TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta pedagogická

Studijní program: Učitelství pro střední školy
Katedra: Katedra aplikované matematiky
Kombinace: Matematika – informatika

Software pro parametrizované úlohy

Software for parametric exams

Diplomová práce: 05–FP–KAP–01

Autor: Michal Stehlík
Adresa: Jáchymovská 264, Liberec, 460 10
E-mail: michal.stehlik@centrum.cz

Podpis: .....

Vedoucí diplomové práce: RNDr. Martina Šimůnková, PhD., KAP TU v Liberci
Konzultant: Doc. RNDr. Miroslav Brzezina, CSc., KAP TU v Liberci

Počet	stran	slov	obrázků	tabulek	pramenů	příloh
	60	13430	7	3	21	2

Zpracování a sazba: LaTeX 2ε

V Liberci dne: 15. května 2005

TU v Liberci, FAKULTA PEDAGOGICKÁ
461 17 Liberec 1, Hálkova 6

---

katedra: Katedra aplikované matematiky

Zadání diplomové práce

(Pro magisterský studijní program)

pro (diplomant): Michal Stehlík

adresa: Jáchymovská 264, Liberec 10, 46010

obor: Učitelství pro střední školy, MA–IF

název DP: Software pro parametrizované úlohy

název DP v angličtině: Software for parametric exams

vedoucí DP: RNDr. Martina Šimůnková, PhD., KAP TU v Liberci

konzultant: Doc. RNDr. Miroslav Brzezina, CSc., KAP TU v Liberci

termín odevzdání: květen 2005

....
děkan vedoucí katedry

Převzal (diplomant): Michal Stehlík

Datum: .....

Podpis: .....

Název DP:
Software pro parametrizované úlohy

Vedoucí práce:
RNDr. Martina Šimůnková, PhD.

Úvod:
Téma bylo vypsáno na katedře aplikované matematiky FP TU v Liberci. Na této katedře vznikl pod vedením doc. Brzeziny před několika lety on-line systém pro vytváření písemných parametrizovaných prací. V roce 2002 jej rozšířili studenti FP TUL v rámci grantu FRV (s řešitelkou dr. Šimůnkovou) o příklady z matematiky pro základní školy.

Cíl:
Cílem diplomové práce je naprogramování off-line aplikace pro jednoduché a pro uživatele příjemné vytváření nových příkladů parametrizovaných úloh a sestavování a tisk celých sestav příkladů pro použití jako kontrolní testy v matematice na základních a středních školách.

Požadavky:
Znalost typografického systému TEX a zvládnutí programování ve vhodném vývojovém prostředí.

Literatura:

- Cantù, M.: *Myslíme v jazyku Delphi 6 (1. díl)*. Grada Publishing, Praha, 2002.
- Cantù, M.: *Myslíme v jazyku Delphi 6 (2. díl)*. Grada Publishing, Praha, 2002.
- Svoboda, L. a kol.: *1001 tipů a triků pro Delphi*. Computer Press, Brno, 2001.
- Rybička, J.: *LATEX pro začátečníky*. Konvoj, Brno, 1999.

Prohlášení o původnosti práce

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a že jsem uvedl veškerou použitou literaturu.

V Liberci dne: 15. května 2005 Michal Stehlík

. . . . . .

Prohlášení k využívání výsledků DP

Byl jsem se seznámen s tím, že na mou práci se plně vztahuje zákon č.121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že Technická univerzita v Liberci (TUL) má právo na uzavření licenční smlouvy o užití mé diplomové práce a prohlašuji, že souhlasím s případným užitím mé diplomové práce (prodej, zapůjčení, kopírování, apod.).

Jsem si vědom toho, že: užít své diplomové práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše). Diplomová práce je majetkem školy, s diplomovou prací nelze bez svolení školy disponovat.

Beru na vědomí, že po pěti letech si mohu diplomovou práci vyžádat v Univerzitní knihovně Technické univerzity v Liberci, kde bude uložena.

Autor: **Podpis:**

Michal Stehlík . . . . . .

Adresa: **Datum:**
Jáchymovská 264, Liberec, 460 10 15. května 2005

O registrovaných známkách

Všechny registrované nebo jiné obchodní známky použité v této práci jsou majetkem jejich vlastníků. Uvedením nejsou zpochybněna z toho vyplývající vlastnická práva.

Poděkování

Děkuji všem, bez nichž bych svou práci asi nikdy nedokončil. Děkuji své vedoucí diplomové práce RNDr. Martině Šimůnkové, PhD. za konzultace a odborné vedení v průběhu vypracovávání tématu. Dále děkuji svým rodičům za psychickou a finanční podporu během studia.

Anotace

Software pro parametrizované úlohy

Cílem této diplomové práce je naprogramování off-line aplikace pro jednoduché vytváření nových příkladů parametrizovaných úloh, sestavování a tisk celých sad příkladů použitelných jako kontrolní testy z matematiky na základních, středních i vysokých školách. V textu jsou popsány prostředky použité k naprogramování této aplikace a problémy které autor řešil.

Annotation

Software for parametric exams

Goal of this work is to programm off-line application for simple creating of new sets of parametrized exams, building and printing of whole sets usable as control tests in maths on every level of education. In this text, there are described resources used for completion of this application and problems solved by author.

Obsah

1 Úvod **9**

2 Parametrizované úlohy **11**
2.1 Historie „Parametrizovaných úloh“ na TU 12
2.1.1 Před dobou počítačovou 12
2.1.2 S pomocí počítačů 12
2.1.3 Na internetu 12
2.1.4 MiStEdit a Prohlížeč 14
2.1.5 Generátor parametrizovaných úloh 15
2.1.6 Písemky 16
2.2 Požadavky kladené na novou aplikaci 16

3 TEX a LATEX **18**
3.1 Historie TEXu 19
3.2 Distribuce systému 20
3.3 Použití 21
3.4 Dokument v LATEXu 21
3.4.1 Prostředí 21
3.4.2 Začátek dokumentu 21
3.4.3 Formát textu 22
3.4.4 Zvláštní znaky a čeština 22
3.4.5 Vzorce 23
3.5 Závěrem k LATEXu 26

4 Vývojové prostředky **27**
4.1 Programovací jazyk a vývojové prostředí 27
4.1.1 Borland Delphi 27
4.1.2 Historie 27
4.1.3 Vlastnosti 28
4.1.4 Mechanismus výjimek 29
4.2 Nápověda 30

4.3 Instalační program 30

5 Program „Parametrizované úlohy“ 31
5.1 Spuštění programu 31
5.2 Formáty datových souborů 32
5.2.1 .TEX 32
5.2.2 .PU 33
5.3 Editace souboru parametrizovaných úloh 33
5.4 Sestavování písemných prací 35
5.4.1 Soubory s úlohami 35
5.4.2 Nastavení vzhledu 37
5.4.3 Výsledný dokument 37
5.5 Vytváření nových souborů parametrizovaných úloh 37
5.5.1 Testové otázky 38
5.5.2 Jednoduché matematické úlohy 38

6 Problémy 41
6.1 Vyhodnocování výrazů 41
6.1.1 Závorky 44
6.1.2 Funkce 45
6.2 Číselné soustavy 45
6.2.1 Nepoziční číselné soustavy 46
6.2.2 Poziční číselné soustavy 47
6.3 Náhodné pořadí příkladů 48

7 Závěrem 50
7.1 Praktické použití 50
7.2 Veřejné prezentace systému 51
7.3 Výhody a nevýhody 52

A Obsah přiloženého CD 54

B Ukázka výsledné písemky 55

Typografické konvence a použité značení

V následujícím textu jsou textové prvky odlišeny takto:

- Pro běžný text je použito normální proporcionální patkové písmo.
- Pro zvýraznění pojmů a názvů použijeme **tučné písmo**.
- *Kurzíva* bude použita pro odlišení méně důležitých pojmů – například názvů programů, apod.
- Pro výpis částí programů a jiných citací programového kódu je zvykem používat `neproporcionální písmo`.
- Internetové odkazy v textu (tedy mimo seznam literatury) budou samozřejmě `podtržené` .
- Bezpatkové písmo bude použito při popisu přesných názvů prvků programu – např. tlačítek, položek menu, apod.

Kapitola 1

Úvod

Tato práce si klade za hlavní cíl vytvořit softwarový systém pro sestavování velkého množství zadání písemných prací a domácích úkolů z matematiky. Program „Parametrizované úlohy“ je stěžejní částí práce a je uložen na doprovodném disku. Pokud by čtenář postrádal některé informace o tomto programu, nechť jej sám vyzkouší. Chybějící informace nalezne také v souboru nápovědy této aplikace.

Je-li cílem této diplomové práce naprogramovat nástroj pro jednoduchou práci se soubory parametrizovaných úloh, je úkolem tohoto textu seznámit čtenáře s tím, co to parametrizované úlohy jsou, s prostředky při vývoji programu použitými, se samotným výsledkem práce a v neposlední řadě také s problémy, se kterými se autor při realizaci práce setkal a se způsobem, jak je vyřešil.

Následující text je členěn do kapitol, každá kapitola je věnována jednomu tématu:

Kapitola první je úvodní. Popisuje rozčlenění a vytváří tak jakýsi komentovaný obsah.

Kapitola druhá se zabývá tím, co to parametrizované úlohy jsou a k čemu je lze použít. Je zde uvedena historie systému na naší univerzitě a ukázány dosavadní aplikace.

Kapitola třetí otevírá téma prostředků použitých při vývoji programu představením typografického systému LaTeX.

Kapitola čtvrtá pokračuje popisem vývojového prostředí Borland Delphi jako aplikace, ve které je nový systém naprogramován. Zmiňuje také další programy při vývoji použité.

Kapitola pátá konečně pojednává o samotném produktu této práce – tedy o programu „Parametrizované úlohy“. Popisuje možnosti aplikace a způsob práce s ním.

Kapitola šestá se zabývá řešením problémů vzniklých při programování této aplikace.

Kapitola sedmá je závěrečnou kapitolou této práce. Zabývá se výsledným stavem projektu a jeho použitím v praxi.

Příloha A je soupisem obsahu kompaktního disku, na kterém se nachází samotný program, jeho zdrojové kódy a další zajímavé materiály. Zdrojové kódy programu mají přes 300 kB, jejich vytištění a zařazení mezi přílohy by pro tuto práci mělo jen mizivý přínos.

Příloha B přináší ukázku výsledného produktu této aplikace – tedy připravené písemky.

Kapitola 2

Parametrizované úlohy

Představme si, že jsme učitelé matematiky na základní škole. Tuto představu můžeme dále konkretizovat zadáním prostého domácího úkolu. Při pohledu na jeho výsledky by nás mohlo zarazit, že víc než 80 procent žáků má nejen stejný (nesprávný) výsledek, ale bohužel i stejnou chybu. Najdeme u nich něco podobného výrazu $2 + 3 = 4$. Domácí úkoly jsou většinou podepsané od rodičů. Taková shoda výsledků může mít mnoho důvodů – nastiňme alespoň tři:

1. Žáci jsou opravdu přesvědčeni o správnosti výsledku. To pro nás jako jejich učitele není zrovna dobrá vizitka.

2. Výsledek je produktem jejich společné práce. Pak se možná scházejí po škole. Možná mají poměrně vysoký účet za telefon. A možná držíme v ruce důkaz toho, že mimosmyslové vnímání opravdu funguje.

3. Rodiče jim podepsali nehotový domácí úkol a oni si jej opsali ráno ve škole od některého ze svých spolužáků.

Přestože byl úkol zadáván jako samostatná práce, jeho výsledky ukazují, že jako taková nebyl vypracováván. Ponecháme však stranou negativa jednotlivých zde nastíněných možností a soustředíme se na otázku, zda by nebylo možné takovéto situaci zabránit a nezatěžovat tak svědomí žáků nepříliš morálně čistými praktikami.

Proti opisování při písemce se již dlouhá léta používá osvědčený postup: vytvořit více variant zadání. Nejčastěji se vytvářejí varianty dvě, tradičně označované jako A a B. I při takovém postupu je opisování možné a bohužel také obvyklé. Pokud by však vyučující chtěl vytvářet víc variant písemky, strávil by její přípravou neadekvátní množství času. Navíc by šlo o práci spíše rutinní a nezajímavou, neboť všechny varianty mají být srovnatelně obtížné.

Naštěstí můžeme v dnešní době takové činnosti svěřit počítačům. Výsledkem pak budou desítky (až tisíce) stejných zadání, lišících se jen vstupními hodnotami – parametry. Pro vzniklé úlohy pak budeme používat název „**Parametrizované úlohy**“.

Takto můžeme dokonce vytvářet stejný počet zadání, jaký je počet žáků. Samozřejmě, tímto způsobem nezabráníme „inspirování se“ postupem spolužáka, mělo by se tak však odstranit doslovné opisování, které je ve školách až příliš obvyklé.

2.1 Historie „Parametrizovaných úloh“ na TU

2.1.1 Před dobou počítačovou

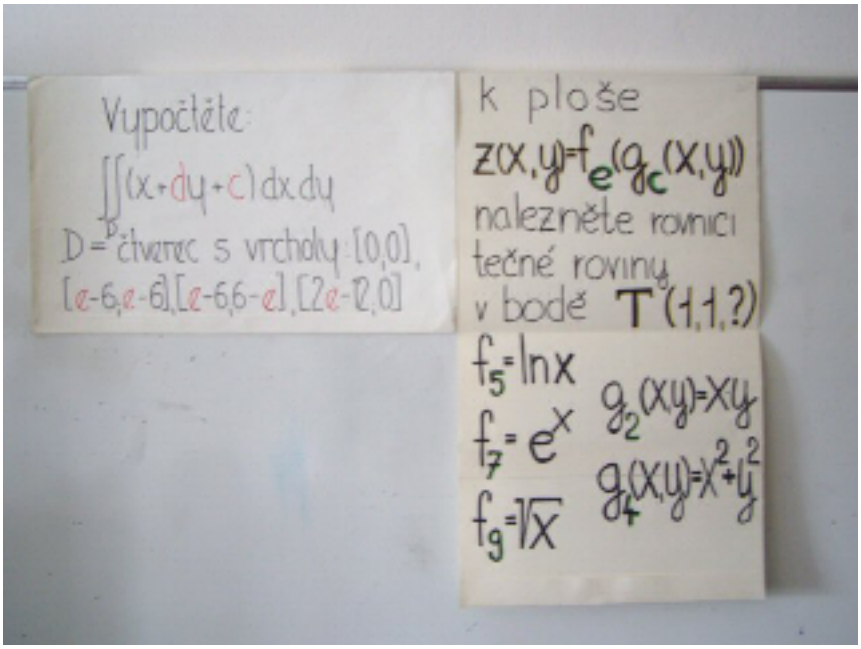
Parametrizovaná zadání mají na naší škole již dlouhou tradici – již od roku 1966 se na katedře matematiky používala tzv. proměnná zadání. Prvním systémem byl „EXAS“, jehož autory byli pánové Nekvinda, Šrubař a Vild, a který vznikl na konci sedmdesátých let. Jako autoři dalšího systému je uváděna čtveřice Kalousek, Přívratská, Staněk a Staňková. Myšlenka tohoto systému byla jednoduchá. Studenti naší vysoké školy dostali na začátku ročníku přidělený pěticiferný kód a cifry tohoto kódu dosazovali na určená místa v zadáních. To, jak tato zadání vypadala, je zachyceno na obrázku 2.1. Tímto způsobem získal každý student své vlastní zadání práce, kterou tak musel vypracovat samostatně.

