVA MODULU PROGRAM

Modul Program, který slouží k automatické opravě úkolů, měl jeden dost podstatný nedostatek. Vzorové řešení interpretoval jenom jednou. Mým úkolem bylo dosáhnout jiného zadání pro každého studenta. Z toho důvodu jsem byl nucen modul Program doplnit o jednu funkci.

3.1 Funkce teacher_debug

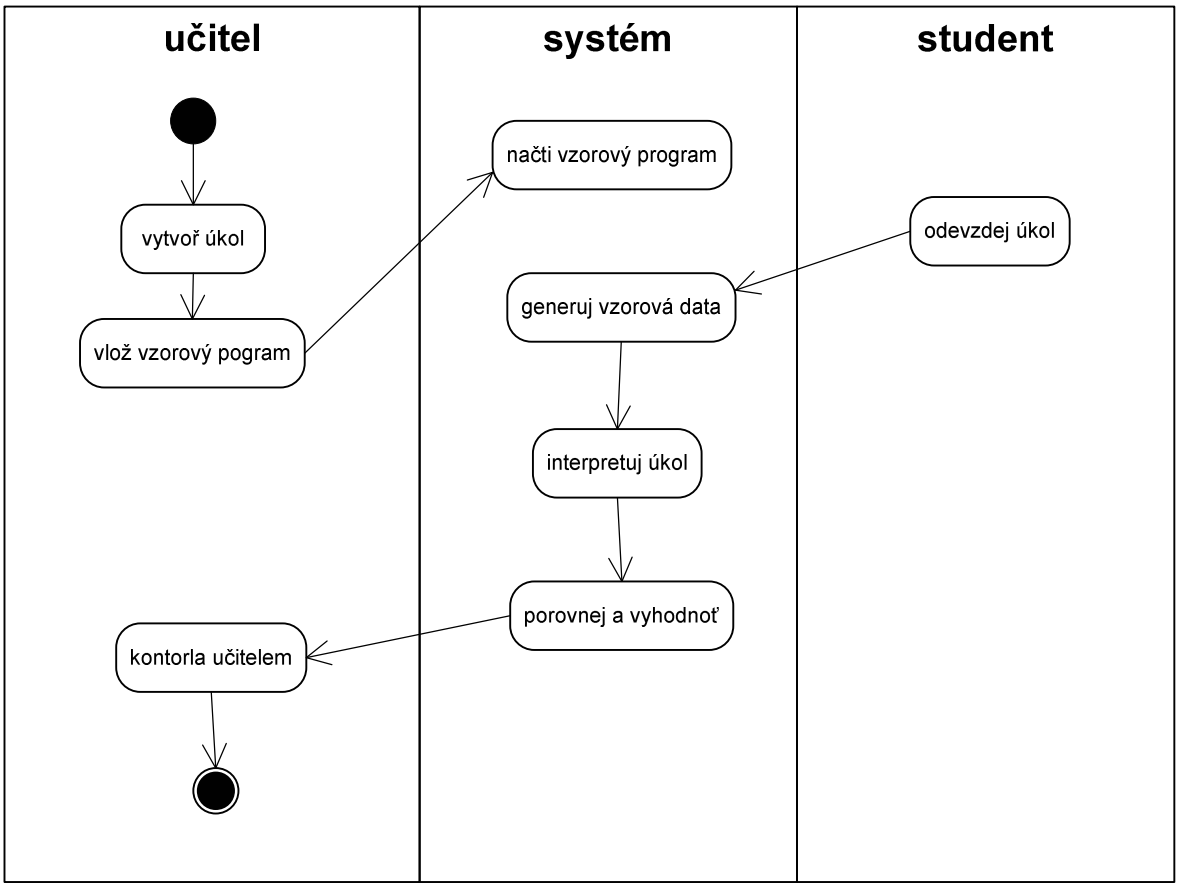
Funkce teacher_debug se nachází v souboru lib.php v kořenovém adresáři modulu. Obsahuje mimo jiné dvě velmi důležité funkce, a to upload a interpretaci vzorového i studentského programu. Funkce pro interpretaci vzorového programu se volá jenom při uploadu vzorového programu, stejně tak se volá funkce pro interpretaci studentského programu jenom při uploadu studentského programu. Funkce teacher_debug obsahuje kód pro interpretaci vzorového zadání, volá se na začátku funkce pro interpretaci studentského programu. Díky tomu se při každém odevzdání vygenerují jiná vstupní data a zároveň se uloží výstupní data vzorového zadání.

Tato funkce mnohem zvýšila praktičnost modulu a téměř eliminovala opisování, či jinou možnost podvádění, při odevzdávání úkolů.

4 REALIZACE VZOROVÝCH ŘEŠENÍ

4.1 Životní cyklus úkolu

Diagram zobrazený na Obr. 3. znázorňuje životní cyklus úkolu. Vyčleňuje roli učitele, studenta a systému od vytvoření úkolu do jeho konce.



Obr. 3. Životní cyklus úkolu - diagram

4.2 Řazení prvků pole – bubble sort

Napište program, který seřadí prvky pole vzestupně, podle velikosti, metodou "bubble sort" - tzn. pole se prochází NxN krát (N je počet prvků pole), v každém průchodu se porovnávají 2 sousední prvky, pokud nejsou ve správném pořadí tak se prohodí. Nezapomeňte soubor vybavit před odevzdáním hlavičkou!

