Zpravodaj Československého sdružení uživatelů TeXu

Marian Genčev

Vícejazyčné pseudonáhodné generování písemných testů z databází

Zpravodaj Československého sdružení uživatelů TeXu, Vol. 30 (2020), No. 1-2, 12–47

Persistent URL: http://dml.cz/dmlcz/150263

Vícejazyčné pseudonáhodné generování písemných testů z databází

Marian Genčev

Cílem tohoto článku je popis třídy `ngt.cls`, kterou autor vytvořil s cílem zjednodušení přípravy písemných testů pro pedagogy disponující běžnou uživatelskou obeznámeností s LaTeXem. Popisované zjednodušení přípravy spočívá především v pseudonáhodném generování testů z předem připravené databáze úloh. K dalším přednostem vytvořené soustavy lze řadit snadnost ovládání pro koncového uživatele a možnost vytvoření verze s výsledky, nebo bez nich. Zápis zadání úloh je uzpůsoben tak, aby bylo možno pracovat s libovolným počtem jazykových verzí v jediném zdrojovém souboru s jednoduchým přepínáním mezi nimi.

1. Úvod

Běžnou součástí pedagogické činnosti na většině typů škol je zjišťování míry pochopení probraného učiva pomocí didaktických testů. Vhodné testy má pedagog zpravidla připraveny již napřed a při menším počtu studentů a s jistou dávkou opatrnosti si může s několika verzemi vystačit.

V případě většího počtu studentů se však již pedagogové mohou setkat při sestavování testů s obtížemi plynoucími z potřeby značného počtu jejich verzí. Charakteristickým problémem jsou třeba situace u vyučovaných celofakultních kursů, jež nejsou u studentů příliš populární (např. matematika nebo fyzika v prvním semestru). V takových případech je procento opakujících studentů poměrně vysoké, čemuž je nutno přizpůsobit strategii vytváření písemných testů za účelem zamezení konfrontace opakujícího studenta s téměř stejným, nebo dokonce stejným zadáním.

Vzhledem k tomu, že studenti rozšiřují písemné testy prostřednictvím sociálních sítí nebo jinak elektronicky, je nutno vytvářet testy buď zcela nové, nebo přeskupit úlohy mezi již existujícími testy. V souvislosti s tímto fenoménem lze v určitém smyslu označit za zajímavý projekt nazvaný `Math4U` provozovaný VŠB-TUO, který studentům nabízí webové rozhraní pro generování testů k procvičování látky, resp. pedagogům možnost náhodně vygenerovat písemný test z vybraných otázek.[1] Všeobecně známou výhodou takového přístupu je, že vyučující nemusí disponovat enormním počtem statických verzí písemných testů, ani nemusí provádět ruční přeskupení úloh. Dle propozicí uvedených na webu projektu obsahuje

[1] `http://math4u.vsb.cz/`

 doi: 10.5300/2020-1-2/12

databáze celkem asi 4 000 úloh z různých oblastí matematiky (středoškolské i vysokoškolské).

Jistou alternativou k `Math4U` je přístup prezentovaný v tomto článku. V něm se zaměříme na popis relativně jednoduchého systému maker v pdfLaTeXu, který je vhodný k automatickému generování písemných testů. Ve srovnání s `Math4U` může být počáteční nevýhodou vytvoření dostatečně velké databáze v jazykových verzích, s nimiž budeme pracovat. Na druhou stranu máme možnost zcela kontrolovat obsah databáze, přirozeně rozšiřovat databázi o další úlohy nebo nové jazykové verze, resp. možnost si kdykoliv výstup nebo formát upravit podle vlastních představ.

Princip, který používáme při náhodném generování testů, spočívá na vytvoření databáze typových úloh s t dílčími databázemi $d_1, \ldots, d_t$, přitom z každé z nich bude náhodně[2] vybrána úloha. Z takto vybraných úloh sestavíme písemný test, který bude pro studenta při dostatečně početné databázi úloh vždy poměrně nový.

2. Existující přístupy a motivace

V současné době existuje v distribuci TeXu několik balíčků, které podporují náhodné generování písemných testů z databází.[3] Mezi nimi vynikají např.

`AcroTeX`, `esami`, `examdesign`, `probsoln`.