2.1.2 S pomocí počítačů

Nyní přeskočme několik let a vydejme se do éry počítačů roku 1997. V tomto roce napsal Kalousek verzi systému v jazyce C++. Nicméně, ani jednu z dosud popsaných verzí systému nelze považovat za předchůdce systému nově vyvíjeného. Je však pravda, že databáze příkladů z tohoto systému posloužila jako základ souboru příkladů pro následující program.

2.1.3 Na internetu

Tím je aplikace fungující od roku 1999 na katedře aplikované matematiky na internetové adrese `www.fp.vslib.cz/matika` . Tuto on-line aplikaci naprogramoval student Martin Michale strojní fakulty naší univerzity roku 2000 v rámci své diplomové práce [21] pod vedením Doc. Miroslava Brzeziny. Aplikace, přístupná přes internet (a dostupná tak v podstatě odkudkoli komukoli



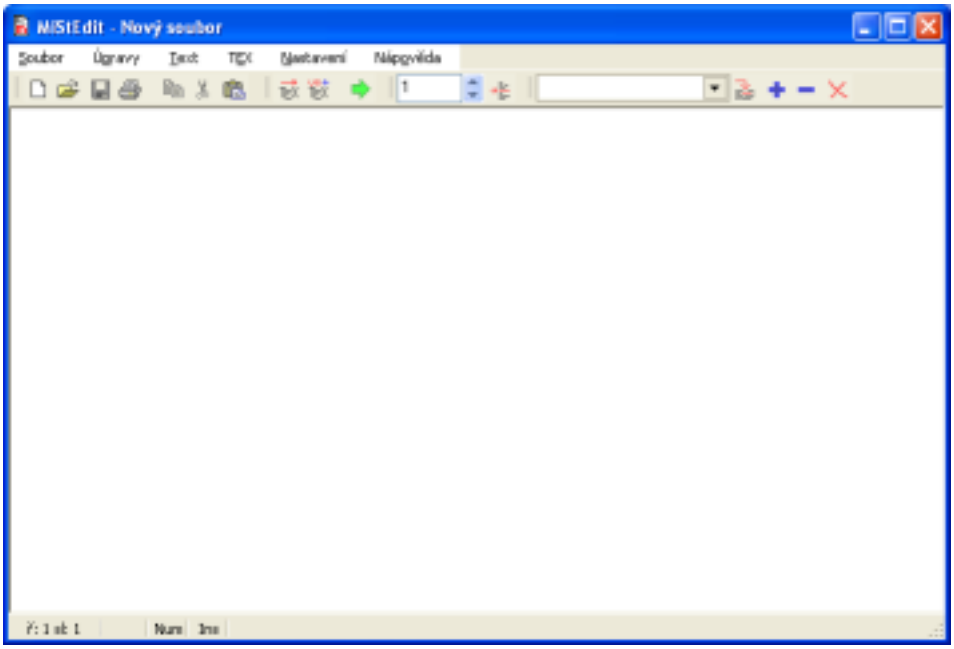
Obrázek 2.1: Papírová verze „Parametrizovaných úloh“ ze sedmdesátých let.

znalému hesla a uživatelského jména) sloužila pro vytváření testů a samostatných prací určených studentům liberecké Technické univerzity. V té době obsahovala její databáze příklady z matematiky pro vysoké a střední školy. Dnes ji uživatel najde na adrese `zombie.kap.vslib.cz/matematika` .

V roce 2002 začala trojice studentů (Martin Pecka, Michal Stehlík, Tomáš Vojtek) z tehdejšího druhého ročníku programovat soubory úloh pro základní školy v rámci grantu FRNŠ č. 1463/2002, pod vedením RNDr. Šimůnkové. Výsledkem jejich práce bylo množství souborů obsahujících sta až tisíce variant zajímavých (nebo naopak častých) příkladů ze „základoškolské“ matematiky.

Přirozeně ani tehdy tyto soubory nevznikaly ručně. Každý z této trojice uměl dostatečně programovat v některém z programovacích jazyků – ať už v Pascalu, nebo v jazyce C. I přes počítačovou podporu při samotném vytváření příkladů, stále ještě zbývalo vyřešit některé problémy.

Nejdůležitějším z těchto problémů bylo kódování češtiny. Historicky existuje víc než šest různých standardních (a nepočítaně nestandardních) způsobů kódování akcentovaných znaků do celosvětového standardu ANSI. Vybraný systém LaTeX naštěstí obsahuje možnost jednotného zapisování akcentovaných znaků. I přesto však bylo nutné převést češtinu kódovanou nejčastěji prostřednictvím systému Windows na požadovaný zápis (tento problém bude



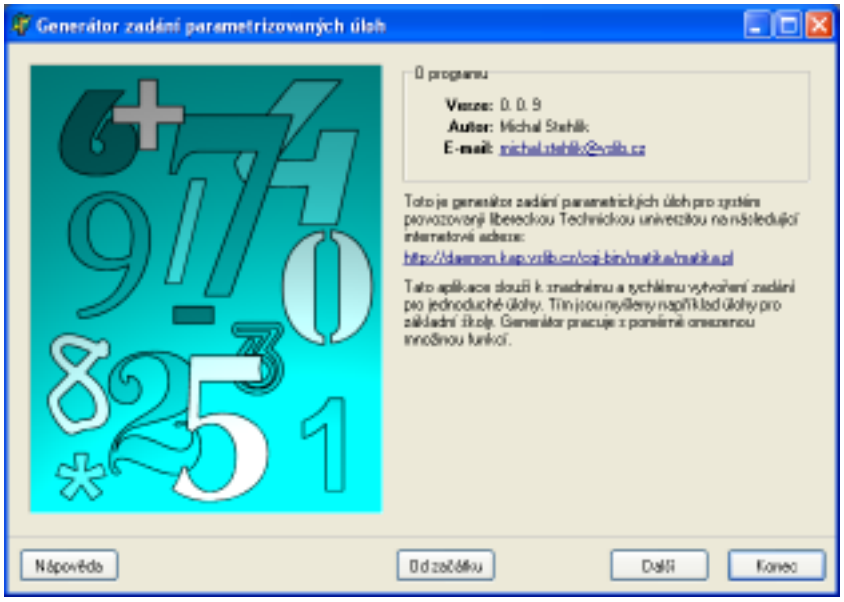
Obrázek 2.2: Jednoduchý editor „MiStEdit“ obsahoval funkce pro vytváření zdrojových souborů parametrizovaných úloh.

podrobněji popsán v kapitole 3.4.4).

Všichni studenti se věnovali vyhledávání vhodných příkladů a sestavování programů pro jejich vložení do systému. Michal Stehlík však k tomu ještě převáděl texty do požadované podoby. Samozřejmě si chtěl práci maximálně usnadnit naprogramováním nástroje pro automatizování této činnosti. Zároveň začalo být jasné, že omezený počet jedinců nebude nikdy schopný vytvořit tolik úloh, aby se systém dočkal širšího použití. Myšlenka vytvořit nástroje, které by umožňovaly širší veřejnosti vytvářet vlastní soubory příkladů se tak zdála býti lákavou. Autorem následujících programů je právě autor této práce – Michal Stehlík.

2.1.4 MiStEdit a Prohlížeč

První program, nazvaný MiStEdit, byl jednoduchý textový editor, podobný například všudypřítomnému „Poznámkovému bloku“. Navíc však obsahoval průvodce pro návrh zdrojových souborů programovacích jazyků Pascal a MatLab. Tento kód byl zdrojem pro soubor parametrizovaných úloh. Podobně jednoduchým programem byl také „Prohlížeč“, umožňující zobrazovat vygenerované soubory ve stejné podobě, jakou jim měl poskytovat on-line



Obrázek 2.3: Tato jednoduchá aplikace sloužila k sestavování souborů příkladů pro základní školy.

systém. Takto mohla být laděna i vizuální podoba připravovaných příkladů.

2.1.5 Generátor parametrizovaných úloh

Jestliže první dva programy měly pouze usnadnit práci programátorům souborů paramemetrizovaných úloh, program třetí si kladl cíle odlišné. Úspěšné komerční i nekomerční projekty často umožňují uživatelům vytvářet vlastní části programu pomocí speciálního editoru. I v případě tohoto projektu by bylo ideální umožnit uživatelům (tedy učitelům) vytvářet si vlastní příklady a ty poté zařazovat do systému.

Tuto funkci měl zajišťovat jednoduchý program, který formou oblíbeného průvodce, známého i z jiných aplikací pro Windows, zjistil od uživatele potřebné informace a na jejich základě sestavil kompletní soubor parametrizovaných úloh. Naplnění této funkce je především dáno schopností programu porozumět vkládaným matematickým výrazům. Matematické jádro, kolem kterého byl tento program postaven a které mělo aritmetické výrazy analyzovat, však trpělo několika neduhy. Snad nejzávažnějším byla neschopnost jádra vyhodnocovat prioritu operátorů. Díky této chybě bylo nezbytné výraz správně uzávorkovat. V programovacích jazycích není neobvyklé, že překladače priority operací vyhodnocují jinak, než je zvykem v matematice. Tento

program však byl určen pro učitele matematiky a pro ty by byl jiný způsob vyhodnocování matoucí. Dalším neduhem byla omezená skupina funkcí, které mohly být programem vyhodnoceny. Tyto chyby byly později opraveny.

Program byl v roce 2003 předveden na semináři pro učitele matematiky na základních a středních školách pořádaném katedrou aplikované matematiky. Výsledkem této demonstrace byla řada připomínek, vedoucích ke zlepšení uživatelského rozhraní a funkčnosti programu. Přestože byla tato ukázka úspěšná, odhalila slabiny programu a ukázala směr jeho dalšího vývoje.

2.1.6 Písemky

Nejprve však byl učiněn pokus nahradit existující on-line aplikaci takovým programem, který by umožnil sestavovat testy a písemky i uživatelům nepřipojeným na internet. Zároveň by díky tomuto programu mohli uživatelé sestavovat své vlastní písemky se svými vlastními příklady. To, že tento program je pro celý projekt klíčový, se ukázalo během výpadku serveru na katedře aplikované matematiky. Uživatelé by takto neměli přístup k možnosti sestavovat písemky pro své žáky.

Program „Písemky“ měl tuto díru zacelit. Byl dokončen nedlouho před svým představením na prezentaci prací SVUČ. Zcela nahrazoval možnosti svého předchůdce a doplňoval několik nových funkcí – vkládání jmen žáků do záhlaví, změny vzhledu a možnost upravovat již existující soubory příkladů. Program byl sice dokončen, nikdy však nebyl zveřejněn. Směr ukázaný požadavky učitelů u „generátoru“ si žádal integraci všech těchto programů do jediné aplikace. Vydat další jednoúčelový program by bylo zbytečné.

2.2 Požadavky kladené na novou aplikaci

Před začátkem práce na jakémkoli projektu je třeba přesně definovat zadání. K tomu je v tomto případě třeba odpovědět na několik otázek.

K jakému účelu má program sloužit? Program má plnit několik funkcí. Nejdůležitější je sestavování a tisk zadání písemných prací. Další funkcí je vytváření a editace souborů zadání parametrizovaných úloh.

Pro koho jsou určeny výstupy programu? Výsledná zadání předloží vyučující svým žákům. Ti musí zadání přečíst a úlohy vypracovat. V úlohách z některých předmětů se mohou vyskytnout i poměrně netypické prvky – jako například vzorce. Zadání tak musí být pohodlně čitelné a musí do něj být možné tyto prvky vkládat v typograficky správné podobě.

Kdo bude s programem pracovat? Učitelé na všech stupních škol. Z hlediska práce s počítačem jsou v této skupině zastoupeni „počítačově negramotní“, běžní uživatelé, pokročilí uživatelé a také programátoři. Program navrhovaný pro běžného uživatele by měl mít jednoduché a přehledné ovládání takového stylu, na který je uživatel zvyklý. Funkce programu by měly být dokumentovány v souborech nápovědy.

Jaká je tedy úroveň počítačové gramotnosti uživatelů? Tato úroveň je bohužel velmi rozdílná – většinou však nezahrnuje programování. I když již od počátku své existence měly počítače velmi blízký vztah s matematikou a schopnost algoritmizace je v matematice klíčová, není znalost alespoň základů programování vůbec běžná. Pokud tedy má program nabízet možnost tvorby vlastních matematických úloh, musí být tato pojata jednoduchým způsobem, který je uživatelům známý a blízký.

Kde bude program provozován? Předpokládá se, že program bude nainstalován na počítačích ve školách a v domovech učitelů. Počítače na těchto místech mají rozdílný výkon, o drtivé většině z nich lze předpokládat, že jsou osazeny operačním systémem firmy Microsoft, získaným v rámci OEM prodeje a zvýhodněných licencí pro školy. Kromě toho je však pro tato místa typický nedostatek finančních prostředků. Proto by program a jeho komponenty měly být velmi levné, pokud možno zcela zdarma.

Tyto požadavky můžeme shrnout následovně:

- Typograficky správný výstup
- Jednoduché a příjemné rozhraní
- Cenová dostupnost
- Snadná obsluha
- Hardwarová nenáročnost
- Srozumitelná dokumentace
- Nekomplikované prostředí pro vytváření nových příkladů
- Zpětná kompatibilita se stávajícím on-line systémem

Z těchto požadavků plyne volba prostředků k vytvoření této aplikace a způsob jejího naprogramování.

Kapitola 3

TEX a LATEX

Pokud vytváříme parametrizované úlohy a pracujeme s nimi na počítači, budeme je pravděpodobně pomocí počítače také zobrazovat. Jestliže se rozhodneme naprogramovat aplikaci pro práci s takovými úlohami, zdálo by se logické naprogramovat také jejich zobrazování. Naprogramování kompletního prostředí pro správné zobrazování včetně alespoň základních typografických pravidel je však velmi složité. Situaci navíc nemálo komplikuje skutečnost, že v tomto případě chceme zobrazovat také složitější vzorce, což je činnost, kterou dodnes nezvládají ani profesionální textové editory. Proto je logické, rozhodnout se využít nějaký již existující software. V úvahu lze vzít následující produkty:

- **Standardní textový editor** má velkou výhodu v širokém rozšíření. Tyto programy se nacházejí v téměř každém počítači. Problémem zde jsou datové formáty. Každá firma totiž má vlastní formát a z těchto formátů je komplikované třeba i jen data získat, natož je měnit, či dokonce sestavovat vlastní soubory.