Pole se v C deklaruje podobně jako v Pascalu:

double pole[30]; /* deklarace pole 30 ti prvků double */

Pole můžete naplnit (dle vlastního uvážení) hodnotami od uživatele nebo náhodnými čísly (funkce rand a srand ze stdlib.h), nebo jej inicializujte staticky, např.:

int pole[]={5,4,3,2,1}; /* deklarace inicializovaného pole. Pokud nechcete, nemusíte v hranatých závorkách uvádět jeho velikost, C si ji nastaví samo */

Pozn.: počet prvků takto nainicializovaného pole zjistíte jako podíl sizeof(pole)/sizeof(int)

4.2.1 Struktura vstupního souboru bubble sort

Vstupní soubor generovaný učitelským programem k úloze Řazení prvků pole – bubble sort obsahuje posloupnost náhodných čísel. Čísla jsou kladná i záporná, seřazena v různém pořadí.

Názorná ukázka:

```
42 -18 -33 -37 168 46 -23 -5 61 90 75 162 5 16 164 920 -24 120 87 10 -10
-13 126 35 127 77 -30 126 -5 -33
```

4.2.2 Postup při řešení úlohy bubble sort

Při spuštění vzorové úlohy se nejprve volá funkce generuj, která vytvoří soubor se vstupními daty. Další příkazy hlavní funkce main() už řeší samotný problém řazení pole. Nejdříve se však otevře soubor s vygenerovanými vstupními daty a ty se načtou do pole. Pole se následně seřadí již zmiňovanou metodou bubble sort. Výsledné pole uspořádané od nejmenšího k největšímu se vytiskne na obrazovku.

4.3 Násobení matic

Napište program pro násobení dvou matic. Teorie:

Součin matic je dán vztahem:

```
C[i,j] = SUMA(A[i,k] x B[k,j]),
     pro k = 1,2, ..... , colsA_rowsB
         i = 1,2, ..... , rowsA
         j = 1,2, ..... , colsB

 kde matice A  má rowsA řádků, colsA_rowsB sloupců
     matice B  má colsA_rowsB řádků, colsB sloupců
     matice C  má rowsA řádků, colsB sloupců
```

Jádro algoritmu je velmi jednoduché:

```
for (i=0; i<=(rowsA-1);i++)       /* opakuj pro všechny řádky v A */
      for (j=0; j<=(colsB-1);j++)   /* opakuj pro všechny sloupce v B */
          for (k=0; k<=(colsA_rowsB-1); k++)
C[i][j]=C[i][j]+(A[i][k]*B[k][j]);
                                  /* pronásob vektor i-tého řádku matice A */
                                  /* s vektorem j-tého sloupce matice B*/
```

Uživatel zadá programu rozměry matic *rowsA*, *colsA_rowsB*, *rowsB* a následně zadá hodnoty všech prvků vstupních matic. Vstup všech hodnot může být buď z klávesnice nebo ze souboru.

Rozměry matic mohou být pro jednoduchost omezeny (např. 10x10), takže všechny matice můžete nadeklarovat jako statická 2D pole s velikostí, kterou určíte jako maximální.

Nápověda: Nezapomeňte před výpočtem vynulovat matici C!

4.3.1 Struktura vstupního souboru úlohy Násobení matic

Vstupní soubor generovaný učitelským programem k úloze Násobení matic obsahuje dvě matice s náhodnými čísly. Matice mají pět řádků a pět sloupců.

Názorná ukázka:

```
198 178 37 172 34
150 106 144 130 206
20 38 176 78 119
118 181 8 137 28
99 27 152 163 143

44 193 156 48 116
79 100 168 45 16
3 163 10 201 184
181 142 208 191 74
102 49 99 85 117
```

4.3.2 Postup při řešení úlohy Násobení matic

Při spuštění vzorové úlohy se nejprve volá funkce generuj, která vytvoří soubor se vstupními daty. Další příkazy hlavní funkce main() už řeší samotný problém násobení matic. Nejdříve se však otevře soubor s vygenerovanými vstupními daty a ty se načtou do dvou 2D polí. V prvním poli je uložena matice A, ve druhém matice B. Tyto matice se mezi sebou vynásobí podle jádra algoritmu uvedeného v zadání. Výsledná matice C se vytiskne na obrazovku.

4.4 Násobení matic přes ukazatele

4.4.1 Motivační úvod

Operace násobení matic je jedna z nejčastěji používaných operací v grafických algoritmech popř. při číslicovém zpracování obrázků. Například algoritmy filtrů pro zaostřování nebo rozmlžování obrázku jsou založeny buď na 2D Fourierově transformaci, nebo právě na násobení matic - násobí matice obrázku s maticí filtru, a to opakovaně pro každý bod obrázku.