Uvedené balíky využívají pro generování pseudonáhodných čísel makra D. Arsenaua oficiálně k dispozici na `https://ctan.org/pkg/random`, o nichž je známo, že generují nové pseudonáhodné číslo při opětovné kompilaci až při změně hodnoty minutového čítače `\time`. Protože se chceme takovému omezení vyhnout, je v třídě `ngt.cls` pro generování pseudonáhodných čísel využito základní funkcionality pdfTeXu,[4] primitivu `\pdfuniformdeviate`, viz [1, s. 40]. Pro vygenerování pseudonáhodného celého čísla z množiny $\{0, 1, \ldots, n-1\}$ tak stačí psát

```
1 \pdfuniformdeviate n
```

přičemž toto číslo není generováno pouze jednou za minutu, ale každou mikrosekundu.

Kromě zmíněného nedostatečně častého generování pseudonáhodného čísla ve výše uvedených balících je pro precizní výstup podle požadavků uživatele nutno nastavit řadu parametrů, což vyžaduje prostudování někdy i velmi obsáhlých manuálů (např. balík `AcroTeX`). Vzhledem ke skutečnosti, že studium rozsáhlejších manuálů nebo akceptování ústupků oproti naší představě může být málo

[2] Slovem „náhodně“ zde i v dalším textu myslíme vždy „pseudonáhodně“.

[3] Podrobněji viz třeba `https://ctan.org/topic/exam`.

[4] Nabízené řešení popsané v dalším je proto vhodné pro uživatele kompilující pdfTeXem s verzí 1.30.0 nebo vyšší.

uspokojující, rozhodl se autor sestavit vlastní makra, která slouží popisovanému účelu a jejich případná další modifikace se jeví jako snadná. Motivací pro jejich vytvoření bylo zformování takového prostředí pro koncového uživatele, aby zápis úloh a řešení, resp. následné generování písemných testů z nich, bylo velmi jednoduché. K přednostem lze řadit tyto vlastnosti:

- snadnost zápisu úloh a doprovodných informací do databází,
- kontrola nad obsahem databáze i formou výstupu,
- generování nové náhodné písemky nezávisle na hodnotě čítače `\time`,
- možnost přidání libovolného počtu jazykových verzí,
- intuitivní přepínání mezi jednotlivými verzemi výstupu,
- manuál umožňující rychlé pochopení ovládání i pro začátečníky.

Autor věří, že popisovaná třída může být pro běžné uživatele LaTeXu užitečným nástrojem při generování testů, zatímco zkušeným TeXpertům poslouží třeba jako východisko pro její další úpravy, které jsou jistě možné v mnoha ohledech. Pro oba dva jmenované účely je zájemcům k dispozici zdrojový kód `ngt.cls` na webové stránce `http://robimematiku.cz/bebechy/`.

Zbytek článku lze rozdělit do dvou logických oddílů. První má formu uživatelského manuálu a popisuje, jakým způsobem je možno třídu `ngt.cls` použít. Druhý oddíl je věnován implementaci a popisujeme v něm zvolené technické řešení.

3. Uživatelský manuál třídy `ngt.cls`

Tato část je věnována uživatelskému seznámení s třídou `ngt.cls`, která disponuje dvěma volbami – `databaze`, určenou k sazbě úloh v dílčích databázích, a `pisemka`, určenou k sazbě náhodně generovaného testu z úloh uvedených v připraveném souboru databází. Jejich aktivace je obvyklá, tj. stačí psát jednu z možností

```
2 \documentclass[databaze]{ngt}
```

nebo

```
3 \documentclass[pisemka]{ngt}
```

Pro správné fungování třídy `ngt.cls` je vyžadována instalace třídy `article` a balíků `lmodern`, `inputenc`, `fontenc`, `geometry` a `babel`, které jsou třídou načítány (detaily viz sekci 4 popisující implementaci). Pro sazbu databáze i písemky předpokládáme, že zdroj bude zapisován v kódování `utf8`.

Než se v dalším blíže vyjádříme k jednotlivým volbám, připojíme ještě krátkou technickou poznámku. Při popisu příkazů uvádíme vždy název příkazu a jeho syntaxi, přitom na konci řádku bude uvedeno `[P]` nebo `[T]` značící, že daný příkaz lze použít jen v preambuli, resp. jen v textové části, tj. za `\begin{document}`. Jsou-li uvedeny obě volby, je možno příkaz použít v obou částech dokumentu. Pokud je v syntaxi za názvem příkazu použit místo černé barvy odstín šedé,

označuje tato skutečnost variabilitu syntaxe, která je v popisu příkazu vysvětlena. Např. zápis

```
\prikaz #1/#2                                                    [P]
```

uvozuje uživatelský popis příkazu `\prikaz`, který lze použít pouze v preambuli a má jeden povinný parametr. Ten je možno v případě potřeby doplnit nepovinným druhým argumentem za lomítkem.