- **MathXML** je speciální formát podobný HTML a XML, schválený nedávno konsorciem W3C jako standard pro zobrazování matematických výrazů. Tento formát je nový a má být zobrazitelný v každém internetovém prohlížeči. Bohužel, prohlížeče jsou v tomto ohledu nespolehlivé, neboť jejich výrobci standardy příliš nectí.

- **TEX** je v akademickém prostředí poměrně oblíbený formát, napsaný matematikem a snad právě proto velmi vhodný právě pro zobrazování vzorců. Datový soubor je snadné vytvořit, neboť jde o posloupnost textových příkazů. Interpret těchto příkazů – tedy systém TEX lze navíc získat zdarma. Pro náš účel je tak tento formát nejvhodnějším kandidátem.

3.1 Historie TEXu

TEX a také LATEX jsou typografické systémy určené k sazbě vědeckých a hlavně matematických dokumentů velmi vysoké typografické kvality. Pochopitelně se tento systém neomezuje jen na sazbu dokumentů vědeckých, stejně dobré výsledky získáme i při zpracovávání dopisů nebo knih. Systém LATEX je postaven nad formátovacím programem TEX Donalda E. Knutha.

Profesor Knuth ze Stanfordské univerzity, známý expert v oboru výpočetní techniky, jej tehdy údajně [10] napsal, aby mohl své matematické texty publikovat v požadovaném tvaru, neboť sazeči v tiskárně vnesli obvykle do matematických vztahů mnoho chyb. To se mu podařilo dokonale – v TEXu nedošlo od roku 1983, kdy byla zveřejněna jeho první verze, k žádným významným změnám. Program byl od počátku koncipován tak, aby umožňoval přizpůsobení národnímu prostředí, díky čemuž lze vkládat do textu libovolné akcenty ke znakům, používat exotické abecedy nebo dokonce psát zcela odlišným způsobem a směrem.

Samotný název programu je tvořen první trojicí písmen řeckého slova pro technologii a umění. To odpovídá tomu, že posláním TEXu je produkování umělecky velmi kvalitních technických textů. V názvu programu jsou tato písmena použita jako velká, a tak se $\tau\varepsilon\chi$ jeví jako TEX. Protože je posledním písmenem názvu „chí“, čte se správně ne „tex“, ale „tech“.

Stejnou chybou by bylo čtení „latex“, místo správného „la-tech“, případně „lej-tech“. LATEX je název souboru maker, jejichž autorem je Leslie Lamport. Byl vytvořen, aby zjednodušil sazbu dokumentů v TEXu a přiblížil tak poněkud složitý jazyk běžnému uživateli. Nabízí například automatické číslování objektů (kapitoly, obrázky, apod.), dále sestavuje obsahy a rejstříky, má předdefinovaný vzhled kapitol a dalších objektů.

LATEX je nástroj, na kterém se stále pracuje. Od verze 2.09 vzniklo mnoho rozšiřujících verzí tohoto systému. Cílem týmu vedeného Frankem Mittelbachem je vytvoření standardního, jednotného LATEXu – ten je označován jako LATEX3. Současná verze je pak LATEX 2_ε – a právě to je systém, ve kterém je také zpracována tato práce.

Původní sada maker však byla vytvořena dle amerických konvencí. Ty se od těch českých značně liší, stejně jako se liší jazyky v těchto zemích používané. Jednou z možností, jak používat správné české prostředí, je $\mathcal{CS}$ LATEX. Tím nastavíte správné uvozovky, dělení slov, názvy – „Kapitola“, „Dodatek“, apod. a datumový formát.

3.2 Distribuce systému

TEX je používán na tisících počítačů po celém světě. Pak je ovšem pochopitelné, že existují verze pro nejrozšířenější operační systémy. Vzhledem k nekomerční, akademické povaze TEXu je použití na Linuxu samozřejmostí. Nicméně již pro MS-DOS existovala distribuce jménem emTEX.

My se však soustředíme na distribuce pro operační systémy Windows. Z nich jsou nejznámější následující dvě:

MikTEX je vyvíjen a šířen na adrese `www.miktex.org` . V době vzniku této práce je aktuální verze 2.4. Ve světě je tato distribuce oblíbenější, její použití v našich podmínkách je poněkud problematické kvůli nutnosti doinstalovat češtinu.

TEXLive lze nalézt na `tug.org/texlive` . Distribuce existuje ve třech verzích: Jako **live** se označuje kompletní verze spustitelná přímo z DVD (je příliš velká, než aby se vešla na obyčejné CD). **Inst(allable)** je rovněž kompletní varianta. Před jejím použitím je však nutné ji nainstalovat na pevný disk počítače. Poslední možností je **demo**, což je systém ochuzený o možnost psát noty, psát čínsky, japonsky a korejsky. Je určen pouze pro Windows, Linux a Mac OS. Lze jej používat přímo z distribučního CD.

V následující tabulce se nacházejí velikosti zde zmiňovaných instalačních balíků.

Balík	**Varianta**	**Velikost (MB)**	**Instalace (MB)**
MikTEX	Small	24 - 250	60
TEXLive	Live 2003	493	0
	Inst 2003	545	264
	Demo 2003	236	
	Inst 2004		
WinShell		2	4
WinEdt		5	0
Parametrizované úlohy		2	7

Tabulka 3.1: Přehled velikostí instalačních balíků programů použitelných při práci s TEXem a Parametrizovanými úlohami. Tabulka obsahuje i orientační velikosti po nainstalování.

3.3 Použití

Protože LaTeX není žádný textový editor, ale profesionální typografický systém vytvořený inteligentními lidmi pro inteligentní lidi, používá se způsobem, známým z dávných dob systému MS-DOS a nedávných dob systému UNIX – TeX se řadí mezi „řádkově orientované“ programy. To znamená, že systém je spouštěn napsáním správně formátovaného příkazu do příkazové řádky. Například překlad tohoto dokumentu s češtinou a ve formátu PDF se spouští povelem:

```
e:\TEXLive\bin\pdfcslatex.bat e:\diplomka\diplomka.tex
```

Samozřejmě, již zmiňovaní inteligentní lidé se snažili práci si usnadnit. Výsledkem jejich snahy je dlouhá řada textových editorů zvýrazňujících syntaxi TeXu a umožňujících relativně pohodlnou práci s dokumenty. Jmenujme alespoň dva takové editory – a to **WinShell for TeX** (`www.winshell.de`) a **WinEdt** (`www.winedt.com`).

3.4 Dokument v LaTeXu

Byla zde již uvedena historie TeXu a způsob, jak systém spustit. Pro člověka rozhodnutého napsat dokument v LaTeXu je však spíše důležité vědět, jak takový dokument vypadá. V následujícím textu bude popisován systém LaTeX. Nemá však jít o vyčerpávající příručku, ale spíše o popis možností tohoto typografického programu.

3.4.1 Prostředí

Pro LaTeX je typické použití tzv. prostředí. Začátkem takového prostředí je příkaz `\begin{` *název prostředí* `}`. Koncem prostředí pak je celkem logicky `{`*název prostředí*`}`. Mezi touto dvojicí značek se nachází blok textu, formátovaný způsobem daným vlastnostmi prostředí. Takto je například realizováno zarovnávání textu na prapor doleva, doprava nebo na střed.

3.4.2 Začátek dokumentu

Soubor dokumentu v LaTeXu začíná definicí stylu dokumentu a velikosti základního písma ve tvaru `\documentclass[` *volitelné parametry* `]{` *třída dokumentu* `}` kde

Volitelné parametry zahrnují druh papíru (`a4paper`, `a5paper`), způsob tisku a jiná nastavení vzhledu.

Styl dokumentu je obecné nastavení druhu dokumentu. Nejběžnější styly jsou `article` pro krátké zprávy rozsahu několika stran, `report` pro rozsáhlejší zprávy členěné do kapitol, `book` pro sazbu celých knih a `slides` pro tisk prezentačních materiálů na fólie. Samozřejmě existují i jiné styly dokumentů určené pro specifická použití. Styl dokumentu nastavuje například způsob číslování stránek, popisů částí textu a možnost jednostranného tisku.

Následují inicializace vkládaných balíků prostřednictvím `\usepackage{` *balík* `}`. Takovými balíky jsou implementovány nestandardní rysy dokumentu, jako je například vkládání obrázků a použití české národní diakritiky.

Samotný zobrazovaný dokument je vlastně prostředí nazvané `document`. V tomto prostředí jsou popsány jednotlivé prvky dokumentu – kapitoly, podkapitoly, odstavce, obrázky a tabulky.

Například tato práce začíná následovně:

```
\documentclass[a4paper,12pt]{report}
\usepackage{czech}
\usepackage[pdftex]{graphics}
\begin{document}
```

3.4.3 Formát textu

Systém bez možností definovat způsob formátování odstavců a znaků by nebylo možné nazvat systémem typografickým. Ani v TEXu tyto možnosti nechybí. Například zarovnání odstavců je realizováno pomocí prostředí – lze je zarovnávat vlevo (`flushleft`), vpravo (`flushright`) nebo na střed (`center`).

Možnosti formátu znaků sahají od změny velikosti písma (`\tiny`, `\scriptsize`, `\footnotesize`, `\small`, `\normalsize`, `\large`, `\Large`, `\LARGE`, `\huge`, `\Huge`), přes změnu typu písma až k různým zvláštním efektům.

3.4.4 Zvláštní znaky a čeština

Protože náš mateřský jazyk – čeština – obsahuje několik znaků, které nejsou ve světě zrovna obvyklé, vzniklo v průběhu minulých let víc než sedm různých způsobů kódování naší národní abecedy do standardního ASCII. LATEX používá svůj vlastní způsob zápisu znaků s akcenty. Ukázka těchto znaků je v následující tabulce.

ò	\‘{o}	õ	\~{o}	ǒ	\v{o}	o̧	\c{o}
ó	\’{o}	ō	\={o}	ő	\H{o}	ọ	\d{o}
ô	\^{o}	ȯ	\.{o}	o͡o	\t{oo}	o̲	\b{o}
ö	\"{o}	ŏ	\u{o}	ů	\r{o}		
œ	\oe	å	\aa	ł	\l	¿	?‘
Œ	\OE	Å	\AA	Ł	\L	¡	!‘
æ	\ae	ø	\o	ß	\ss	Æ	\AE
Ø	\O	†	\dag	§	\S	©	\copyright
‡	\ddag	¶	\P	£	\pounds		

Tabulka 3.2: Znaky s akcenty a speciální symboly

3.4.5 Vzorce

Jednoznačně nejsilnější stránkou TEXu je sázení vzorců. Tato vlastnost je také pro každého matematika nejzajímavější a nejdůležitější. Pro psaní matematických výrazů disponuje TEX vlastním prostředím. Do něj lze vstoupit několika způsoby. Vzorce můžeme vkládat přímo do textu, nebo na střed samostatného řádku, vzorce mohou být automaticky číslovány, jednotlivé rovnice mohou být správně zarovnávány s rovnítky pod sebou.

Nejobvyklejším prostředím pro sazbu matematických vzorců je prostředí `math`. Používá se pro vkládání vzorců přímo do textu. Místo obvyklých příkazů začátku prostředí je zvykem používat symboly `\(` a `\)` a nebo (a to nejčastěji) `$`. Pak tedy sekvence `$c=\sqrt{a^2+b^2}$` bude vypadat takto: $c = \sqrt{a^2 + b^2}$.

Prostředí `displaymath` se používá pro sazbu vzorců na střed nové řádky. To je ideální pro velké nebo důležité vzorce. I toto prostředí bývá častěji voláno alternativními symboly `\[` a `\]`. Najčastěji se používá symbolu `$$`. Vzorec

$$c = \sqrt{a^2 + b^2}$$

vznikne z řetězce `$$c=\sqrt{a^2+b^2}$$`.

Při sazbě rozsáhlých matematických textů je pro větší přehlednost tyto vzorce očíslovat. Číslování vzorců zajišťuje prostředí `equation`. Vzorce se číslují v rámci kapitoly. V tomto prostředí tak vzorový vzorec vypadá takto:

$$c = \sqrt{a^2 + b^2} \tag{3.1}$$

Poslední zde uvedenou možností je sazba posloupnosti rovnic, které mají být vzájemně zarovnávány. Předchozími prostředky vysázené rovnice totiž například nebudou mít „rovnítka“ pod sebou. Pokud v prostředí `eqnarray` připravíme rovnice následovně:

```
\begin{eqnarray}
3x &= & 6(x-9) + 15 \\
3x-6 &= & -54 +15\\
x &= & 13
\end{eqnarray}
```

pak bude ve výsledku vypadat takto:

$$\begin{aligned} 3x &= 6(x-9)+15 && (3.2)\\ 3x-6 &= -54+15 && (3.3)\\ x &= 13 && (3.4) \end{aligned}$$

Uvnitř matematických prostředí lze psát vzorce a používat zvláštní symboly. Mezi tyto symboly patří například písmena řecké abecedy – písmeno α tak v běžném textu musíme vložit sekvencí `$\alpha$`.

Základní možnosti matematického prostředí

Uvnitř matematického prostředí lze již používat širokou škálu výrazových prostředků, na které jsou již matematici zvyklí. Indexy se tvoří pomocí znaménka „_“ a exponenty prostřednoctvím „^“. Zlomky lze psát jak šikmou zlomkovou čárou, tak i čárou vodorovnou. Pak se `$$x=\frac{y+z/2}{y^2+1}$$` zobrazí jako

$$x = \frac{y+z/2}{y^2+1}.$$

Do vzorce lze vložit symbol odmocniny tak, že `$c=\sqrt{a^2+b^2}$` vypadá jako $c = \sqrt{a^2+b^2}$ a `$y=\sqrt[6]{-1}$` jako $y = \sqrt[6]{-1}$.

Složitější výrazy

Kromě takto jednoduchých vzorců lze samozřejmě sázet také složité výrazy obsahující sumy, diferenciály, integrály a jiné typograficky složité prvky. Znaková sada používaná ve vzorcích obsahuje všemožné typy šipek, relačních symbolů (včetně množinových) binárních operátorů a také další symboly. Předvedeme si jen několik ukázek (vždy bude uveden zdrojový text a pod ním výsledná podoba):

```
$$\sum_{i=1}^n f_i \rightarrow \bigcup_0^1 f\,\mathrm{d}x$$
```

$$\sum_{i=1}^{n} f_i \rightarrow \bigcup_0^1 f\,\mathrm{d}x$$

```
$$\int_E s\,\mathrm{d}\mu = \sum_{i=1}^n \alpha_i \mu(A_i \cap E)$$
```

$$\int_E s \,\mathrm{d}\mu = \sum_{i=1}^{n} \alpha_i \mu(A_i \cap E)$$

Vektory a matice

Jak vektory, tak i matice jsou realizovány prostředím `array`. Toto prostředí je ekvivalentní prostředí pro vytváření tabulek (`tabular`). Parametrem tohoto prostředí je sekvence udávající způsob, jakým bude daný sloupec zarovnáván – možnosti jsou: `c` (na střed), `l` (doleva), `r` (doprava), kromě toho lze symbolem `|` vložit mezi sloupce svislou čáru.