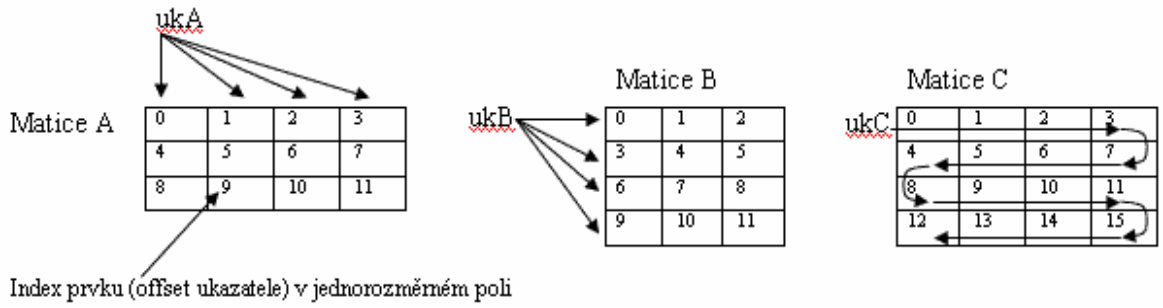
V grafických aplikacích nebo hrách, kde je potřeba generovat obrázky s rozlišením např. 1024x768=800tis. bodů, a to vše třeba ještě 25x za sekundu, je pak čas, který program stráví operací násobení matice, primárním faktorem ovlivňujícím výkon celé aplikace.

4.4.2 Zadání

Upravte algoritmus z úkolu "Násobení matic" tak, aby nepoužíval v cyklech maticových operací indexaci pole. Přístup do pole pomocí indexů je značně neefektivní, např. k přístupu na 1 prvek pole typu p[i][j] musí procesor provést 2 násobení, 2 sčítání a 2 odkazy přes pointer. Proto všechny přístupy do pole uvnitř cyklů nahraďte přístupem přes pointery s tím, že při adresování používejte pouze přičítání konstant k ukazateli, čímž eliminujete potřebu operace celočíselného násobení při indexaci.

Pro maximální efektivitu implementujte matice pomocí jednorozměrného pole.

V polích matic A, B, C se budete pohybovat pomocí ukazatelů takovýmto způsobem:



Obr. 4. Násobení matic přes ukazatele

ukA v poli Matice A se bude pohybovat po řádku, tzn. pomocí operace ukA++.

ukB v poli Matice B se musí pohybovat po prvcích v jednom sloupci, čehož dosáhnete pomocí operace ukB=ukB+3 (obecněji: ukB=ukB+colsB).

Na začátku každého cyklu přitom musíte nastavit ukA na začátek správného řádku a ukB na začátek správného sloupce. To můžete udělat třeba tak, že si budete v proměnných ukZacRadkuA a ukZacSloupceB uchovávat ukazatele s těmi správnými hodnotami.

ukC se může pohybovat od začátku až do konce algoritmu pouze pomocí operace ukC++, která jej nastaví vždy na další prvek výsledku.

Při návrhu algoritmu je nezbytně nutné vědět, jak se vůbec matice násobí - čili vaším nejlepším přítelem bude tužka a papír, který vám pomůže zjistit, jak nastavovat ukazatele, aby v každém cyklu byly tam, kde je potřebujete mít.

4.4.3 Struktura vstupního souboru úlohy Násobení matic přes ukazatele

Vstupní soubor generovaný učitelským programem k úloze Násobení matic přes ukazatele obsahuje dvě matice s náhodnými čísly. Matice mají pět řádků a pět sloupců.

Názorná ukázka:

```
198 178 37 172 34
150 106 144 130 206
20 38 176 78 119
118 181 8 137 28
99 27 152 163 143

44 193 156 48 116
79 100 168 45 16
3 163 10 201 184
181 142 208 191 74
102 49 99 85 117
```

4.4.4 Postup při řešení úlohy Násobení matic přes ukazatele

Při spuštění vzorové úlohy se nejprve volá funkce generuj, která vytvoří soubor se vstupními daty. Další příkazy hlavní funkce main() už řeší samotný problém násobení matic. Nejdříve se však otevře soubor s vygenerovanými vstupními daty a ty se načtou do dvou 2D polí. V prvním poli je uložena matice A, ve druhém matice B. Tyto matice se mezi sebou vynásobí podle stejného algoritmu jako u předchozího zadání. Na rozdíl od předchozího zadání se pole neadresuje pomocí indexů, ale pomocí pointrů. Stejně tak se pole indexovaly při nulování, či načítání hodnot. Výsledná matice C se vytiskne na obrazovku.

4.5 Načtení souboru do dynamického pole

Vytvořte funkci:

char* load(char * filename, unsigned long *ukSize),

která načte libovolně velký soubor se jménem "filename" do pole, které alokuje v dynamické paměti a vrátí na toto pole ukazatel. Velikost alokovaného pole ještě navíc uloží do proměnné, na kterou ukazuje ukSize. Pokud při otvírání souboru nebo alokaci paměti dojde k chybě, vrátí funkce NULL.

Funkci otestujte v programu, který načte do paměti počítače zadaný soubor a vypíše jeho obsah na obrazovku. Jméno souboru bude určeno prvním parametrem programu na příkazové řádce operačního systému.

4.5.1 Struktura vstupního souboru úlohy Načtení souboru do dynamického pole

Vstupní soubor generovaný učitelským programem k úloze Načtení souboru do dynamického pole obsahuje náhodný počet náhodných znaků. Za znaky považuji malé písmena abecedy.