3.1. Volba `databaze`

3.1.1. Vytvoření databáze – základní přehled

Při aktivní volbě `databaze` jsou uživateli k dispozici prostředky uvedené v Tabulce 1.

`\databaze` `\subdatabaze`	zahájení sazby příslušné databáze úloh nebo její části
`\uloha` `\reseni`	zápis zadání úlohy, resp. jejího řešení
`\jazyky`	načtení jazyků, v nichž budeme text zapisovat
`\jazyk`	specifikace jazyku, v němž bude text vysázen
`\vyres`	ovládání zobrazení výsledků

Tabulka 1: Příkazy dostupné při volbě `databaze`

Pro základní představu uvádíme příklad struktury souboru `banka_uloh.tex`, v němž `<uloha_m_n>` značí zadání a eventuální řešení úlohy, která budou později zapsána pomocí příkazu `\uloha`, eventuálně `\reseni`, viz následující odstavce.

banka_uloh.tex

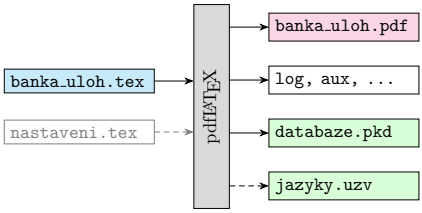
```
\documentclass[databaze]{ngt}
  \jazyky <deklarace pouzivanych jazyku>
  \jazyk  <specifikace jazyku>
  \vyres  <0/1>

\begin{document}
\databaze <Nazev_databaze_1>/<bodovy_zisk_1>b
  <uloha_1_1>
  <uloha_1_2>
  ...
  <uloha_1_n>

```

```
\databaze <Nazev_databaze_2>/<bodovy_zisk_2>b
  ...
\databaze <Nazev_databaze_3>/<bodovy_zisk_3>b
  ...
\end{document}
```

Podstatným výstupem kompilace souboru databází pomocí

```
pdflatex banka_uloh.tex
```

je soubor `banka_uloh.pdf` obsahující zapsané úlohy a řešení ve zvoleném jazyce a v podobě, kterou budou mít na písemce generované později. Uživatel tak má náhled do souhrnu úloh a lepší kontrolu nad sázeným obsahem. Při kompilaci vznikají ještě další pracovní soubory s příponami `pkd` a `uzv` (viz Obrázek 1), které jsou třídou `ngt` načítány při volbě `pisemka` (viz Obrázek 3). Soubory s příponami `pkd` a `uzv` mají vždy stejný název nezávisle na názvu kompilovaného souboru (zde `banka_uloh.tex`). Pro běžného uživatele a pro popis příkazů v této části textu nejsou důležité, a proto se o nich přesněji zmíníme až v sekci o implementaci.



Obrázek 1: Výstupy kompilace souboru `banka_uloh.tex`

O významu nepovinného souboru s uživatelským nastavením `nastaveni.tex` je podrobněji pojednáno v sekci 3.1.3.

3.1.2. Popis příkazů pro sazbu databáze

`\jazyky` `<jazyk_1>`/`<jzk_1>`,...,`<jazyk_s>`/`<jzk_s>` [P]
`\jazyk` `<jzk_i>` [P]

Vzhledem k tomu, že příkazy `\jazyky` a `\jazyk` jsou úzce svázány, vyložíme jejich užití současně v následujících odstavcích.

Pomocí příkazu `\jazyky` deklarujeme v preambuli zdrojového souboru názvy jazyků, v nichž budeme text databáze sázet (`<jazyk_1>`, ..., `<jazyk_s>`) a zástupné zkratky jazyků (`<jzk_1>`, ..., `<jzk_s>`), pomocí nichž budeme v případě vícejazyčné volby konkrétní jazyk v preambuli souboru `banka_uloh.tex` volat příkazem `\jazyk`.

Název jazyku a s ním svázaná zkratka musí být v deklaraci `\jazyky` odděleny lomítkem `/`. Koncovým separátorem příkazu `\jazyky` je konec řádku a v případě užití více jazyků je nutno dílčí jazykové deklarace oddělit čárkou. Je důležité poznamenat, že název jazyku `<jazyk_i>` musí být ve shodě s názvy jazyků balíku `babel`, který třída `ngt` načítá, a že mezera je v syntaxi příkazu `\jazyky` nežádoucím znakem. Další popis obou příkazů rozčleníme podle počtu načítaných jazyků.