Zdrojové texty většiny složitějších schémat jsou však poněkud nepřehledné. Příkladem budiž obecný zápis soustavy rovnic:

$$\begin{array}{l} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n \end{array}$$

Ten vznikl z řetězce:

```
\begin{displaymath}\begin{array}{*{3}{c@{\:+\:}}c@{\;=\;}c}
a_{11}x_1 & a_{12}x_2 & \cdots & a_{1n}x_n & b_1 \\
a_{21}x_1 & a_{22}x_2 & \cdots & a_{2n}x_n & b_2 \\
  \multicolumn{5}{c}{\dotfill} \\
a_{n1}x_1 & a_{n2}x_2 & \cdots & a_{nn}x_n & b_n
\end{array}
\end{displaymath}
```

Když možnosti nestačí

Přestože existují obecně uznávané konvence pro sazbu matematického textu (stanovené normou ISO), je třeba říci, že LaTeX je ne zcela dodržuje. Naštěstí (a na rozdíl od jiných typografických systémů) jsou známy postupy, jak tyto normy splnit – tyto postupy se nacházejí například v [12] na str. 138.

Již od první verze TeXu podporovala jeho vývoj americká matematická společnost AMS. Tato společnost brzy zveřejnila svůj balík maker zjednodušující sazbu matematických textů. Po vzniku LaTeXu společnost AMS začala pracovat na portování tohoto balíku pro LaTeX. Výsledný produkt se nazývá AMS-LaTeX a jeho obsah lze rozdělit do tří částí: balíčky pro rozšíření nástrojů sazby matematických podkladů, zvláštní třídy určené pro články a knihy a konečně doplňkové fonty obsahující další symboly, matematickou abecedu a azbuku.

Pokud se tedy sazeč dostane do situace, kdy mu nebudou stačit (i tak impozantní) možnosti LaTeXu, má možnost připojit do textu balíčky AMS-LaTeXu, které by měly uspokojit i nejnáročnější matematickou sazbu. Ostatně – vzhledem k míře usnadnění a komfortu, kterou tyto balíky nabízejí, je rozumné matematické texty vytvářet rovnou v tomto prostředí.

3.5 Závěrem k LaTeXu

Pokud vezmeme v úvahu pouze vzhled výsledného dokumentu, nemá LaTeX konkurenci. Navíc je téměř zdarma, umožňuje výstup do vhodných formátů (PostScript a PDF), je mnohostranně rozšiřitelný. Je obzvláště vhodný pro sazbu matematických vzorců.

Na druhou stranu není jeho instalace a formátování textu v něm právě triviální záležitostí. Na jeho instalaci ovšem existují podrobné návody.

Součástí vyvíjeného programu by tak měly být prostředky, které formátování textu usnadní. Měly by však zároveň zachovávat určitou tvůrčí svobodu. Vhodné by tedy bylo některé činnosti automatizovat a pro ostatní vytvořit panel s nejobvyklejšími možnostmi, umožňující alespoň dosazovat hotové části textu způsobem známým například z editorů HTML.

Kapitola 4

Vývojové prostředky

4.1 Programovací jazyk a vývojové prostředí

Z požadavků kladených na novou aplikaci vyplývá nutnost ji napsat pro prostředí MS Windows. Uživatelé jsou z tohoto prostředí zvyklí na určitý styl ovládání a celkovou filosofii programů. Nemá proto smysl vytvářet aplikaci, na kterou by si uživatelé museli dlouho zvykat.

4.1.1 Borland Delphi

Jedním z prostředí, které programátorovi umožní jednoduše vytvořit takovéto uživatelské rozhraní, je Delphi firmy Borland. Prostředí je založeno na jazyce Pascal. Jazyk používaný v tomto prostředí byl označován jako „Object pascal“, ale nyní se pro něj pomalu prosazuje název „Delphi“.

4.1.2 Historie

Pascal je jedním z relativně moderních programovacích jazyků. Čerpá ze zkušeností s prací se staršími jazyky – jako jsou *Fortran*, *Cobol* a *PL/1*. Podle jeho tvůrce *Niklause Wirtha* vývoj Pascalu začal v roce 1968, jeho první implementace byla dokončena v roce 1970.

Na rozdíl od jazyka *C*, který byl navržen jako jazyk vysoké úrovně s možností pracovat i na nízké úrovni jazyka (tedy pracovat s assemblerem) a je tedy vhodný pro programování operačních systémů a vysoce výkonných rutin, je Pascal určen pro jednoduchý vývoj aplikací a jeho hlavní výhodou je přehlednost zdrojového kódu, která usnadňuje správu projektu a předurčuje tak jazyk k vývoji rozsáhlejších programů.

V roce 1980 byl vydán standard ISO Pascal a v následujících letech se Pascal dočkal velké obliby a rozšíření. S příchodem objektového programování

však Pascal začal ze scény pomalu ustupovat – implementace objektového rozšíření se ukázala jako nepříliš dobrá a ve vývoji aplikací začal dominovat jazyk C++. Pascal byl tak zdánlivě odsouzen do role výukového jazyka. Nejznámějším překladačem a vývojovým prostředím byl Turbo Pascal firmy Borland.

Když tato firma začala uvažovat o znovudobytí pozic na trhu vývojových prostředí, měla na výběr hned z několika jazyků, které mohla použít. Pokud by se rozhodla pro jazyk C, střetla by se však přímo s firmou *Microsoft* a jejím „Visual C++“. Borland tehdy vsadil na ustupující jazyk Pascal. Sázka to byla riskantní, ale rozhodně se vyplatila. Když v roce 1995 vyšla první verze nástroje Delphi vybaveného modernizovaným, plně objektovým Pascalem, dočkala se ohromného úspěchu.

4.1.3 Vlastnosti

Borland Delphi se řadí mezi tzv. prostředí *RAD* (Rapid Application Development). To znamená, že v Delphi lze snadno a především velmi rychle vytvářet nejrůznější programy. Jejich obrovskou výhodou je velmi pohodlné (až přátelské) prostředí. Jsou především určeny pro vývoj středních a velkých programů. Existují však také varianty (*Personal Edition*) pro vývoj aplikací skromnějších. Tyto verze byly vydány na CD nosičích přibalených k časopisům (v ČR šlo například o časopis „Computer“). Pro vývoj aplikace „Parametrizované úlohy“ tato verze prostředí zcela dostačovala.

Velkou výhodou jsou existující knihovny „komponent“, které umožňují do aplikace zahrnout nové prvky. Těmito prvky může být jednodušší přístup k databázím, k internetu, sítím, používání nových prvků uživatelského rozhrani. Většinou však do těchto komponent není možné zasahovat a přímo do nich přidávat nové funkce. Vlastností objektového přístupu však je možnost vytvořit objekt dědící všechny vlastnosti jiného objektu, doplněný o vlastnosti nové. Jsou však také případy, kdy tento postup buď není vhodný, nebo je nevýhodný.

Pascal v Delphi si stále zachovává určitou jednoduchost konstrukcí a přehlednou syntaxi, díky čemuž jsou Delphi stále používané pro výuku programování. K této jednoduchosti se však přidávají také velice pokročilé rysy jazyka – například objektovost jazyka a mechanismus „výjimek“ . Přestože v tomto textu nemá valného smyslu zabývat se podrobně jazykem Pascal, tak, protože program „Parametrizované úlohy“ je právě na výjimkách postavený, můžeme se na tento mechanismus podívat podrobněji.

4.1.4 Mechanismus výjimek

Smyslem výjimek je zvýšit odolnost programu doplněním schopnosti jednoduchého a jednotného řešení hardwarových a softwarových chyb. Pro řešení chyb se obecně dá použít několik strategií. První (a téměř nepoužitelnou) je napsat program tak, aby v něm k chybě nemohlo dojít. To lze víceméně jen u programů, do jejichž chodu nemůže nijak zasahovat ani uživatel, ani jejich okolí. Dalším postupem je předcházení chybám. To v praxi znamená, že například před každým přístupem k souboru musíme zkontrolovat, zda existuje, je otevřený a na disku je dostatek místa. Programování takové aplikace vyžaduje osobu schopnou myslet na několik kroků dopředu, předvídat, co se může pokazit. Výsledný kód programu je dlouhý a nepřehledný. Nachází se v něm dlouhé úseky programu, které v praxi k řešení daného problému nijak nepřispívají a jen zbytečně program zpomalují.

Mechanismus výjimek umožňuje označit bloky kódu, ve kterých může dojít k potenciálně nebezpečné situaci. Pokud zde dojde k chybě, je zvláštním kódem tato chyba řešena. Obsluha chyby tak probíhá skutečně jen v případě, že k chybě došlo. Hlavní výhodou je zpřehlednění programového kódu, zrychlení programu a značná pružnost takového postupu.

Podívejme se na příklad kódu programu s výjimkou z [1] str. 109.

```
function DivideTwicePlusOne (a, b: integer): integer;
begin
 try        // Začátek chráněného bloku
  Result := a div b;
  Result := Result div b;
  Result := Result + 1;
 except     // Konec chráněného bloku, začátek obsluhy
  on EDivByZero do // Obsluha dělení nulou
   Result := 0;
  on Exception do  // Obsluha jiné chyby
   Result := 6000;
 end;
end;
```

Pokud je tento podprogram spuštěný s parametrem `b` rovným nule, dojde k pokusu o dělení nulou a k vyvolání výjimky. Jejím výsledkem je, že výsledkem tohoto algoritmu bude nula. Pokud by došlo k chybě jiné, bude výsledkem hodnota 6000. Jestliže k vyvolání výjimky nedojde, program skončí s hodnotou $\frac{a}{b^2} + 1$. Uvedený podprogram je samozřejmě jen ilustrační, v praxi by takové řešení bylo nesmyslné.

4.2 Nápověda

Součástí vývoje aplikace je také tvorba její dokumentace. I zde jsou uživatelé zvyklí na určitý standard. Tímto standardem byla dlouho nápověda v souborech s koncovkou **hlp**. Tyto soubory se objevovaly už s prvními verzemi Windows a používají se dodnes. Pro jejich vytváření se používá nejčastěji *Microsoft Help Workshop*. Ten sestavuje nápovědu ze speciálně formátovaných souborů *RTF* (RichText Format).

Modernějším systémem je nápověda ve formě *HTML* souborů. Autoři novějších aplikací používají nejčastěji zkomprimovaný archiv HTML textů v souboru **.CHM**. Tento soubor lze sestavit například v programu *HTML Help Workshop* firmy Microsoft.

Vzhledem k možnostem, které tento formát má, se začíná používat také k elektronickému publikování nejrůznějších textů odborného charakteru (manuály, tutorialy, ...). Pro zobrazování textu však vyžaduje v počítači internetový prohlížeč *MS Internet Explorer*. Borland Delphi ve verzích 6 a 7 však nedisponují nativní podporou tohoto formátu a tu je tak třeba do vyvýjeného programu doinstalovat.

Oba zde uvedené editory souborů nápovědy jsou k dispozici zdarma ze stránek firmy Microsoft. Existují však (mnohdy lepší a pohodlnější) editory jiných výrobců.

4.3 Instalační program

Stejně jako v případě nápovědy, i při instalaci samotného programu, jsou uživatelé zvyklí na pohodlí a přehlednost. Doba, kdy instalace programu probíhala jeho prostým překopírováním, je již dávno pryč (i když v případě „Parametrizovaných úloh“ je to stále možné – aplikace si sama vytvoří chybějící soubory). I mezi instalačními programy existují nepsané standardy. Ty by měly dodržovat komerční i nekomerční generátory instalačních souborů.

Jedním z nekomerčních „instalátorů“ je *Inno Setup* , který lze zdarma získat na adrese `http://www.jrsoftware.org/isinfo.php` . Disponuje prostředky, kterými se vyrovná i komerčním programům tohoto typu.

Kapitola 5

Program „Parametrizované úlohy“

Nyní, když byl čtenář seznámen s tím, co to jsou parametrizované úlohy a s prostředky použitými pro naprogramování stejnojmenné aplikace, můžeme přistoupit k aplikaci samotné. Je samozřejmě vhodné si práci s programem vyzkoušet. Ten si lze opatřit z následujících zdrojů:

- **Příloha této práce** – Instalační soubory tohoto programu se nacházejí na CD nosiči přiloženém k této práci.
- **Internet** – Na začátku roku 2005 byly dokončeny a zveřejněny stránky tohoto projektu na adrese `http://zombie.kap.vslib.cz/~simunkova/pu/` .

5.1 Spuštění programu

Program se spouští poklepáním na ikonu programu, na které je stylizované logo aplikace v podobě písmen *P* a *U*. Tato ikona by měla být umístěna na plochu Windows po jeho instalaci. První, co uživatel uvidí, je uvítací obrazovka (viz. obr. 5.3). Ta indikuje, že se program spouští, načítá potřebné knihovny a připravuje své prostředí. Jakmile je program připraven, obrazovka zmizí a zobrazí se prostředí pro editaci souborů. Inicializace programu nutně selže v případě nesplnění minimálních požadavků. Ty jsou následující:

- **Operační systém:** *Windows 98* a vyšší
- **Místo na disku:** 10 MB
- **Internet Explorer:** verze 5 a vyšší – tento prohlížeč je nezbytný kvůli zobrazování nápovědy a práci s XML soubory

Obrázek 5.1: Úvodní obrázek s logem programu

- **Další programy:** Interpret LaTeXu (pro start programu není nutný, ale program bez něj lze jen těžko prakticky používat)

Program plní tři různé funkce:

- Editace existujících souborů parametrizovaných úloh
- Sestavování písemek
- Vytváření a generování nových souborů parametrizovaných úloh

Než si však popíšeme jednotlivé části programu, je nutné si ujasnit názvosloví:

Soubor parametrizovaných úloh je množina jednotlivých úloh bez ohledu na skutečnou reprezentaci dat, jde víceméně o organizační jednotku

Datový soubor, soubor PU (DUP) je přesně formátovaný soubor uložený na disku obsahující soubor parametrizovaných úloh. PU je koncovka jednoho z použitých formátů (další koncovky jsou TEX a DUP).

5.2 Formáty datových souborů

5.2.1 .TEX

Původním datovým formátem, který byl pro práci s parametrizovanými úlohami v tomto systému používán, byl jednoduše formátovaný textový soubor. Skládal se vlastně ze sekvence připravených, v TeXu formátovaných, úloh. Každá úloha vypadá tak, jak je zachyceno na obrázku 5.2. Výhodou tohoto formátu je jeho snadná sestavitelnost a přenositelnost. Takový soubor lze napsat ručně v libovolném textovém editoru nebo sestavit za pomoci vhodného programovacího jazyka.

```
Na star\'{y}ch stavb\'{a}ch se \v{c}asto uv\'{a}d\'{\i} datum
jejich dokon\v{c}en\'{\i} v \v{r}\'{\i}msk\'{e}m z\'{a}pisu.

Jak\'{y} rok p\v{r}edstavuje n\'{a}sleduj\'{\i}c\'{\i}
z\'{a}pis -- MCII?
---------------------------------
1102
---------------------------------
---------------------------------
```

Obrázek 5.2: Úloha v základním datovém formátu

5.2.2 .PU

Když vývoj projektu dostatečně pokročil, vyžádala si potřeba určitého uživatelského komfortu změnu datového formátu. Konkrétně si změnu vyžádalo zavedení náhledů na obsah datového souboru. Ten je realizován autorem souboru připraveným bitmapovým obrázkem (*.bmp*) se shodným jménem jako má datový soubor. Pokud však došlo ke zkopírování datového souboru, zůstal náhled na původním místě a byl tak nepřístupný.