Názorná ukázka:

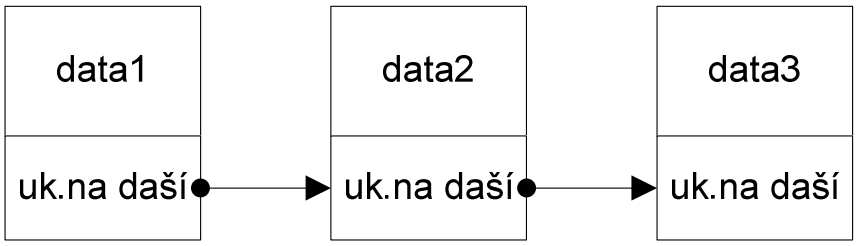
```
bbfjdynwrlkpiwubattavebaymoatwwebkfxjzfthjxrjlaimsdnbbfjdynwrlk-
piwubattavebaymoatwwebkfxjzfthjxrjlaimsdnbbfjdynwrlkpiwubattave-
baymoatwwebkfxjzfthjxrjlaimsdnbbfjdynwrlkpiwubattavebaymo-
atwwebkfxjzfthjxrjlaimsdn
```

4.5.2 Postup při řešení úlohy Načtení souboru do dynamického pole

Při spuštění vzorové úlohy se nejprve volá funkce generuj, která vytvoří soubor se vstupními daty. Další příkazy hlavní funkce main() už řeší samotný problém načtení souboru do dynamického pole. Nejdříve se volá funkce load, která otevře soubor se vstupními daty (v tomto případě jde o náhradu běžného souboru). Zjistí se velikost souboru a potřebný prostor paměti se naalokuje. Obsah souboru se uloží do pole, které je zároveň vratnou hodnotou funkce load. Program se ukončí výpisem pole na obrazovku.

4.6 Lineární seznam

Lineární seznam slouží jako struktura dynamických proměnných. Každá buňka seznamu se skládá z uložených dat a ukazatele na další buňku.



Obr. 5. Struktura lineárního seznamu

Implementujte kompletní sadu funkcí pro práci s lineárním seznamem. Do kostry programu uvedené níže doplňte kód všem funkcím, jejichž implementace schází.

V programu jsou ukázkově implementovány funkce insertFirst a deleteFirst, což si můžete ověřit při krokování funkce main(), kde se demonstruje použití funkcí pro práci se seznamem.

4.6.1 Struktura vstupního souboru úlohy Lineární seznam

Vstupní soubor generovaný učitelským programem k úloze Lineární seznam obsahuje náhodně vygenerovaný seznam jmen lidí, jejich váhy, výšky a věku. Tyto hodnoty se vybírají z pole všech hodnot, které jsou uloženy v kódu vzorového řešení jako globální proměnná.

Názorná ukázka:

```
167 57.0 19.0 Vladislava Malinovska
10 0.1 0.1 Jan Vorisek
170 60.0 20.0 Franta Maly
170 50.0 35.0 Pepa Dlouhy
183 82.0 25.0 Jenda Kolik
```

4.6.2 Postup při řešení úlohy Lineární seznam

Při spuštění vzorové úlohy se nejprve volá funkce generuj, která vytvoří soubor se vstupními daty. Další příkazy hlavní funkce main() už řeší samotný problém lineárního seznamu. Nejdříve se volají funkce pro vkládání dat do lineárního seznamu, ať už insertFirst (vloží nový prvek na začátek seznamu), insertLast (vloží nový prvek na konec seznamu) a insert (vloží nový prvek za prvek, na který ukazuje ukazatel). Další na řadě je test funkčnosti funkce search, která vyhledává zadaný prvek v seznamu (vrací ukazatel na hledaný prvek nebo NULL). A nakonec se testují funkce k mazání seznamu. Mezi ně patří deleteFirst (vymaže první prvek v seznamu), deleteLast (vymaže poslední prvek seznamu) a deleteAny (vymaže libovolný prvek seznamu – určí se ukazatelem).

4.7 Ukazatel na funkci

Typický příklad použití ukazatele na funkci můžete nalézt ve standardní knihovně stdlib.h a její funkci qsort, která umí setřídit pole libovolného typu podle libovolného kritéria, a to neefektivnějším možným algoritmem Quick Sort, jehož časová složitost je pouze Nxlog(N).

Funkce qsort má jako poslední parametr ukazatel na uživatelskou funkci, která pro ni porovnává jednotlivé prvky řazeného pole.

Upravte níže uvedený příklad tak, aby v případě shody názvu byly stejnojmenné nápoje seřazeny podle obsahu alkoholu, v případě shody obsahu alkoholu podle výrobce a v případě shody výrobce ještě podle objemu balení.

```
# include <stdlib.h>
# include <stdio.h>

typedef struct {
  char * nazev;
  float promile;
  char * vyrobce;
  float objemBaleni;
} tNapoj;

int porovnejNapoje(const void *n1, const void *n2) {
  return strcmp(((tNapoj*)n1)->nazev, ((tNapoj*)n2)->nazev);
}

tNapoj poleNapoju[]={{"Pivo", 3.5}, {"Vino", 6.5}, {"Lih", 45} /* zde
vložte další položky a stávající rozšiřte, aby všechny prvky struktury
byly inicializovány!*/ };

int main() {
  int pocetPrvku=sizeof(poleNapoju)/sizeof(tNapoj);

  qsort(poleNapoju, pocetPrvku, sizeof(tNapoj), &porovnejNapoje);

/* zde vložte kód pro tisk jednotlivých položek pole */
}
```

4.7.1 Struktura vstupního souboru úlohy Ukazatel na funkci

Vstupní soubor generovaný učitelským programem k úloze Ukazatel na funkci obsahuje náhodně vygenerovaný seznam nápojů, výrobců, jejich objemu a počtu promile. Tyto hodnoty se vybírají z pole všech hodnot, které jsou uloženy v kódu vzorového řešení jako globální proměnná.