Sazba textu databáze v jediném jazyce

V tomto případě lze deklaraci uvádět pouze ve tvaru

```
24 \jazyky <jazyk_1>
```

Tím je implicitně zajištěno načtení balíku `babel` s vybraným jazykem `<jazyk_1>`. Uvedení specifikace `\jazyk` se v tomto triviálním případě neočekává, protože jsme v jazykové deklaraci žádnou zkratku jazyku neuvedli. Chceme-li např. sázet databázi úloh ve slovenštině, píšeme

```
25 \jazyky slovak
```

Situace může být ovšem mírně odlišná u uživatelů sázejících text databáze prozatím v jediném jazyce, avšak v budoucnu zamýšlejících doplnit ještě alespoň jednu další jazykovou verzi. V tomto případě je vhodné již v přípravné jednojazyčné fázi zapisovat texty úloh do makra odpovídajícího příslušné jazykové verzi. Je proto lepší deklarovat

```
26 \jazyky <jazyk_1>/<jzk_1>
```

a text úlohy zapisovat užitím

```
27 \<jzk_1>{<text v jazyce jazyk_1>}
```

kde řetězec `<jzk_1>` je uveden v deklaraci `\jazyky` za lomítkem. Pokud budeme chtít psát např. text databáze v budoucnu vícejazyčně a máme aktuálně k dispozici prozatím třeba slovenskou verzi zadání, deklarujeme v preambuli

```
28 \jazyky slovak/sk
```

a text úlohy sázíme pomocí

```
29 \sk{Slovenská verzia...}
```

Případná specifikace jazyku sazby pomocí `\jazyk sk` je sice možná, ale nadbytečná, protože třída `ngt` ji načítá automaticky.

Konečně poslední situací jednojazyčné sazby textu databáze je forma privilegia poskytnutá uživateli preferujícímu češtinu. Pokud v preambuli souboru `banka_uloh.tex` neuvedeme deklaraci `\jazyky`, je tento stav ekvivalentní s deklarací `\jazyky czech`. Sázíme-li ale v jediném jazyce, kterým není čeština, explicitně zapíšeme do preambule `\jazyky <nejaky_jazyk>` (např. řádek 25), případně i jeho zkratku za lomítko (např. řádek 28). Tím uživatel zajistí načtení vzorů pro dělení slov odpovídajících deklarovanému jazyku.

Sazba textu databáze ve více jazycích

Zde je popis snadnější. Mohou nastat celkem dvě běžné situace. První z nich je uvedení obou příkazů `\jazyky` a `\jazyk`. Uveďme příklad, kdy bude uživatel sázet text databáze ve třech jazycích, němčině, češtině a angličtině. Jazyková deklarace bude mít formát

```
30 \jazyky german/de,czech/cz,english/en
```

čímž je zajištěno načtení `\usepackage[german,czech,english]{babel}` a uživatelské zpřístupnění příkazů `\de`, `\cz` a `\en`, pomocí nichž budeme zapisovat jednotlivé jazykové verze textu databáze, např.

```
31 \cz{Toto je česká verze.}%
32 \en{This is the english version.}%
33 \de{...und endlich auch die deutsche.}
```

Jazyková verze zobrazená ve výstupním souboru je určena specifikací `\jazyk`. Např. pro zobrazení anglické verze píšeme do preambule

```
34 \jazyky german/de,czech/cz,english/en
35 \jazyk  en
```

přitom se automaticky na začátek dokumentu zapíše `\selectlanguage{english}` pro načtení odpovídajících vzorů dělení slov. Platí, že třída `ngt` při specifikaci `\jazyk <jzk_i>` zapíše na začátek dokumentu `\selectlanguage{<jazyk_i>}` a není třeba se o tuto věc dále starat.

Je nutno upozornit, že zkratku jazyku `<jzk_i>` musíme volit s jistou opatrností tak, aby zpřístupněné makro `\<jzk_i>` nekolidovalo s již definovanými makry. Nebylo by např. vhodné zavádět zkratku `ge` pro německý jazyk, protože příkaz `\ge` je běžně užíván v matematické sazbě pro vysázení relace $\geq$.

Druhá situace, která může nastat, je případ, kdy sice uvedeme deklaraci `\jazyky`, ale již nikoliv specifikaci `\jazyk`. V takovém případě bude ve výstupním souboru zobrazena jazyková verze odpovídající prvnímu jazyku v pořadí uvedeném za `\jazyky`. Ve výše uvedeném příkladu by to znamenalo zobrazení varianty v německém jazyce.