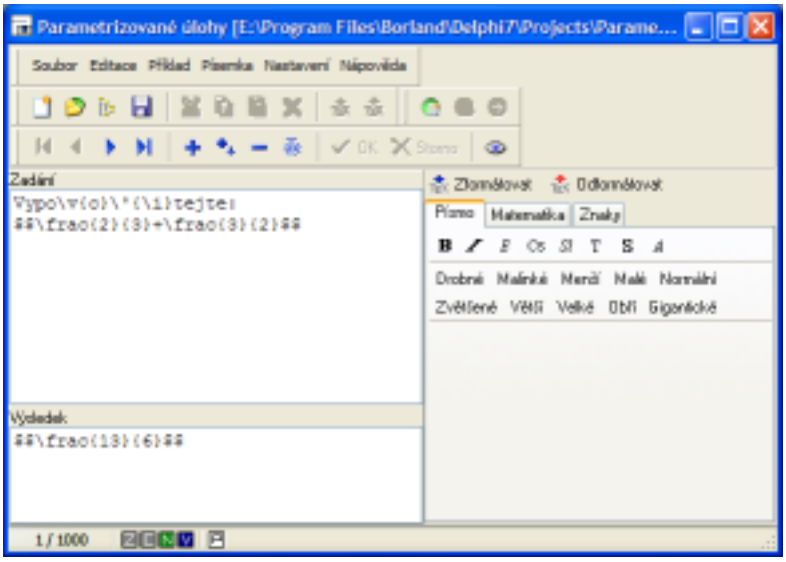
Bylo proto vhodné ukládat soubor parametrizovaných úloh společně s náhledem a informacemi o souboru a autorovi do jediného souboru. Zde se autor aplikace nechal inspirovat kancelářským balíkem *OpenOffice*, jehož datové soubory jsou vlastně systémem souborů zapouzdřeným do archivu typu *ZIP*. Pro vytvoření datového souboru PU je tak použita externí komponenta *ZipForge*.

Uživatel se o struktuře tohoto souboru může snadno přesvědčit, pokud změní jeho koncovku z *.pu* na *.zip*. Pak může v mnoha souborových manažerech soubor otevřít. Najde v něm tři podsoubory. První (*.tex*) je soubor se samotnými úlohami, druhý (*.bmp*) je soubor s náhledem a konečně třetí obsahuje textové informace o souboru a jeho autorovi.

Nyní je čas podívat se na základní funkce programu. Popis to však bude spíše obecný, konkrétní popis jednotlivých prvků programu najde uživatel v nápovědě programu (Hlavní nabídka → Nápověda → Témata nápovědy).

5.3 Editace souboru parametrizovaných úloh

Po spuštění je aktivní právě obrazovka pro tuto činnost. Tato část programu slouží k manuálnímu vytváření souborů parametrizovaných úloh nebo k jejich úpravám. Je zde tedy možné přidávat nové úlohy, mazat nevyhovující, ma-



Obrázek 5.3: Editační (hlavní) formulář programu

nuálně měnit text jednotlivých úloh, vyhledávat a provádět záměny v celém souboru.

Samotné úpravy probíhají „bezpečným“ způsobem. Uživatel se musí nejprve přepnout do editačního režimu, v tom provést požadované změny a když je s výsledkem svých úprav spokojený, musí uložení upravené úlohy potvrdit. Je tomu tak proto, že soubor parametrizovaných úloh může obsahovat až tisíce úloh, a náhodným stiskem kláves provedené změny, se v něm posléze dají vyhledat jen velmi obtížně.

Protože všechny úlohy jsou standardně formátované v LaTeXu, a to není systém, jehož znalost je mezi učiteli běžná, mají uživatelé k dispozici panel na pravé straně tohoto okna. V něm se nacházejí často používané formátovací příkazy tohoto typografického systému. Panel samotný je pro větší přehlednost rozdělen na tři záložky: editaci textu, matematické výrazy a zvláštní znaky.

V nabídce Soubor lze vytvářet nové soubory, otevírat soubory z disku a na disk je také ukládat. Lze odsud rovněž otevřít soubor z připravené stromové struktury, kde mohou být úlohy řazeny podle potřeb učitele. Kromě disku lze se soubory parametrizovaných úloh manipulovat také přes schránku Windows. A lze zde také nastavit a zobrazit informace o souboru a jeho autorovi.

Obsah nabídky Editace se zaměřuje na dvě věci. První je stejně jako v jiných programech práce s textem a schránkou – kopírování, vkládání, mazání,

ale také automatické zformátování „obyčejného“ textu do textu připraveného pro sazbu v TEXu. Tento převod však funguje jen pro jinak nezformátovaný text, nelze jej tedy použít pro převod vzorců a speciálních znaků. Druhou skupinou funkcí je vyhledávání a nahrazování v celém souboru úloh.

Nabídka Příklad umožňuje úpravy aktuálního souboru. V souboru úloh se lze pohybovat pomocí příkazů První, Předchozí, Další a Poslední. Aktuální úlohu lze zobrazit v připravené šabloně pomocí funkce (Náhled). Přepnout se do editačního režimu lze přes Přidat, Vložit a Upravit, ukončit jej lze potvrzením (OK) nebo odmítnutím (Storno) změn. Nevyhovující úlohy lze samozřejmě Smazat.

K popisu zbývají dvě nabídky. V Nastavení lze modifikovat chování programu, v nabídce Nápověda se samozřejmě nachází přístup k souboru nápovědy. V té se nachází podrobnější popis funkcí programu, jeho funkcí a možností. Dále je zde však přímý přístup k „matematickému jádru“ sloužícímu k vyhodnocování hodnoty aritmetických výrazů. A nakonec zde jsou informace o programu a přístup k internetovým stránkám projektu.

5.4 Sestavování písemných prací

I k sestavování výsledných písemek je přístup z hlavního okna aplikace. Samozřejmě takto sestavená zadání se nemusí využívat jen jako písemky nebo testy, ale také například jako zadání domácích úkolů. Pro zjednodušení však pro všechna zadání práce generovaná tímto programem, budeme používat označení písemka – koneckonců, vždy jde o písemnou práci.

Všechny příkazy pro vytváření písemek se nacházejí v hlavní nabídce pod názvem Písemka. Samotná cesta vedoucí k vytištění písemky sestává ze tří kroků:

1. Připojení souborů s příklady
2. Nastavení vzhledu
3. Sestavení výsledného dokumentu

5.4.1 Soubory s úlohami

Nejprve je potřeba určit soubory, ze kterých se mají vkládat úlohy do námi připravované písemky. K tomu spustíme v nabídce Písemka položku s názvem Přidat příklady. Největší část okna zabírá seznam souborů parametrizovaných úloh, které budou do písemky zahrnuty.

Do seznamu lze soubory přidávat několika způsoby. Prvním je přidání ze souboru uloženého na disku prostřednictvím běžného dialogu „Otevřít“. Takto lze načíst libovolný soubor ve známém formátu.

Druhou možností je vložení ze stromové struktury, stejně jako v hlavním formuláři programu. Ve skutečnosti je skutečně otevřen stejný dialog. Takto lze otevřít jen soubory, které do této struktury byly zařazeny. Složky, ve kterých se takto vložitelné soubory nacházejí, lze nastavit v nabídce Nastavení → Prostředí hlavního okna.

Konečně třetí možností je zkopírování souboru, který má uživatel již připravený v hlavním okně programu. Skutečně zde jde o kopii – soubor je v okamžiku stisknutí tlačítka Paměť překopírován do datových struktur, ze kterých je písemka posléze připravena, tak, jak je. Pokud uživatel posléze v hlavním okně provede nějaké změny, tyto úpravy se v již připraveném souboru neprojeví.

Vzhledem k tomu, že je možné tento dialog opustit, aniž by došlo ke ztrátě dat, musí existovat možnost jak celou datovou strukturu vymazat – a tím je příkaz Vymazat. Jinak stačí vymazat jednotlivé nechtěné položky seznamu.

Protože do písemky může být vloženo více příkladů z jednoho souboru, nacházejí se při horním okraji formuláře tlačítka pro zvýšení a snížení počtu vybíraných příkladů. Vzhledem k tomu, že příklady budou do písemky zařazovány přesně v pořadí, v jakém jsou uvedeny na formuláři, nechybí ani tlačítka pro změnu jejich pořadí. Systém také umožňuje jednotlivé příklady obodovat. Jednotlivým typům příkladů lze přidělit body v rozsahu 0 - 10.

Na pravé straně formuláře jsou ukazatele současného stavu připravované písemky. V prvé řadě se zde zobrazuje počet souborů parametrizovaných úloh, který pro písemku bude použit. Za druhé je zde ukazatel celkového počtu příkladů. Protože však samotné zobrazování písemky realizuje externí program (TEX) aplikace „Parametrizované úlohy“, nemá žádnou možnost, jak zjistit, zda na výslednou písemku stále stačí jedna stránka. Aby obsah zadání „nepřetekl“ na více stránek, je rozumné nevkládat do písemky víc než 10 zadání. Konečně posledním ukazatelem na pravé straně formuláře je celkový počet bodů, které za písemku může student získat.

Jakmile je uživatel spokojený s obsahem připravované písemky, může stisknutím tlačítka OK formulář uzavřít a vytvořit datové struktury, ze kterých bude písemka vytvořena. Pokud by uživatel nyní chtěl vytvářet další písemku, musí tyto struktury zrušit – to lze položkou nabídky Písemka → Nová. Pokud uživatel tyto struktury nezruší a jen nechá vytvořit novou písemku výše popsaným způsobem, budou nové datové struktur přidány za ty již existující a výsledný soubor zadání tak bude poněkud heterogenní (což nemusí být vždy na škodu).

Celkový obsah vytvořených datových struktur lze z nabídky Písemka uklá-

dat a otevírat. Práci na písemce tak lze přerušit a znovu se k ní vrátit. Lze také takovou surovou formu písemky uložit pro použití v dalších letech.

5.4.2 Nastavení vzhledu

Obsah písemky je již tedy připravený. Co však lze ještě změnit je forma jakou bude práce předkládána. Pokud uživatel nechce nic změnit, budou použita implicitní nastavení. Změnit lze několik prvků:

Celý vzhled vytištěné písemky je definován šablonou – souborem obsahujícím místo skutečných prvků zástupné symboly. Při vytváření písemky pak jsou příklady a určité textové řetězce dosazeny na určená místa.

Jedním z takových řetězců je jméno žáka. Jméno každého žáka je získáno z textového souboru, kde každý řádek reprezentuje jednoho žáka.

Do písemky lze také vložit doprovodný text, ve kterém lze určit například datum odevzdání takto zadaného domácího úkolu.

Číslovat jednotlivá zadání lze několika způsoby. Může jít o arabské číslice (čísla jsou zarovnávána na stejný počet znaků, aby nebylo možné je měnit), lze použít římské číslice, hexadecimální číselnou soustavu a konečně klasické označení písmeny.

Konečně lze změnit nadpis písemky, text vkládaný před bodové hodnocení a bodové hodnocení samotné. Lze také sestavit titulní stránku obsahující některé informace o souboru.

5.4.3 Výsledný dokument

Nyní již zbývá jen vytvořit prostřednictvím LaTeXu výsledný soubor. K tomu je třeba určit jméno výsledného souboru a jeho formát. Ne všechny interprety TeXu umí všechny formáty (PostScript, PDF). Lze zde také určit, zda mají ve výsledném souboru být zadání bez výsledků a zadání s výsledky a počet kolik z připravených zadání bude do souboru vloženo.

5.5 Vytváření nových souborů parametrizovaných úloh

Nové soubory parametrizovaných úloh lze sestavit mnoha způsoby. Lze například ručně napsat celý soubor ve formátu ukázaném na obrázku 5.2. Je také možné soubor vytvořit prostřednictvím některého programovacího jazyka a nebo lze použít editační režim zde popisovaného programu. Tento program však také obsahuje prostředky, které vytváření nových souborů v určitých situacích usnadňují.

5.5.1 Testové otázky

Často se v praxi při prověřování znalostí žáků používají testy s předem známou množinou odpovědí. Zkoušeným v takovém případě stačí zatrhnout jednu správnou odpověď (nebo i více). Také zde popisovaný program umožňuje takové úlohy vytvářet.

Nástroj pro jejich sestavování najde uživatel v hlavní nabídce Soubor → Textové testové otázky. Zadáním takto vytvářené úlohy je otázka a uzavřená množina odpovědí, výsledkem jsou správné odpovědi. Formulář pro vytváření těchto otázek je rozdělen na několik záložek.

První z těchto záložek má nadpis „Otázka“. Zde učitel musí definovat otázku, na kterou má student odpovídat. Tuto otázku musí správně zformátovat pro TEX. Tuto činnost mu může opět usnadnit boční „Panel pro práci s TEXem“.

Další záložkou jsou „Možnosti“. Zde se definují možné odpovědi na položenou otázku a také jejich správnost. Každým stisknutím tlačítka Nová vstoupí uživatel do okna, kde má možnost přidat novou potenciální odpověď a zformátovat ji. Po jejím vložení lze zatrhnutím políčka vedle této odpovědi určit, zda jde o odpověď správnou nebo nesprávnou. Vytvořené odpovědi lze samozřejmě upravovat a mazat.

Konečně na poslední záložce – „Nastavení“ – uživatel specifikuje kolik ze správných a nesprávných odpovědí má být vloženo do každé verze otázky a kolik možností tak bude při odpovídání žák mít. Vytvořené úlohy jsou přidány do souboru úloh hlavního formuláře programu – pokud uživatel chce, aby ve vzniklém souboru byly pouze nové úlohy, může jej vymazat zaškrtnutím příslušného políčka na této záložce. Program se snaží vytvořit co možná největší soubor úloh, proto se nové úlohy vytvářejí ze všech kombinací zadání. Je-li s_{max} celkový počet správných odpovědí, s počet požadovaných správných odpovědí, n_{max} a n je to samé pro nesprávné odpovědi a o je počet nových otázek, pak

$$o = \binom{s_{max}}{s} \cdot \binom{n_{max}}{n} = \frac{s_{max}!}{s! \cdot (s_{max} - s)!} \cdot \frac{n_{max}!}{n! \cdot (n_{max} - n)!}$$

5.5.2 Jednoduché matematické úlohy

Zajímavější než vytváření testových otázek je zpracovávání otázek, ve kterých je nutné něco vypočítat. Tento program obsahuje jednoduchý nástroj pro vyhodnocování „základoškolských“ aritmetických výrazů. Mechanismus pro vytváření takových úloh se v našem programu spouští z nabídky Soubor → Jednoduchá matematika. I zde je formulář rozdělen záložkami na několik částí.