Názorná ukázka:

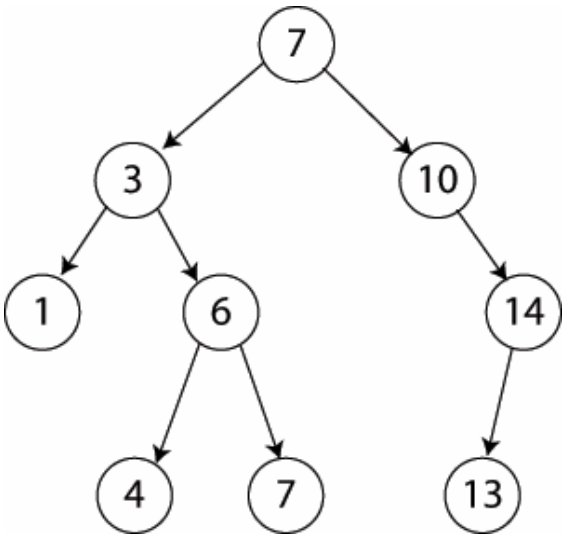
```
0.7 40.0 Irsko Black_&_White
0.5 40.0 Bozkov Rum
0.5 3.5 Branik Pivo
0.5 37.0 Becherovka Becherovka
0.5 45.0 Seliko Lih
```

4.7.2 Postup při řešení úlohy Ukazatel na funkci

Při spuštění vzorové úlohy se nejprve volá funkce generuj, která vytvoří soubor se vstupními daty. Další příkazy hlavní funkce main() už řeší samotný problém úkolu ukazatel na funkci. Nejdříve se zjistí počet prvků načtených ze souboru. Poté se volá samotná funkce qsort, které se předávají následující parametry: pole nápojů, počet prvků pole, kolik paměti zabírá jeden prvek pole a ukazatel na funkci porovnejNapoje, podle které qsort nápoje seřadí. Nakonec se seřazené pole nápojů vytiskne na obrazovku.

4.8 Obecný binární strom

Strom můžeme chápat jako zobecněný seznam, ve kterém může mít každý prvek více následníků. První prvek této struktury se nazývá kořen stromu, prvky bez následníků se označují jako listy. Uzlům, které nejsou listy, obvykle říkáme vnitřní uzly stromu.



Obr. 6. Příklad obecného binárního stromu

Vytvořte knihovnu pro práci s obecným binárním stromem (tj. stromem, který umí pracovat s daty libovolného typu). Knihovna bude implementována jako soubor

STROM.H s deklaracemi všech dat. typů a funkcí a soubor STROM.C s implementací všech funkcí.

Dále vytvořte testovací program "hlavni.c", pomocí kterého ověříte všechny funkce vaší knihovny - můžete použít kostru uvedenou níže. Místo datového typu "char *" ale použijte datový typ tNapoj, který je definován v úloze Ukazatel na funkci. Z této úlohy použijte také funkci pro porovnání 2 prvků typu tNapoj.

4.8.1 Struktura vstupního souboru úlohy Obecný binární strom

Vstupní soubor generovaný učitelským programem k úloze Obecný binární strom obsahuje náhodně vygenerovaný seznam nápojů, výrobců, jejich objemu a počtu promile. Tyto hodnoty se vybírají z pole všech hodnot, které jsou uloženy v kódu vzorového řešení jako globální proměnná.

Názorná ukázka:

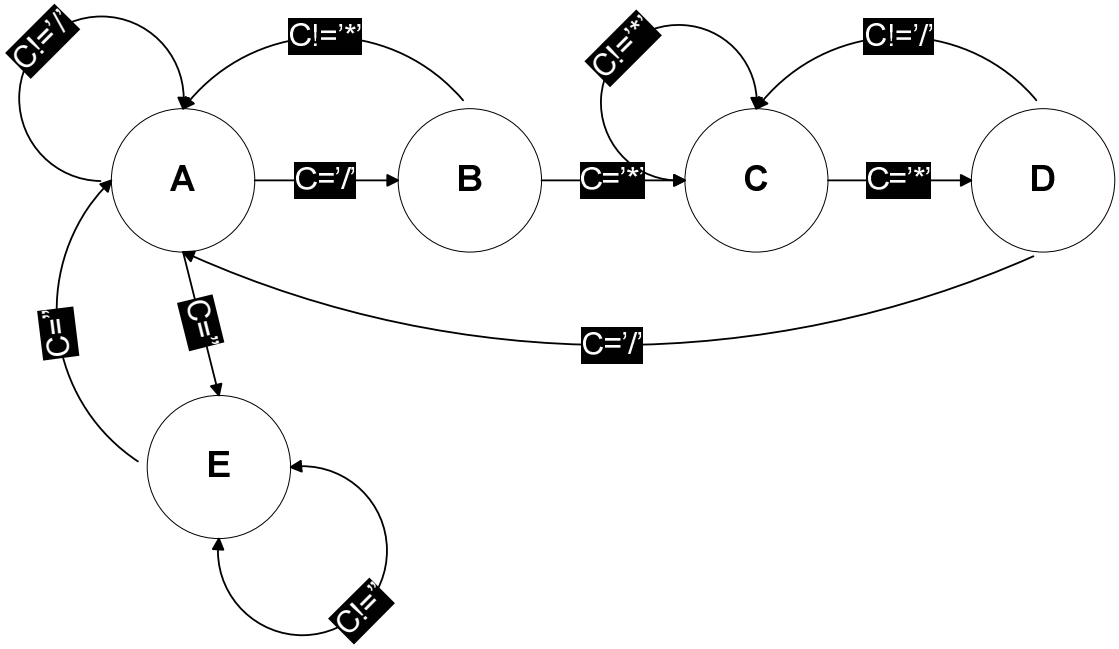
```
0.5 45.0 R.Jelinek PELISKOVKA
0.5 40.0 Finsko Finlandia
0.7 6.5 Valtice Ryzlink_vlassky
0.8 6.2 Valtice Veltlinske_zelene
0.7 40.0 Irsko Black_&_White
```