`\databaze #1/#2b` [T]

Obecně předpokládáme, že písemka bude sestávat z t různých typů úloh. Pro pozdější rozlišení při generování náhodného testu proto každému typu přiřadíme samostatnou dílčí databázi d_i, $i = 1, \ldots, t$, jejíž sazbu zahájíme pomocí `\databaze`. Do argumentů příkazu jsou načítány tyto informace: název databáze (`#1`), číselná hodnota symbolizující bodové ohodnocení úloh dané databáze v písemném testu (`#2`). Koncovým separátorem argumentu `#2` je písmeno `b`, které současně symbolizuje, že se jedná o bodové ohodnocení. Třída `ngt.cls` detekuje konec dané databáze buď v místě následujícího nejbližšího příkazu `\databaze` nebo při dosažení konce souboru `banka_uloh.tex`. Příkladem užití může být zápis

```
37 \databaze Množiny/8b
```

uvozující databázi úloh „Množiny“ s bodovým ohodnocením 8 bodů. V případě vícejazyčné varianty lze psát např.

```
38  \databaze\cz{Množiny}\en{Sets}\de{Mengen}/8b
```

přitom na pořadí jazykových verzí nezáleží.

`\subdatabaze #1` [T]
Někdy je vhodné pro lepší orientaci rozdělit dílčí databázi úloh týkající se jediného vyučovacího celku na menší úseky. V takovém případě lze použít příkaz `\subdatabaze`, který do svého argumentu načítá její název. Koncovým separátorem argumentu je konec řádku. Bodové ohodnocení úloh libovolné subdatabáze je určeno v nejbližší předchozí části `\databaze` a není třeba jej specifikovat. Příkladem užití v jednojazyčné (české) verzi je zápis

```
40  \databaze Soustavy lineárních rovnic/5b
41    \subdatabaze Gaussova eliminace
42      <ulohy a reseni>
43    \subdatabaze Toky v sítích
44      <ulohy a reseni>
45    \subdatabaze Cramerovo pravidlo
46      <ulohy a reseni>
```

`\uloha #1***` [T]
`\reseni` [T]
Pro sazbu úloh v databázích je určen příkaz `\uloha`, jehož jediným argumentem je zadání úlohy a případné řešení. Konec každé úlohy musí být označen třemi hvězdičkami, které slouží jako koncový separátor argumentu. Při běžném užití se jeví tato možnost jak postačující, tj. nepředpokládáme užití tří po sobě jdoucích hvězdiček za sebou v textu úlohy, tak především snadná a přehledná.[5]

Příkladem sazby textu úlohy pro vícejazyčnou databázi při jazykové deklaraci na řádku 30 je třeba

```
49  \uloha
50    \cz{Text úlohy v českém jazyce.}%
51    \de{Aufgabentext in deutscher Sprache.}%
52    \en{Text of the problem in English.}%
53  ***
```

Je nutno si uvědomit význam uvedených znaků `%` na konci zápisu každé jazykové verze. Bez jejich uvedení by byly do textu úlohy zavlečeny nežádoucí mezery.

Jestliže z nějakého důvodu nechceme některou z úloh použít při pozdějším náhodném generování testu a ani ji zobrazit v souhrnném přehledu při kompilaci

[5] Makra byla vytvářena v adventním období. Čtenář se může zamyslet, zda tato skutečnost nemohla mít vliv na volbu koncového separátoru.

souboru `banka_uloh.tex`, je možno namísto smazání celé úlohy z databáze tuto pouze označit hvězdičkou bezprostředně za příkazem a psát

```
\uloha*
  Tady je zadání (ignorované) úlohy.
***
```

Snadné vyřazení vybraných úloh při pozdějším generování testu může být výhodné třeba po zveřejnění vzorového testu obsahujícího takto označené úlohy.

V případě, že k úloze chceme připojit řešení nebo návod k řešení, který můžeme využít při opravování testů, zapíšeme do těla úlohy separátor `\reseni`. Příkladem může být třeba

```
\uloha
  Tady je zadání úlohy.
\reseni
  A sem zapíšeme výsledek, návod nebo i celý postup řešení.
***
```

`\vyres #1` [P]

Přepínač `\vyres` slouží k ovládání zobrazení výsledků. Přípustné hodnoty pro `#1` jsou pouze hodnoty `0` a `1`.

Při nastavení na `\vyres 0`, resp. `\vyres 1`, vypneme, resp. zobrazíme, řešení úloh databáze ve výstupu kompilace `banka_uloh.tex`. Pokud přepínač v preambuli databáze neuvedeme, bude implicitně nastaveno `\vyres 1`. Pokud je zobrazení řešení povoleno, je vždy uvozeno k tomu určenou ikonou . Jinou ikonu nebo návěští je možno nastavit makrem `\ReseniIkona`, viz sekci 4.1.2.