První část je věnována **parametrům**. Právě to jsou hodnoty dosazované do algoritmů, díky těmto parametrům je zajištěna rozdílnost zadání úloh – počet parametrů (a počet hodnot za ně dosazovaných) také určuje kolik úloh bude vytvořeno. Do množin hodnot parametrů lze vkládat souvislé intervaly hodnot nebo lze vyjmenovat prvky množiny, která se má použít. Program tvoří tolik variant zadání, kolik je možných kombinací vstupních parametrů – tedy součin všech počtů hodnot za parametry dosazovaných. Pro program není problém vytvořit soubor o víc než milionu zadání – autor souborů by však neměl vytvářet žádné obsahující víc než 10000 zadání.

Druhá záložka umožňuje definovat algoritmus zpracování příkladu. Takto lze nastavit hodnoty proměnných, které budou dál použity buď v dalších krocích, nebo přímo při formátování výsledku. Prostředky, které tento průvodce pro zpracování algoritmů poskytuje u6ivateli, jsou velmi jednoduché a přesto postačující. Algoritmus se může skládat z libovolného počtu řádků. Na většině řádků je přitom do proměnné přiřazena hodnota určená algebraickým výrazem. Jazyk pro zápis algoritmu má kromě prostředků pro vyhodnocení výrazů jen dvě zvláštní konstrukce. První z nich automaticky upravuje zlomek na jeho základní tvar (vydělí čitatel i jmenovatel jejich největším společným dělitelem). Druhá umožňuje předčasný konec zpracovávání této varianty zadání. Tato druhá „direktiva“ se používá pro zajištění srovnatelné obtížnosti všech zadání v souboru. Učitel tak může zabránit vytváření příliš triviálních zadání.

Na rozdíl od programovacího jazyka Pascal zde není nutné předem deklarovat proměnné. Program totiž používá jediný datový typ – a to „reálná čísla“ (nejde o reálná čísla v matematickém slova smyslu). Protože při vyhodnocování ukončovací podmínky (a zpracování logických výrazů) potřebujeme jednoznačně určit pravdivost a nepravdivost výrazu, zavedeme následující: za nepravdivý budeme považovat výraz, jehož hodnota je nulová a všechny ostatní výrazy jsou pravdivé.

Kromě jejich vypočítání je nutné hodnoty proměnných také vhodně prezentovat. To lze ze třetí záložky, nazvané „Formát výstupu“. Zde musí autor připravit text zadání a výsledku, zformátovat je pro zpracování systémem TEX a na příslušná místa doplnit sekvenci „`<#`*proměnná volitelné parametry* `#>`“, na jejíž místo bude doplněna výsledná hodnota proměnné formátovaná dle přání autora. Ten má k dispozici následující možnosti:

- **Standardní formát** – Za běžný formát bez nutnosti použít zvláštní parametry je považovaný zápis čísla v desítkové soustavě se dvěma desetinnými místy.
- **Římská soustava** – Takto lze formátovat přirozená čísla od 1 do 3999. Použitý je nejobvyklejší formát těchto čísel – tedy z pozdního období

Říma. Protože vyšší čísla se vyjadřovala za pomocí čáry nad příslušnou skupinou číslic a tyto znaky se mezi standardními znaky nevyskytují, jsou čísla takto vyjádřitelná omezená číslem 3000. Za vstup pro formátovací algoritmus je považována celá část čísla.

- **Poziční číselná soustava** – Výstupem může být také číslo zapsané v soustavě o základu 2 až 30. Pro soustavy o vyšším základu chybí standardní znaky pro číslice. Je možné určit počet číslic vypisovaných za „desetinnou čárkou“.

- **Text** – Celou část čísla lze použít také jako index do pole textových řetězců. Tak lze realizovat například slovní vyjádření čísla. Textové řetězce se nastavují na čtvrté záložce formuláře.

Po vyplnění všech dat potřebných ke zpracování zadání a stisku tlačítka OK jsou nová zadání vložena do paměti hlavního okna programu.

Pokud došlo při zpracovávání některé varianty úlohy k chybě (a je generována výjimka), je tato varianta systémem odmítnuta a na závěr zpracovávání je zobrazeno okno, které všechny chyby shrnuje. Program si takto musí poradit s následujícími chybami:

- Matematické chyby (dělení nulou)
- Chyby formátování
- Odmítnutí příkladu z důvodu splnění dané podmínky
- Nesprávné formátování aritmetického výrazu
- Jiné chyby

Kapitola 6

Problémy

Je na čase seznámit čtenáře s anatomií aplikace a naznačit principy, na kterých jsou její jednotlivé části postavené. Nebudeme zde však přímo citovat z příslušných sekcí programu, zdrojové kódy jsou přílohou této práce a zvídavý čtenář si je může sám najít. První dva problémy mají souvislost s vytvářením matematických zadání v prostředí jednoduché matematiky, třetí se týká sestavování písemek.

6.1 Vyhodnocování výrazů

Součástí programu je také systém pro vytváření jednoduchých matematických úloh určených na základní a střední školy. Tento systém ke svému chodu vyžaduje mechanismus který z formátovaného textového řetězce získá hodnotu aritmetického výrazu v tomto řetězci zapsaného.

V podstatě jde tedy o funkci, jejímž vstupním parametrem je řetězec a výstupní hodnotou je reálné číslo s požadovanou přesností. V našem případě bude použitým datovým typem ten nejpřesnější dostupný v Delphi – `extended`. Výrobce uvádí rozsah $\pm(3.6 \cdot 10^{-4951} \ldots 1.1 \cdot 10^{4932})$, při dvaceti platných číslicích. Takový datový typ samozřejmě obsadí mnoho místa v paměti – celých 10 Bajtů. Při základoškolské a středoškolské matematice téměř jistě nenarazíme na žádné hranice tohoto datového typu. Nicméně pro pořádek uveďme, že přestože se tento datový typ označuje jako „real (numeric) type“, nejde o reálná čísla, ale jen o datový typ schopný jejich (značně omezené) reprezentace. Na rozdíl od *Turingova stroje* disponují skutečné počítače pouze omezeným množstvím paměti a proto nejsou schopné reprezentovat žádnou nekonečnou množinu – tedy ani čísel přirozených a racionálních, natož reálných.

Pro absolventa základní školy by neměl být problém přiřadit napsanému

aritmetickému výrazu jeho hodnotu. Takovému žákovi stačí se na výraz podívat, vyhodnotit prioritu operátorů a tak zjistit, které části výrazu již je schopný vyhodnotit a tuto činnost realizovat tak dlouho, dokud není celý výraz zpracovaný a žák nemůže na jeho konec za rovnítko napsat výsledek a dvakrát jej podtrhnout. To, co je pro žáka základní školy (sice ne vždy jednoduchý) zvládnutelný úkon, je pro počítač (a hlavně jeho programátora) problém. Vlastně problémy dva: identifikovat jednotlivé části výrazu a vyhodnotit je ve správném pořadí.

Podívejme se nejprve na tvar výrazu, který má být vyhodnocen. Za aritmetický výraz tedy budeme považovat:

1. konstantu (například `3,14`; π)
2. název proměnné (například `a`)
3. výraz v závorce (`(`*`výraz`*`)`)
4. název funkce, následovaný jejími argumenty uzavřenými v závorkách (`funkce(`*`výraz`*`)`)
5. výrazy spojené pomocí operátorů (*výraz operátor výraz*)

Žádné jiné textové řetězce již za aritmetické výrazy považovány nebudou. Pokus o jejich vyhodnocení musí skončit chybou – k tomuto účelu využijeme mechanismus výjimek z prostředí Delphi.

Pokud bude celý mechanismus vyhodnocování výrazů obsažen ve funkci `Vyraz`, pak při vyhodnocování všech takto definovaných výrazů použijeme tuto funkci rekurentně na obsah všech částí popsaných jako *výraz*. Tím zbývá jen vyhodnocení operátorů. Zde by mohlo stačit při čtení výrazu střídat funkce pro vyhodnocení výrazu a identifikaci operátoru. Mohlo by to stačit v případě, že bychom nepožadovali vyhodnocování operátorů dle běžných priorit (mocnina, součin, ...).

Bohužel je právě toto požadováno. Existují však programovací jazyky, které priority operátorů nezohledňují nebo používají jiný způsob zápisu výrazu (prefixový, postfixový – „Reverzní polská notace“). I přes zajímavost takového formátu se zde budeme zabývat vyhodnocováním tradičního způsobu zápisu aritmetických výrazů.

Žák při vyhodnocování výrazu postupuje tak, jak byl naučen ve škole: podívá se na výraz, určí, které jeho části lze vyhodnotit a sestaví nový – jednodušší – výraz s již vyhodnocenými částmi. Takto postupuje, dokud nezíská požadovanou hodnotu. Žák ovšem netuší, že vlastně přetvářel řádkový výraz do podoby stromové struktury a tu poté vyhodnocoval od konců k vrcholu. Počítač bude postupovat podobně. Avšak místo toho, aby předem

sestavoval celý strom výrazu (což by bylo poměrně neefektivní), bude zatím nepoužitelné části stromu ukládat do tzv. zásobníku. Zásobník je LIFO (Last-In-First-Out) datová struktura, fungující stejně jako zásobník pistole, nebo zásuvka s papíry. Jednoduše jej lze popsat tak, že z ní lze vyjmout pouze poslední vložený předmět. Použití zásobníku je při práci s počítači běžné a pokud bychom vyhodnocovali výraz zapsaný v již zmiňované „Polské notaci“, použili bychom zásobník - a to jediný. My však budeme potřebovat zásobníky dva: jeden na čísla (ZČ) a druhý na operace (ZO). Protože vždy obsahuje výraz o jedna méně operátorů, než výrazů a s tím by se nám pracovalo nepohodlně, přidáme na konec námi vyhodnocovaného výrazu speciální operátor (@), čímž získáme uspořádanou množinu uspořádaných dvojic [výraz, operátor].

Námi hledaný algoritmus lze zapsat například takto:

1. Uložíme hodnotu výrazu na zásobník čísel (ZC).
2. Pokud je zásobník operátorů (ZO) prázdný uložíme na něj aktuální operátor a pokračujeme další dvojicí (krokem 1).
3. Je-li na vrcholu zásobníku ZO operátor s vyšší (nebo stejnou) prioritou, než má operátor aktuální, vyjmeme jej a provedeme příslušnou operaci pro poslední dvě hodnoty ze zásobníku čísel. Výsledek uložíme do zásobníku ZČ. Krok 3 opakujeme dokud není na vrcholu ZO operátor s nižší prioritou.
4. Operátor uložíme do ZO a pokračujeme další dvojicí (krokem 1).

Podívejme se nyní na příkladu, jak tento algoritmus funguje. Vyhodnotíme výraz $2 + 3 \cdot 4 - 3$. Nejprve tento výraz zapíšeme tak, abychom jej mohli zpracovat počítačem: `2 + 3 * 4 - 3 @`.

1. První dvojicí je [2,+]. Hodnotu 2 uložíme na zásobník čísel. Protože je zásobník operátorů prázdný, uložíme na něj +.
   ZČ: 2
   ZO: +
2. Ani s druhou dvojicí [3,*] nebudeme nakládat jinak. Číslo 3 uložíme na zásobník a protože operátor +, který je na vrcholu zásobníku ZO má nižší prioritu než násobení (*), uložíme také aktuální operátor na zásobník.
   ZČ: 3, 2
   ZO: *, +

3. Zajímavá je dvojice [4,-]. Číslo opět uložíme na ZČ. Na vrcholu ZO však najdeme operátor „*“, který má vyšší prioritu, než „-“. Vyzvedneme tedy ze zásobníku dvě hodnoty pod vrcholem a vynásobíme je: $4 \cdot 3 = 12$. Výslednou hodnotu uložíme na zásobník.
   ZČ: 12, 2
   ZO: +
   Tím však tento krok ještě nekončí. Operátor „-“ má stejnou prioritu jako operátor „+“. Proto vybereme další dvě položky ze zásobníku čísel a sečteme je: $12 + 2 = 14$. Tuto hodnotu uložíme do ZČ. A konečně můžeme uložit také operátor „-“.
   ZČ: 14
   ZO: -

4. Poslední dvojicí je [3,@]. Číslo 3 uložíme (ZČ) a protože jakýkoli operátor má vyšší prioritu než ten ukončovací, provedeme rozdíl mezi posledními dvěma hodnotami na ZČ: $14 - 3 = 11$. Výsledek opět uložíme. Zásobník operátorů je prázdný, uložíme na něj tedy operátor „@“.
   ZČ: 11
   ZO: @

5. Protože již nemáme žádnou dvojici ke zpracování, na zásobníku čísel máme jedinou hodnotu a na zásobníku operátorů jen uzavírací operátor, můžeme číslo v ZČ prohlásit za hodnotu výrazu a tedy $2+3 \cdot 4-3 = 11$, což je pravda.

Nyní zbývá ještě vypracovat konkrétní implementaci tohoto algoritmu pro náš program. Případní zájemci o toto zpracování nechť si prohlédnou knihovnu `simplemath.pas` uloženou mezi zdrojovými kódy aplikace na CD. Chybějící matematické funkce (faktoriál, nejmenší společný násobek, největší společný dělitel, a jiné) jsou zapsány v souboru `mistmath.pas`.

6.1.1 Závorky

Přestože je v matematice zvykem používat několik různých závorek pro různé úrovně jejich zanoření, v této implementaci si vystačíme jen se závorkou kulatou. Různé druhy závorek sice zvětšují přehlednost výrazu, při práci s počítačem by celou tuto záležitost pouze komplikovaly.

Pokud tedy interpret výrazů narazí tam, kde očekává výraz, na otevírací závorku, najde k ní odpovídající závorku uzavírací a textový řetězec mezi nimi prohlásí za výraz. Na tento textový řetězec je zavolána funkce pro získání jeho hodnoty. Tento postup se nazývá rekurze a jde o velmi elegantní způsob řešení některých problémů. Sama tato metoda však může být

poměrně nebezpečná, protože při každém volání procedury je spotřebována část paměti pro „režii“ programu a tak může dojít k vyčerpání veškeré paměti operačního systému a tím ke zhroucení programu. Okamžik, kdy k této nemilé situaci dojde, závisí na množství paměti, velikosti uchovávaných dat a počtu zanořených funkcí. V tomto případě však je tato hrozba zanedbatelná vzhledem k povaze zpracovávaného problému – nikdy nedojde k dostatečně velkému počtu volání funkce.