4.8.2 Postup při řešení úlohy Obecný binární strom

Při spuštění vzorové úlohy se nejprve volá funkce generuj, která vytvoří soubor se vstupními daty. Další příkazy hlavní funkce main() už řeší samotný problém obecného binárního stromu. Nejdříve se zjistí počet prvků načtených ze souboru. Poté se prvky vloží do stromu. K tomu slouží funkce insert. Ta vkládá do levé větve stromu prvek, který je vyhodnocen jako menší (funkce porovnejNapoje použitá v úkolu ukazatel na funkci). Další na řadě je test funkčnosti funkcí inorder, preorder a postorder. Nakonec se testuje funkce smazání binárního stromu deleteTree.

4.9 Výmaz poznámek programu v jazyce C

Navrhněte konečný Mealyho automat, který ze vstupního souboru v jazyce ANSI C odstraní poznámky (tzn. vše mezi /* */, včetně ohraničení poznámky). Automat nesmí mazat poznámky v řetězcích - např. printf("/*Toto neni poznamka*/");



Obr. 7. Konečný automat

Automat implementujte standardním způsobem dle přednášky s tím, že výsledný program bude jako vstupní soubor používat buď:

- soubor zadaný jako 1. parametr na příkazové řádce
- vstup z klávesnice, pokud parametr není zadán

Výstup programu bude

- do souboru, který byl zadán jako 2. parametr na příkazové řádce
- na obrazovku, pokud 2. parametr není zadán

Pokud je na příkazové řádce pouze 1 parametr, bere se tento parametr jako jméno vstupního souboru. Abyste nemuseli mít ve vašem kódu 2 různé větve - jednu pro operace se standardním vstupem/výstupem, druhou pro operace se soubory - můžete s výhodou použít standardně definované handly na soubory standardního vstupu a výstupu, které knihovna stdio.h exportuje v proměnných "stdin" a "stdout" - viz příklad Výpočet průměrného prospěchu.

Součástí řešení bude dokumentace s obrázkem a popisem funkce vašeho konečného automatu, a to ve formátu DOC nebo RTF.

4.9.1 Struktura vstupního souboru úlohy Výmaz poznámek programu v jazyce C

Vstupní soubor generovaný učitelským programem k úloze Výmaz poznámek programu v jazyce C obsahuje náhodný počet náhodných znaků. Za znaky považuji malé písmena abecedy. Mezi ně jsou na náhodných pozicích umístěny znaky znázorňující začátek, respektive konec poznámek. Ze zištných důvodů je v „poznámkách“ začleněn vždy samostatně jeden ze znaků označující poznámku.

Názorná ukázka:

```
igay /*lfvlrrbpasoro
yuwzfzvtghhhsxylbljvjqxsrpiihxsadakrhnmyksbat**/
ad /*sahyekdvxevo
yqoeuckixlizchihajufvmrbmkrxagpnrlplodi/kg/kt*i/cfo*/
jxswhwblsmmwlsinchvh /*gpcluuxchpbm
oumelaoxpsblw*zt*g/u*v*ezsv**/
nksafwkdgnbt /*flvomapuwf
jjkeqqrnkwbvmou*xxnou*du*kpi/asv*w*xofyexbj/qbs*/
jbbjfmbbyavkjmgq /*scbujiudetlipgcykxb
qv*rrj/zylfoau*fcr*/
qxtfxnspryernmccmxfbxtqggxpfevcmklzikvaalgbxgxgylnrz /*kuvnod
sxvwffyoilqtyftzrpknb*/
pvwucit /*mbhmejughesp
bfxgjjlebhdcukqwtckg/eqrt*byai/g/dbt*ssh/cj/g/nsnv*kcg/bg/cd*/
yussbftqtlup /*lsuafixbdewxkukyzhld
hrjxjfspllxbizeoitrkdgolvswgcmv*u*k*/
dvcpriixljdcmkbtlkesd /*uxxkgav
onnctsduaolkhoidtti/g/i/pt*bg/oj/j/v*zzft*lh/qxyt*v*ch/canj/*/
wpwuoqpsnbngpgkof /*nbzhdvlurqvl
ccmrnpaxvbdzlkwsnsu*w*rbsi/t*w*cu*u*t*h/cg/qnv**/
xlxxikjjeeeog /*qouwywhbramrbldvmb
jxbexacivtielyonkklh/i/pszyw*cpxu*t**/
ehnuzoccprszxxhyiol /*dpfpuxoz
ctqtpldvylscdxpzllvdwpwv*w*ecaj/zrsxxu*h/lh/u*v*f*/
```