6.4 Řešte danou diferenciální rovnici s počátečními podmínkami:

$$y'' + 9y = 27x^2 - 18, \qquad y(0) = 2, \quad y'(0) = 0.$$

$y(x) = \frac{14}{3} \cdot \cos(3x) + 3x^2 - \frac{8}{3}$.

Obrázek 2: Zobrazené řešení úlohy v databázi

3.1.3. Načítání dalších uživatelských nastavení

`nastaveni.tex`

Pro generování písemky pomocí třídy `ngt` je nutná úspěšná kompilace souboru `banka_uloh.tex`, při niž jsou zapsána podstatná data do souboru `databaze.pkd`. V případě, že je nutno načíst v preambuli další balíky, definice nebo nastavení, lze to provést explicitně v preambuli souboru `banka_uloh.tex`.

Vzhledem k tomu, že popisované nastavení bude potřeba i při generování písemky, doporučujeme spíše vytvořit v aktuálním adresáři soubor s názvem `nastaveni.tex` obsahující tato nastavení. Pokud bude takový soubor v adresáři existovat (s identickým názvem i příponou), třída `ngt` jej automaticky načte při obou volbách (viz Obrázek 3), aniž by bylo nutno to provést ručně. Pokud takový soubor v aktuálním adresáři nebude zjištěn, žádné doplňující nastavení se nenačte. Oproti přímému zápisu nastavení do preambule souboru `banka_uloh.tex` se takto zásadně zvýší její přehlednost, což je důležité. Další výhodou je, že nastavení pro dva soubory (databáze a písemka) můžeme kontrolovat na jediném místě. Praktickou možností využtí je ale i zápis definic maker obsahujících opakující se textové části úloh, např. v rámci jednotlivých databází.

3.2. Volba `pisemka`

3.2.1. Vygenerování písemného testu – základní přehled

Pro vygenerování testu nejprve vytvoříme samostatný soubor, např. `pisemka.tex`, který musí být ve stejném adresáři jako i dříve vytvořený soubor `banka_uloh.tex` a který načítá třídu `ngt` s volbou `pisemka`. Pro ovládání a generování výstupu jsou k dispozici příkazy uvedené v Tabulce 2.

`\generujTEST`	makro pro generování písemného testu
`\jazyk`	ovládání jazyku výstupu
`\vyres`	ovládání zobrazení výsledků
`\novy`	ovládání náhodnosti generování při další kompilaci
`\body`	ovládání zobrazení bodového zisku u každé úlohy
`\hlavicka`	makro pro sazbu uživatelem definovaného záhlaví testu
`\datum` `\Datum`	makro pro formátování, resp. tisk datumu

Tabulka 2: Příkazy dostupné při volbě `pisemka`

Základní struktura souboru `pisemka.tex` může vypadat takto:

pisemka.tex

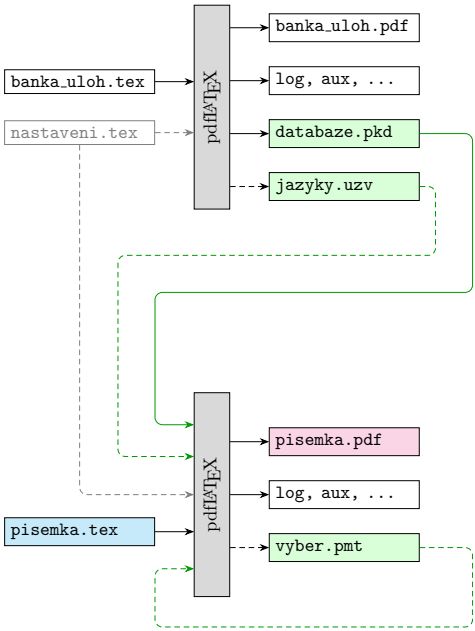
```
\documentclass[pisemka]{ngt}
  \jazyk <jzk_i>
  \vyres <0/1>
  \novy  <0/1>
  \body  <0/1>
  \datum <specifikace>
```

```
70    \hlavicka
71      <(vicejazycny) obsah sazeny v hlavicce>
72    ***
73
74  \begin{document}
75    \generujTEST
76  \end{document}
```

Pro úspěšnou kompilaci pomocí

```
77  pdflatex pisemka.tex
```

se předpokládá dříve provedená kompilace databáze `banka_uloh.tex`.