6.1.2 Funkce

Řeč samozřejmě bude o funkcích matematických dle definice výrazu uvedené výše – tedy o posloupnosti „funkce(argumenty)“. Kompletní seznam funkcí je uvedený v nápovědě programu. Vezmeme-li však v úvahu použitý datový typ, mohla by uživatele překvapit přítomnost funkcí jako je faktoriál nebo jiné funkce definované na celých (přirozených) číslech. Je tedy potřeba popsat jak program postupuje při zpracovávání reálného argumentu takovou funkcí. Řešení je samozřejmě jednoduché – bude nutné hodnotu zaokrouhlit. To lze dvěma způsoby:

- Běžné zaokrouhlení je v Delphi realizováno funkcí `round`. Jejím výstupem je celé číslo ekvivalentní s argumentem zaokrouhleným na nejbližší celé číslo. Hodnota přesně mezi oběma celými čísly je zaokrouhlena na sudé číslo – jde tedy o tzv. bankéřův algoritmus.
- Lze také vzít v úvahu jen celou část argumentu pomocí funkce `int` (je nutné ji volat jako `round(int())`). Takto zaokrouhlujeme vždy směrem k nule.
- Konečně lze celý argument, který nemá ekvivalent v požadované číselné struktuře prohlásit za chybu.

Nakonec byla vybrána první možnost, protože tento způsob zaokrouhlování je standardní a předvídatelný. Je však dobré vědět, že v tomto systému je $2,7! = 3! = 6$. Program se tak vlastně pokouší (jednoduchou formou) korigovat chyby uživatele a dosadit hodnotu, kterou patrně uživatel chtěl použít.

6.2 Číselné soustavy

Jak jsme si popsali v předchozí kapitole, je v tomto programu oddělena hodnota proměnné a její reprezentace. Jednoduše řečeno, lze hodnotu proměnné vyjádřit v mnoha číselných soustavách. Proto by v této práci neměl chybět

popis toho, co to vlastně číselné soustavy jsou. Na číselné soustavy se budeme dívat jako na způsob reprezentace čísel. V dějinách matematiky bylo používáno mnoho takových způsobů a několik se jich používá i nyní.

Ignorujme pro účely tohoto textu bohatou historii číselných soustav a soustřeďme se na její realizovanou, konkrétní podobu v tomto programu. Pro větší přehlednost si však téma rozdělíme tak, jak je obvyklé: na soustavy nepoziční a poziční.

6.2.1 Nepoziční číselné soustavy

Za historicky starší jsou považovány soustavy nepoziční. Nejznámější takovou soustavou je systém římský [14]. Během dlouhé existence římské říše se tento systém vyvíjel a v jeho vývoji se pokračovalo i ve středověku – proto existuje v mnoha podobách. Základem té nejznámější jsou číslice zapsané v tabulce 6.2.1. Čísla se tvoří sčítáním a řetězením symbolů – (3 = III), (8 = 5 + 3 = VIII), kde řetězením symbolů vznikají symboly vyššího řádu (V + V = X). Takto jednoduše vypadal římský systém ještě v období republiky. I z tohoto období nám zůstal relikt – na některých starých hodinách je číslo 4 zapsáno jako IIII a 9 jako VIIII. Pozdějším vývojem (snad pod Etruským vlivem) však bylo do systému zavedeno kromě sčítání také odečítání symbolů o řád nižších. Od té doby se číslo 4 značí IV a 9 jako IX.

Právě to je stav implementovaný v algoritmu ze souboru `mistmath.pas`. Tím lze bez problémů vyjádřit čísla mezi 1 a 3999 (I a IMMMM), což je také ta část římské matematiky, která se používá i dnes. Dnes totiž už najdeme takto formátovaná čísla jen v historických textech, na budovách, při číslování stran v knihách a při uvádění roků (například rok vydání filmů).

Vyšší čísla lze vyjádřit pomocí čáry nad danou číslovkou (nebo jejich sekvencí), tím vyjádříme násobek tisíci. Takto se $\overline{\text{V}} = 5000$, $\overline{\text{X}} = 10000$, $\overline{\text{L}} = 50000$, $\overline{\text{C}} = 100000$, $\overline{\text{D}} = 500000$ a $\overline{\text{M}} = 1000000$. Ve středověku pak

Číslice	**Název**	**Hodnota**	**Význam**
I	unus	1	Jeden prst
V	quinque	5	Pět prstů, dlaň
X	decem	10	Dvě ruce
L	quinquaginta	50	Pochází patrně od Etrusků
C	centum	100	První písmeno římského slova pro sto
D	quingenti	500	Pochází patrně od Etrusků
M	mille	1000	První písmeno římského slova pro tisíc

Tabulka 6.1: Základní číslice římské soustavy

údajně byly do systému zaneseny symboly S pro 7 (nebo 70), O jako 11, F pro 40, A pro 50 (nebo 500), R pro 80, N = 90, Y = 150, T = 160, H = 200, E = 250, K = 250, B = 300, G = 400, P = 400, Q = 500 a Z = 2000. Jejich použití je však natolik nestandardní a dokonce sporné, že většinou nejsou nikde ani zmiňovány.

6.2.2 Poziční číselné soustavy

Obrovským přínosem pro matematiku bylo objevení pozičních číselných soustav. Tohoto objevu ovšem patrně dosáhly pouhé dvě kultury: mayové a indové. Klíčovou podmínkou pro tento objev bylo totiž vynalezení nuly. Ale i po tomto objevu ještě zbývá k pozičním číselným soustavám dlouhá cesta. V těchto soustavách je hodnota číslice určena její pozicí v čísle.

V poziční číselné soustavě lze každé kladné reálné číslo vyjádřit takto:

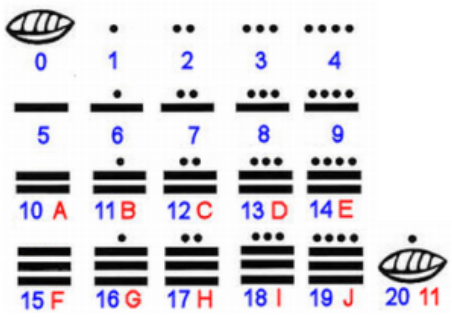
$$a = (a_n a_{n-1} \ldots a_3 a_2 a_1 a_0,\ a_{-1} a_{-2} \ldots)_z =$$

$$= a_n z^n + a_{n-1} z^{n-1} + \ldots + a_3 z^3 + a_2 z^2 + a_1 z + a_0 + a_{-1} z^{-1} + a_{-2} z^{-2} + \ldots$$

kde z je přirozené číslo větší než jedna nazývané **základ soustavy** a všechna a_i jsou přirozená čísla z intervalu $\langle 0; z)$. Záporná čísla zapisujeme podobně, jen se znaménkem „-“ před číslem.

Každá z výše zmíněných kultur používala jiný základ své soustavy. Protože soustavu indickou, do Evropy přinesenou prostřednictvím Arabů a jako „arabskou“ také známou, dnes používáme a dobře víme, že jejím základem je číslo deset, podíváme se na systém vytvořený starými Mayi.

Bohužel toho o mayské kultuře nevíme tolik, kolik by si patrně zasloužila. Pokud procházíme třeba i jen volně přístupné materiály na internetu týkající se jejich matematiky [16], zjistíme, že si zdroje sice přímo neodporují, ale v některých se objevují prvky nepopsané nikde jinde – například druhá číselná soustava, nebo nepravidelnost druhého řádu čísel (kde se údajně používaly jen číslovky 1 - 13). Jaké informace o mayich jsou tedy těmto zdrojům společné? Základem jejich číselného systému bylo číslo 20. To samozřejmě znamená, že potřebovali 20 číslic (symbolů pro vyjádření čísel). Tyto číslice jsou na obrázku 6.1. Jednotlivé číslice čísla se zapisovaly pod sebe. Při výuce číselných soustav jsou obvykle zmiňovány „počítačové“ soustavy – soustavy o základech 2, 8 a 16 – s tím, že z historických soustav nám zůstala slova jako pár anebo tucet a že pozůstatky soustav o jiných základech lze nalézt v kalendářích a na hodinkách. Mayská soustava může být při výuce poměrně atraktivní svou jednoduchostí a exotičností. Bohužel však bez zásahu do samotného systému TEX nelze přímo tisknout mayské číslice, lze však zavést jejich substituci dnes používanými znaky, jak je naznačeno na obrázku 6.1.



Obrázek 6.1: Mayské číslice a jejich reprezentace ve dvacítkové soustavě

Program „Parametrizované úlohy“ se při práci s číselnými soustavami neomezuje pouze na zde popsané soustavy – dokáže zobrazovat čísla v číselných soustavách o základech 2 - 36. To jsou totiž soustavy vyjádřitelné prostřednictvím číslic a znaků velké standardní abecedy (ASCII kód). Většinou takové algoritmy převádějí pouze čísla přirozená (či celá). Protože matematický interpret tohoto programu disponuje datovým typem s desetinnou čárkou, je třeba zobrazit jakékoli číslo, které lze tímto typem vyjádřit. Příslušný algoritmus to samozřejmě umí a je opět v knihovně `mistmath.pas`.

6.3 Náhodné pořadí příkladů

Pokud sestavujeme písemku, je žádoucí, aby příklady za sebou nenásledovaly ve stejném pořadí, v jakém jsou uložené v souboru. Příklady v souboru jsou totiž většinou seřazené tak, jak byly vytvořeny – to znamená, že dva po sobě následující příklady se většinou liší jen v jednom parametru. Žákovi by tak stačilo podívat se ke svému sousedovi, kde by nalezl zadání velmi podobné tomu, které by držel v ruce. Tím, že zadání patřičně „promícháme“, dosáhneme jejich vyšší variabilty (uvědomme si, že například vybíráme 30 zadání ze souboru 1000). Tento problém se stane ještě lépe viditelným, když si představíme, že v každé písemce může být z daného souboru příkladů víc.

I zde je řešení jednoduché. Máme uspořádanou množinu příkladů a výsledkem nějakých operací má být tatáž množina, ve které se jen změní pořadí prvků – jde tedy o prosté zobrazení množiny sama na sebe a to nazýváme permutací. Ve skutečnosti však na množině n zadání provedeme n transpozic. Během nich procházíme postupně množinu zadání a prvek na této pozici zaměníme s prvkem na pozici dané funkcí `random(n)` – tedy (pseudo)náhodné číslo v rozsahu $\langle 0; n)$. Složením těchto transpozic vznikne výsledná permutovaná množina příkladů.

Pro situace, kdy je zapotřebí méně příkladů, než je jich v příslušném souboru, je vše v pořádku. To ovšem neplatí pro opačný případ.

Představme si nyní následující situaci: Máme soubor s deseti příklady, chceme ale patnáct zadání, každé s dvojicí těchto příkladů (potřebujeme tedy třicet různých zadání). Pokud bychom prostě jen zařadili stejnou uspořádanou množinu příkladů třikrát za sebe, stane se jedna poměrně nepříjemná věc: šestý žák dostane stejná zadání, jako žák první. Čtení ze souboru příkladů je ve skutečni realizováno pomocí ukazatele, který se po každém čtení posune na další příklad, pokud dosáhne konce souboru, přesune se na začátek. Abychom se tohoto malého problému zbavili, musíme pokaždé, když ukazatel přechází z konce souboru na jeho začátek, celý tento soubor opětovně permutovat.

I když by se nyní zdálo, že jsme tímto postupem odstranili všechny díry v tomto algoritmu, není to pravda. Ve skutečnosti jsme společně s odstraněním této chyby zanesli do algoritmu jinou možnost nežádoucího chování. Vznikla na konci souboru úloh. Pokud totiž bude počet všech požadovaných příkladů nesoudělný s počtem příkladů vkládaných do jedné varianty zadání (to vše stále v rámci jednoho souboru parametrizovaných úloh) dojde při zde popsaném postupu k situaci, kdy vložíme do připravovaného zadání nějaký počet příkladů, ukazatel se přesune na začátek souboru, proběhne permutace tohoto souboru a do zadání jsou doplněny zbývající příklady. Nikde však není zajištěno, že se příklad, který byl původně na konci souboru, po „zpřeházení“ souboru neocitne na jeho začátku a my jej tak do téhož zadání nevložíme po druhé. To by žákovi, který by toto zadání dostal, poměrně usnadnilo práci. K jeho velkému zklamání je však snadné vzniku této situace zabránit.

Stačí totiž, aby byly počty vkládaných (p) a všech požadovaných příkladů (n) čísla soudělná. Mohli bychom například ze souboru odstranit několik příkladů. Ještě lepší však bude pokud pouze nastavíme zarážku ukazatele tak, aby místo na konec souboru ukazovala na příklad s indexem rovným nejvýššímu násobku čísla p, menšímu než n.

Kapitola 7

Závěrem

Protože se již čtenář seznámil s vytvořeným programem a dokonce nahlédl do jeho nitra, je načase shrnout zkušenosti z jeho testování, veřejných prezentací a na závěr popsat výhody a nevýhody tohoto systému.

7.1 Praktické použití

Samotná koncepce parametrizovaných úloh se jen na liberecké univerzitě používá víc než 20 let. Během této doby se systém na úrovni vysoké školy osvědčil – viz [20]. Nový systém má na tento úspěch navázat a přenést ho na školy střední a základní. I na nich se totiž učitelé potýkají s opisováním svých svěřenců.

Během své práce na všech těchto aplikacích měl autor možnost programy otestovat v rámci svých praxí. Získaných zkušeností bylo využito k odhalení chyb a vylepšení rozhraní. Žáci se sice podivili, že místo obvyklých dvou variant zadání desetiminutovky dostávají variant více, poté však pracovali jak měli. Z víceméně ekonomických důvodů se však příliš neosvědčilo vytváření zvláštního zadání pro každého žáka – vytištění stránky na inkoustové tiskárně může stát i 3 Kč. Vynásobíme-li toto číslo počtem žáků ve třídě, je jasné, že je tisk takové práce je pro některé školy příliš drahý. Osvědčilo se však připravit šest variant (čtyři pro regulérní termín + dva pro opravný) – při správném rozložení variant mezi žáky je možnost opisování zredukována buď na metody, které jsou poměrně dobře viditelné nebo na pouhé opsání postupu. Rozhodně však pravděpodobnost, že výsledku práce dosáhl student svou samostatnou prací vzrostla. Tento počet variant je pak možno napsat na tabuli, nebo je rozmnožit pomocí kopírky.

Naopak při zadávání domácího úkolu je nutné připravit tolik variant, kolik žáků bude úkol vypracovávat. Tak totiž budou všichni žáci nuceni pracovat

samostatně a nenastane tak úvodní situace z první kapitoly. Pokud si žáci budou při vypracovávání domácího úkolu vzájemně pomáhat, je to jen dobře, protože to přispívá k pochopení učiva jak žákem vysvětlujícím, tak i poslouchajícím. Je zde sice stále ještě možnost, že jeden žák vypracuje domácí úkol za svého kolegu (nebo rodič za svého potomka), tomu se však patrně nebude možno zcela vyhnout nikdy.