4.9.2 Postup při řešení úlohy Výmaz poznámek programu v jazyce C

Při spuštění vzorové úlohy se nejprve volá funkce generuj, která vytvoří soubor se vstupními daty. Další příkazy hlavní funkce main() už řeší samotný problém výmazu poznámek programu v jazyce C. Nejdříve se nastaví počáteční stav automatu do pozice A (viz. Obr. 7.). Poté se zjistí, zda se má vstup/výstup číst/zapisovat do souboru nebo na obrazovku. Následně se prochází soubor po znacích a hledá se začátek a konec poznámek. Pokud automat zjistí, že se jedná o poznámky, přestane znaky tisknout na výstup (soubor, obrazovka).

4.10 Syntéza slovního vyjádření čísel

Napište program, který převede celé číslo na jeho slovní vyjádření, tzn. např. pro zadané číslo 512121856340162 se vypíše "pět set dvanáct bilionů sto dvacet jedna miliard osm set padesát šest milionů třista čtyřicet tisíc (jedno) sto šedesát dva".

- Ošetřete správně české skloňování řádů stovek a tisíců(máme čtyři **sta**, ale pět **set**!)
- Ošetřete správě koncovky počtu řádů dle rodu daného řádu - máme "**jedno** sto" ale "**jeden** tisíc" a "**jedna** miliarda"

Program přečte číslo z klávesnice a výsledek vypíše na obrazovku. Nejdříve si musíte uvědomit pravidla, která určují tvorbu číslovek v češtině. Nad desítkovou soustavou má totiž čeština vybudovanou ještě nadstavbu bilionů, miliard, milionů, stovek a tisíců. Přitom musíte ošetřovat spoustu výjimek, které jako čech používáte, aniž byste o tom většinou věděli. Základní pravidla jsou:

- Číslo musíte zpracovávat od nejvyššího řádu - stejně jako jej "zpracováváte" v hlavě, když jej čtete.
- Všimněte si, že při převodu čísla pracujete s jednotlivými číslicemi po trojicích (biliony, miliardy, miliony, tisíce) - viz příklad 512 121 856 340 162 - a tyto trojice jsou na sobě zcela nezávislé (tzn. číslo v rámci jedné trojice neovlivňuje převod sousední trojice).
- v rámci každé trojice má výjimečné postavení 2. číslice a to díky výjimce u čísel 11-19, které se nečtou jako "deset jedna" (jako např. 21 = "dvacet jedna") - ale jedenáct až devatenáct. Toto musíte správně ošetřit, čili pokud na 2. pozici narazíte na číslici 1, musíte použít tabulku pro převod čísel 10-19 na slovní vyjádření.
- pokud chcete, můžete u čísel jako 1620 místo "jeden tisíc šest set dvacet" vypisovat pouze zkrácený tvar "tisíc šest set dvacet"

Použití příkazu goto je zakázáno!

4.10.1 Struktura vstupního souboru úlohy Syntéza slovního vyjádření čísel

Vstupní soubor generovaný učitelským programem k úloze Syntéza slovního vyjádření čísel obsahuje náhodně vygenerované číslo o proměnném počtu řádů.

4.10.2 Postup při řešení úlohy Syntéza slovního vyjádření čísel

Při spuštění vzorové úlohy se nejprve volá funkce generuj, která vytvoří soubor se vstupními daty. Další příkazy hlavní funkce main() už řeší samotný problém syntézy slovního vyjádření čísel. Nejdříve se spočítá počet řádů a načtené číslo se přeskládá od nejvyššího řádu k nejmenšímu. Dále se podle řádu volá funkce, která zadané číslo převede a vytiskne ve slovní podobě.

4.11 Analýza aritmetického výrazu

Napište program, který bude analyzovat aritmetické výrazy, oddělené středníkem, jejichž syntaxe je určena LL gramatikou s pravidly:

```
S   -> EX; S | e

EX  -> M EX2
EX2 -> + M EX2 |
       - M EX2 | e

M   -> T M2


M2  -> * T M2|
       / T M2| e


T-> + num | -num | (EX) | round(EX)
```

Pro správně zadané aritmetické výrazy - například "3.5E-3*5.0*(100+2.1)/(5+6);" program vypíše jejich hodnotu, pro chybně zadané výrazy vypíše "Syntax error". Výrazy zadávejte na klávesnici.