Obrázek 3: Kompilace souboru `pisemka.tex`

Jsou-li všechny požadované soubory k dispozici (viz kompilační schéma na Obrázku 3)[6], dostáváme spuštěním příkazu na řádku 77 soubory `pisemka.pdf` a `vyber.pmt`. Zatímco soubor PDF obsahuje vygenerovaný test z dříve vytvořených databází, soubor s příponou `pmt` obsahuje informace o posledním náhodném

[6]Čárkovaná propojení symbolizují načítání nebo zápis souborů pouze při jejich existenci, resp. při odpovídajícím uživatelském nastavení.

výběru úloh z jednotlivých databází. Tento soubor je možno chápat jako paměťový, který při vypnutém náhodném generování v `pisemka.tex` umožňuje načíst údaje o výběru úloh z minulé kompilace (podrobněji viz popis ovladače `\novy`).

3.2.2. Popis příkazů pro generování testu

`\generujTEST` [T]
Příkaz zajišťující generování testu z dříve vytvořených databází. Ve většině případů se jedná o jediný údaj zapsaný mezi `\begin{document}` a `\end{document}`.

`\jazyk <jzk_i>` [P]
Nastavením specifikace `\jazyk <jzk_i>` určujeme, v jakém jazyce bude vygenerován test, přitom zkratka `<jzk_i>` musí být jedna z platných zkratek jazyků uvedených v souboru `banka_uloh.tex` v deklaraci `\jazyky`. Pokud není uvedena žádná specifikace, bude test vygenerován v jazyce, který je uveden v deklaraci `\jazyky` na prvním místě. Zapsání deklarace `\jazyky` do souboru `pisemka.tex` je provedeno automaticky ve shodě s deklarací v soboru `banka_uloh.tex`. To je znázorněno na Obrázku 3 v podobě načtení souboru `jazyky.uzv`. Uvádění této deklarace v souboru `pisemka.tex` je tudíž nadbytečné a neočekává se.

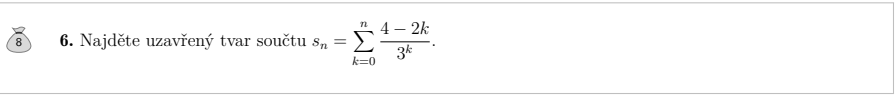
`\novy #1` [P]
Přepínač umožňující povolení, nebo naopak zablokování generování nové náhodné písemky při následující kompilaci. Povolené hodnoty za `#1` jsou buď `0`, nebo `1`.

Zamknutí náhodného výběru pomocí `\novy 0` může být užitečné tehdy, když chceme vygenerovat písemku se stejným výběrem úloh i při opakované kompilaci (třeba nejprve v českém a potom v anglickém jazyce, resp. nejprve písemku pro studenty bez řešení a potom tutéž písemku s řešeními nebo návody pro pedagoga). Není-li přepínač v preambuli uveden, je načteno implicitně `\novy 1`.

Je nutno poznamenat, že při zcela první kompilaci souboru pro generování písemky není rozumné mít nastaveno `\novy 0`, kdy nebyl paměťový soubor s příponou `pmt` ještě vytvořen. Nastane-li přece jen tento stav, je vyhodnocen jako chyba.

`\body #1` [P]
Přepínač pro ovládání zobrazení maximálního bodového zisku u každého návěští úlohy na levém okraji písemky (připomeňme, že zmíněné bodové zisky byly zadány u zahájení každé databáze v souboru `banka_uloh.tex`). Povolené hodnoty za `#1` jsou buď `0` nebo `1`.

Při `\body 0` nejsou bodová ohodnocení zobrazena, při `\body 1` naopak ano. Pokud není přepínač uveden, je implicitně nastaveno `\body 1`. Protože předpokládáme sazbu v libovolném jazyce podle deklarace `\jazyky`, bylo nutno zvolit místo jednotek bodového ohodnocení (např. body, b, points, pts) grafický prvek, který by evokoval patřičný dojem zisku, zde v podobě měšce. Pokud není čtenáři



Obrázek 4: Zobrazený bodový zisk u zadání úlohy písemného testu

popisovaná grafická úprava sympatická, může nastavit `\body 0` nebo předefinovat makro `\BodyIkona`, viz odstavec 4.3.4.

`\vyres` [P]
Funkce tohoto ovladače je shodná s funkcí stejně nazvaného ovladače u volby `databaze`.