7.2 Veřejné prezentace systému

Během vývoje jednotlivých částí projektu byl tento několikrát představen veřejnosti. Kromě v první kapitole zmíněného semináře pro učitele základních a středních škol to bylo při těchto příležitostech:

- **Pedagogický software 2002** je dnes již tradiční konference pořádaná Zemědělskou fakultou Jihočeské univerzity v Českých Budějovicích. Ukázán byl on-line systém a první verze prostředků pro vytváření nových zadání a celková koncepce systému.
- **SVUČ 2004** – Také na naší univerzitě byla práce na tomto projektu představena. Na této „interní“ prezentaci určené pro představení prací studentů v rámci „Studentské vědecké a umělecké činnosti“, byly představeny staré a v té době již vyzkoušené programy pro práci s parametrizovanými úlohami. Šlo tedy zejména o Generátor a Prohlížeč, zmíněn byl také MiStEdit. Na závěr byl také zmíněn záměr funkce těchto programů spojit – záměr, ze kterého vzešla aplikace, kterou jsme v předcházejících kapitolách popisovali. Tehdy však existovala více méně jen jako rozpracovaná koncepce s naznačenými funkcemi.
- **Pedagogický software 2004** – Ve dnech 9. - 10. června 2004 se uskutečnil devátý ročník této konference na stejném místě. Tentokrát byla v příspěvku „Program na generování parametrizovaných testů“ představena nová aplikace - v té době již funkční ve svých základních funkcích – editaci souborů a jednoduchém sestavování písemek.

Při těchto prezentacích samozřejmě byly autorovi kladeny otázky. Některé byly spíše technického rázu a na ty již v tomto textu buď bylo odpovězeno, nebo jsou pro jeho účely nezajímavé. Přesto však existuje otázka, která by zde zodpovězena rozhodně být měla – „Jak je zajištěna stejná obtížnost takto sestavovaných písemek?“ Krátká odpověď zní, že nijak. Je třeba brát v úvahu, že obtížnost úloh je vždy subjektivní a nelze předem s určitostí říct, zda žák bude mít s danou úlohou problémy nebo ne. V každém případě by

však postup ve všech exemplářích příkladu v jednom souboru parametrizovaných úloh měl být zhruba stejný. To však relativně jednoduchý algoritmus této aplikace nedokáže zajistit. Musel by totiž sám tuto úlohu počítat a postupovat při tom stejně jako žák a to je úkol zatím nerealizovatelný. Proto se o srovnatelnou složitost úloh v souboru musí postarat jeho autor. Je jeho odpovědnost vyřadit triviální zadání, zbavit se zadání, která nemají řešení a odstranit i jiné abnormality v souboru.

7.3 Výhody a nevýhody

Systém byl od počátku navrhován pro vytváření zadání písemných prací z předmětů jako je matematika, fyzika, nebo chemie – tedy z předmětů, kde mohou být úlohy postavené na jednoduchých či složitějších výpočtech. Protože však při samotném sestavování prací se pracuje s již předpočítanými úlohami v textovém formátu, lze nový systém použít i v jiných předmětech. Je však vhodné poznamenat, že vytvoření souboru otázek pro (například) zeměpis bude poměrně zdlouhavé a výsledný soubor bude mít velikost spíše skromější.

Na druhou stranu s sebou práce s textem nese také negativa. Systém je takto určen pouze pro tisk zadání písemných prací a je téměř nepoužitelný, pokud budeme chtít zde vytvořené úlohy později použít v nějakém čistě počítačovém systému. Budeme-li chtít tyto úlohy v takovémto systému později používat musíme s touto eventualitou počítat už při návrhu úloh. Nevznikne žádný problém se zadáním úlohy, převod textu z jednoho formátování do jiného je algoritmicky řešitelný. Předpokládáme, že výsledkem počítačově zpracovávané úlohy je buď jediná (celočíselná) hodnota, nebo hodnota s udaným rozptylem přípustných hodnot považovaných za správné. Z textového výsledku by tak měly být tyto hodnoty odvoditelné. Proto u takových úloh autor doporučuje neuvádět výsledek v podobě požadované od studentů (slovní odpověď, jednotky, ...) ale uvést například pouze $2,4$ nebo $2,4 \pm 0,5$.

Hlavními výhodami tedy jsou:

- **Pružnost** – protože je systém založený na předpřipravených úlohách, není systém omezený jen na matematiku a jiné předměty s matematickými prvky, ale lze jej využít v prakticky libovolném vyučovacím předmětu.

- **Otevřenost formátů** – protože je formát datových souborů veřejný, je snadné vytvářet další soubory úloh i v jiných programech a „Parametrizované úlohy“ používat jen pro sestavování prací samotných. Takto lze vytvářet například úlohy založené na výstupu z databází.

- **Otevřenost systému** – čtenáři se společně s touto prací dostávají do rukou kompletní zdrojové kódy aplikace. Takto může být aplikace v budoucnu snadno rozšiřována o nové možosti. Samotná aplikace byla programována právě se zřetelem na skutečnost, že může vyvstat potřeba takových nových prvků.

- **Kvalita výstupu** – co se týká typografické kvality (správný formát vzorců, apod.), nemá TEX konkurenci (a když, tak velmi drahou).

- **Cenová dostupnost** – systém je k dispozici zdarma.

- **Jednoduchost** – ovládání aplikace je relativně snadné, ty jeho prvky, které nejsou intuitivní, jsou popsány v souboru nápovědy. Každý člověk by měl být schopen se s programem naučit pracovat.

- **Nezávislost** – systém lze provozovat bez připojení na internet, na prakticky libovolném počítači PC, vybaveném Windows. Uživatel tak může systém provozovat lokálně a neumožnit nikomu dalšímu přístup ke svým souborům parametriovaných úloh.

Systém má také slabiny:

- **TEX** – tento program nám poskytuje prostředky jak správně zobrazovat vzorce a podobné objekty. Je zde však cena, kterou za jeho použití musíme zaplatit. Je to: čas, který musíme investovat do jeho osvojení, místo na disku, který jeho instalalace zabere a konečně čas potřebný k jeho stažení z internetu.

- **Přiložené příklady** jsou spíše demonstrační – jen naznačují možnosti programu. Jsou to úlohy většinou již vyzkoušené při autorově praxi. Samozřejmě by jich mohlo být více, ale autor se rozhodl věnovat spíše systému pro jejich generování, než vytváření samotnému.

Nově vytvořený systém tedy splňuje všechny požadavky kladené na něj jak v době zadání této práce, tak i vzniklé v průběhu jejího vypracovávání. Snad je dokonce ještě překračuje. Program by tak měl být přínosem pro všechny učitele všech stupňů škol hledající způsob, jak zabránit opisování a přinutit studenty pracovat samostatně v případech, kdy se je k tomu nedaří motivovat jinými prostředky.

Příloha A

Obsah přiloženého CD

Protože psát o čerstvě naprogramované aplikaci a nedat čtenáři možnost ji osobně prozkoumat by bylo přinejlepším neslušné a protože délka zdrojových kódů zmíněné aplikace již dávno přerostla mez, kdy ještě bylo únosné je vytisknout a vložit do této práce jako přílohu (překročila 70 stran), zdá se vhodné přidat k této práci spíše nějaký jiný datový nosič a tyto věci na něj uložit.

Aby se čtenář na kompaktním disku snáze zorientoval, má k dispozici následující přehled jeho obsahu:

- **Instalace** je název složky obsahující instalační soubory aplikace.
- **Zdroje** – tato složka obsahuje kompletní zdrojové soubory programu a také jiné zdroje pro jeho vytvoření použité (například obrázky).
- **Text** je složka, kde je uložena elektronická verze tohoto dokumentu.

Příloha B

Ukázka výsledné písemky

Varianta pro žáky

Desetiminutovka: Neúplná čísla

Michal Stehlík		001
	Bodování:	1;1;1;1
1	Určete součet a rozdíl čísel $a = 17,47 \pm 0,22$ $b = 11,765 \pm 0.012$	
2	Horní a dolní aproximace čísla jsou $a_1 = 16,26$ a $a_2 = 16,32$. Určete střední hodnotu, absolutní a relativní chybu	
3	Určete obsah obdélníka zadaného stranami v cm: $a = 33,32 \pm 0,16$ $b = 10,25 \pm 0,15$	
4	Zapkrohlete číslo π na setiny a určete relativní chybu takového zaokrouhlení.	

Vytvořeno v programu Parametrizované úlohy, formátováno v LaTeXu

Varianta pro vyučujícího

DESETIMINUTOVKA: NEÚPLNÁ ČÍSLA

Michal Stehlík		001
		Bodování: 1;1;1;1
1	Určete součet a rozdíl čísel $a = 17,47 \pm 0,22$ $b = 11,765 \pm 0.012$	$a+b = 29,235 \pm 0,232$ $a-b = 5,705 \pm 0,232$
2	Horní a dolní aproximace čísla jsou $a_1 = 16,26$ a $a_2 = 16,32$. Určete střední hodnotu, absolutní a relativní chybu	$a_0 = 16,29$ $\alpha = 0,03$ $\delta = 0,00184$
3	Určete obsah obdélníka zadaného stranami v cm: $a = 33,32 \pm 0,16$ $b = 10,25 \pm 0,15$	$S = 341,53 \pm 6,638$
4	Zapkrohlete číslo π na setiny a určete relativní chybu takového zaokrouhlení.	$\pi = 3,14 \pm 0,005$ $\delta = 0,0016$

Literatura

[1] Cantù, M.: *Myslíme v jazyku Delphi 6 (1. díl)*. 1. vydání, Praha, Grada Publishing, 2002. 496 str., ISBN 80-247-0334-3.

[2] Cantù, M.: *Myslíme v jazyku Delphi 6 (2. díl)*. 1. vydání, Praha, Grada Publishing, 2002. 510 str., ISBN 80-247-0335-1.

[3] Cantù, M.: *Myslíme v jazyku Delphi 7*. 1. vydání, Praha, Grada Publishing, 2003. 580 str., ISBN 80-247-0694-6.

[4] Holan, T.: *Delphi v příkladech*. 2. vydání, Praha, BEN, 2001. 206 str., ISBN 80-7300-033-4.

[5] Písek, S.: *Delphi – praktické příklady*. 1. vydání, Praha, Grada, 2001. 220 str., ISBN 80-247-0323-8.

[6] Svoboda, L. a kol.: *1001 tipů a triků pro Delphi*. 1. vydání, Brno, Computer Press, 2001. 390 str., ISBN 80-7226-529-6.

[7] Kadlec, V.: *Delphi – Hotová řešení*. 1. vydání, Brno, Computer Press, 2003. 312 str., ISBN 80-251-0017-0.

[8] Bráza, J.: *Delphi 4 – kompletní kapesní průvodce*. 1. vydání, Praha, Grada Publishing, 1999. 600 str., ISBN 80-7169-684-6.

[9] Kadlec, V.: *Učíme se programovat v Delphi a jazyce Object Pascal*. 1. vydání, Praha, Computer Press, 2001. 288 str., ISBN 80-7226-245-9.

[10] Rybička, J.: *LaTeX pro začátečníky*. 3. vydání, Brno, Konvoj, 1999. 238 str., ISBN 80-7302-049-1.

[11] Kočer, M., Sýkora, P.: *Ne příliš stručný úvod do systému LaTeX 2ε*. Český překlad, prosinec 1998. http://www.penguin.cz/~kocer/texty/lshort2e/lshort2e-cz.pdf .

[12] Kopka, H.; Daly, P. W.: *LaTeX – Kompletní průvodce*. 1. vydání, Brno, Computer Press, 2005. 576 str., ISBN 80-7226-973-9.

[13] Malý, M.: *Vyhodnocování matematických výrazů (1. - 3.) – PC Svět* [online]. Cit.: 20. 3. 2005. http://www.pcsvet.cz/art/article.php?id=1376 .

[14] Weisstein, E. W.: *Roman Numerals – from MathWorld* [online]. Cit.: 7. 12. 2004. http://mathworld.wolfram.com/RomanNumerals.html .

[15] Šarounová, A.: *Corona Pragensis (Jak zapsat číslo?)*. [online]. Cit.: 5. 4. 2005. http://praha.astro.cz/crp/0001a.phtml .

[16] Dumois, L.: *Inside Mexico – The Maya Insight Series)*. [online]. Cit.: 5. 4. 2005. http://www.mexconnect.com/mex_/travel/ldumois/ldcmaya.html .

[17] Hošek, P.: *Mayský svět času a čísel*. [online]. Cit.: 7. 4. 2005. http://vesmir.msu.cas.cz/Pavel/mayove_cisla.html .

[18] Kalousek, Z.: *Parametrizace matematických testů*. In: *Seminář o využití výpočetní techniky na technických VŠ*, Editor: Brzezina, M., Liberec, Technická univerzita, 2000. 70 str. ISBN 80-7083-395-5.

[19] Staněk, J.: *Parametrizovaná zadání úloh při výuce matematiky na VŠST v Liberci v 80tých a začátkem 90tých let*. In: *Seminář o využití výpočetní techniky na technických VŠ*, Editor: Brzezina, M., Liberec, Technická univerzita, 2000. 70 str. ISBN 80-7083-395-5.

[20] Vild, J.: *Minihistorie p-zadání na VŠST/TU v Liberci*. In: *Seminář o využití výpočetní techniky na technických VŠ*, Editor: Brzezina, M., Liberec, Technická univerzita, 2000. 70 str. ISBN 80-7083-395-5.

[21] Michale, M.: *Počítačová podpora výuky matematiky* [Diplomová práce]. Liberec, Technická univerzita, 2000. 45 str. Vedoucí dipl. práce: Doc. RNDr. Miroslav Brzezina, CSc.

Seznam obrázků

2.1 Papírové „Parametrizované úlohy“ 13
2.2 MiStEdit 14
2.3 Generátor 15

5.1 Úvodní obrázek s logem programu 32
5.2 Úloha v základním datovém formátu 33
5.3 Editační (hlavní) formulář programu 34

6.1 Mayské číslice a jejich reprezentace ve dvacítkové soustavě . . 48

Seznam tabulek

3.1 Přehled velikostí instalačních balíků programů 20
3.2 Speciální znaky 23

6.1 Základní číslice římské soustavy 46

Rejstřík

Aplikace „PU“
  editační režim, 33
  hlavní nabídka, 34
Aritmetický výraz, 42
  funkce, 45
  hodnota, 42
  závorky, 44

Borland, 28

Datový typ
  extended, 41
  real, 41
Delphi
  komponenty, 28
  výjimky, 28

Generátor, 15

Inno Setup, 30

Jednoduchá matematika, 38

LaTeX (LaTeX) , 19

Microsoft Help Workshop, 30
MiStEdit, 14

Numerické soustavy, 45
  mayská, 47
  nepoziční, 46
  poziční, 47
  římská, 46

Parametrizované úlohy, 12
  aplikace, 31
  soubor, *viz* Soubor parametrizovaných úloh
Pascal, 27
Písemná práce
  obsah, 35
  sestavení, 35
  tiskový soubor, 37
  vzhled, 37

RAD, 28

Soubor parametrizovaných úloh, 32
  formát, 32
  permutace, 48

Testové otázky, 38
TeX (TeX), 19

Zásobník, 43