Funkce round(..) zaokrouhlí hodnotu floating point výrazu v závorkách na nejbližší celé číslo (zaokrouhlování musíte ošetřit ručně pomocí "if (x-floor(x)>0.5)...", ANSI C nemá něco jako funkci nearest nebo rint)

Postup: Pro analýzu (vyhledávání) terminálních symbolů - num (celočíselné konstanty), '+', '-', '/', '*', '(', ')') napište funkci lexAnalyzer(..), která bude obsahovat konečný automat. Protože konečný automat "ztrácí" poslední zpracovaný znak, tak aby nemusely být všechny symboly na vstupu odděleny min. jednou mezerou, můžete využít v implementaci konečného automatu funkci ungetc(..) ze stdio.h, která do bufferu vstupního souboru vrací zadaný znak.

Popis toho, jak se dělá přepis pravidel výše uvedené gramatiky na algoritmus rekurzivního sestupu máte v přednášce.

Při řešení můžete vyjít z hotového a funkčního programu pro vyhodnocování jednoduchých výrazů s celočíselnými konstantami (již přeložený si jej můžete stáhnout a vyzkoušet zde - při zadávání výrazů je nezapomeňte ukončovat středníkem, jinak vám program nic nespočítá). Pokud se vydáte touto cestou, musíte program rozšířit takto:

- V konečném automatu budete vyhodnocovat floating point konstanty - doporučuji ulehčit si život použitím funkce strtod(...) ze stdlib.h
- V konečném automatu budete rozlišovat ještě identifikátor "round"
- Funkci t() musíte rozšířit o vyhodnocování round(EX)
- Ve všech pravidlech musíte upravit parametry funkce check(..) tak, aby analyzér přijímal funkci round(..)
- U operace dělení ošetřete dělení nulou
- Program můžete ještě vylepšit tak, že v případě syntaktických chyb bude vypisovat názvy očekávaných symbolů - na to je tam předchystané pole symbols[] a funkce index(..) a pozici na řádku, kde k chybě došlo.

4.11.1 Struktura vstupního souboru úlohy Analýza aritmetického výrazu

Vstupní soubor generovaný učitelským programem k úloze analýza aritmetického výrazu obsahuje náhodně vygenerovanou rovnici ukončenou středníkem. Rovnice se skládá z celých čísel ,aritmetických znamínek (+, -, *, /) a závorek. Četnost výskytu jednotlivých znamínek je náhodná.

Názorná ukázka:

```
33*23-178-(169/107*199)+45*127;
```

4.11.2 Postup při řešení úlohy Analýza aritmetického výrazu

Při spuštění vzorové úlohy se nejprve volá funkce generuj, která vytvoří soubor se vstupními daty. Další příkazy hlavní funkce main() už řeší samotný problém analýzy aritmetického výrazu. Nejdříve se ukazatel na vstupní soubor uloží do globální proměnné. Poté se volá funkce lexAnalyzer(), v jejíchž útrobách se skrývá Mealyho konečný automat. Tato

funkce jako první odstraní všechny mezery, vyhodnotí první znak výrazu a vrací jeho hodnotu. Po lexAnalyzer() následuje funkce s() – startovací pravidlo gramatiky, která prochází matematický výraz rekurzivním voláním podle zadané LL gramatiky. Návratová hodnota této funkce je výsledek aritmetického výrazu.

ZÁVĚR

Cílem práce bylo vytvořit sadu vzorových řešení pro automatickou opravu úkolů v kurzu programování v jazyce C. Vytvořeny byly všechny programy, které se dají interpretovat bez přímého zásahu uživatele.

Tato práce přináší do kurzů Programování v jazyce C a Základy algoritmizace následující vylepšení:

- Testy funkčnosti odevzdaných příkladů jsou mnohem rozsáhlejší, než dosavadní manuální testování, které provádí učitel ve svém omezeném čase. Nově je testována také správná alokace/dealokace paměti, vstupní testovací vektory obsahují jak vybrané sekvence („chytáky“), tak náhodně generovaná data, která mohou odhalit chyby, na které učitel při tvorbě vzorového programu nepomyslel.
- Studenti získávají při odevzdání okamžitou zpětnou vazbu na kvalitu své práce, tj. mají možnost chyby v programech odstranit v době, kdy mají funkci svého kódu ještě v čerstvé paměti.

Struktura všech generovaných vstupních souborů byla navrhnuta tak, aby bylo možné co nejlépe otestovat funkčnost studentských programů. Samotná vzorová řešení jsou ošetřena proti různým krizovým stavům, nemělo by tedy při odevzdávání docházet k jakýmkoliv chybám.

Náměty na vylepšení:

- Jednou z možností je ještě propracovanější struktura vstupních dat, pomocí které by bylo možné studentské programy testovat ještě efektivněji a více do hloubky.
- Další z možností jsou větší nároky na bezpečnost. Je totiž možné, že se po spuštění této práce do provozu objeví chyby, které nebyly odhaleny při testování.

SEZNAM POUŽITÉ LITERATURY

Publikace:

[1] Andrews, Mark: *Programujeme v jazyce C++,* Computer Press, 1997

[2] Chalupa, Radek: *1001 tipů a triků pro Visual C++,* Computer Press, 2003

[3] Vašková Veronika: *Automatické hodnocení úkolů v kurzech programování,* Bakalářská práce UTB, Zlín 2005

Internetové stránky:

[4] Učíme se jazyk C: http://www.builder.cz/serial3.html

[5] Reference jazyka C: http://www.cplusplus.com/ref

[6] Moodle: http://moodle.org

SEZNAM OBRÁZKŮ