`\hlavicka #1***` [P]
Ačkoliv název makra napovídá, že tento příkaz bude zajišťovat sazbu hlavičky písemky, začneme popisem situace, kdy jej v preambuli neuvedeme. V takovém případě bude vysázena implicitně nastavená hlavička obsahující místo pro jméno a osobní číslo studenta a v pravé části hlavičky bude vytištěno také datum.

Obrázek 5: Implicitně nastavená hlavička testu

Navíc při současně uvedeném `\vyres 1` (verze se zobrazenými výsledky pro pedagoga) bude implicitně nastavená hlavička zredukována na datum konání písemného testu a dolní dvojici linek. V takovém případě jsou totiž ikony pro jméno a osobní číslo studenta zbytné.

Pokud do preambule souboru `pisemka.tex` uvedeme makro `\hlavicka`, může pracovat dvěma způsoby, čemuž odpovídá i dvojí syntaxe příkazu. Píšeme-li

```
84 \hlavicka 0
```

nebude zobrazena žádná hlavička. Pokud píšeme

```
85 \hlavicka <nejaky_kod>***
```

kde `<nejaky_kod>` nezačíná znakem `0`, vytiskne se hlavička písemného testu podle zapsaného kódu `<nejaky_kod>`, přitom koncovým separátorem argumentu je již známá trojice hvězdiček.[7] Při této syntaxi je možno makro chápat jako kontejner, do nějž lze zapsat příkazy, které budou vykonány hned po `\begin{document}`

[7] Je malá pravděpodobnost, že hlavička písemky bude začínat tokenem `0`. Pokud taková výjimečná situace nastane, lze psát `\hlavicka\relax 0...***`.

a jazykových nastaveních dokumentu. Příkladem snadné hlavičky v jednojazyčné české verzi může být

```
\hlavicka
  Jméno:\hfill\Datum[8]
  \par\medskip
  \hrule\bigskip
***
```

Po načtení příslušných balíků v preambuli `pisemka.tex` lze však použít i daleko širších grafických možností (např. `pstricks`, `tcolorbox` nebo `tikz`).

`\datum #1` [P]
`\Datum` [P],[T]
Příkaz `\datum` slouží pro nastavení datumu konání písemky, který bude vytištěn pomocí příkazu `\Datum` (to je voláno i při sazbě implicitně nastavené hlavičky). Koncovým separátorem argumentu `#1` příkazu `\datum` je konec řádku.

Chování makra je určeno prvním znakem v načteném argumentu `#1`. Pokud je prvním znakem `+` nebo `-`, které je následováno celým číslem `<z>`, bude aktuální datum v době kompilace změněno o $+$`<z>` nebo $-$`<z>` dní. Pokud `<z>` není celé číslo, bude vygenerována chybová hláška. Nezačíná-li `#1` ani jedním ze znaků `+` nebo `-`, bude při použití `\Datum` (např. v uživatelem definované hlavičce) vytištěn obsah argumentu `#1`. Pokud v preambuli neuvedeme příkaz `\datum` s nějakou specifikací, vytiskne `\Datum` aktuální datum v době kompilace souboru. Příklady užití jsou třeba

```
\datum +2
\datum -1
\datum 10. ledna 2021
```

kde jednotlivé případy postupně nastaví datum na pozítří, včerejší datum, resp. fixní datum (à la DEK). Výhodou notace `\datum +/-<z>` je, že při změně hodnoty přepínače `\jazyk` bude datum vytištěný pomocí `\Datum` přeložen do příslušného jazyku.

4. Implementace

Tato část je věnována technické realizaci třídy `ngt`, jejíž báze je tvořena základní třídou v LaTeXu, `article`.

[8] Makro `\Datum` je popisováno v následujícím odstavci.

4.1. Preambule

4.1.1. Zavádění voleb třídy

Nejdříve standardním způsobem zavádíme obě volby třídy `ngt`, tj. `databaze` a `pisemka`. Přitom požadujeme, aby byla třída načtena vždy s právě jednou volbou. Proto zavádíme čítač `\PocetVoleb`, do nějž je uložen počet načtených voleb třídy `ngt`.

ngt.cls

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{ngt}[2020/02/29]
\LoadClass[12pt]{article}

\newcount\PocetVoleb
\newif\ifdatabaze

\DeclareOption{databaze}{%
  \databazetrue
  \advance\PocetVoleb by 1}

\DeclareOption{pisemka}{%
  \databazefalse
  \advance\PocetVoleb by 1}

\ProcessOptions
```

V případě, že není načtena žádná volba nebo načítáme obě volby současně, bude tento stav vyhodnocen jako chyba.

```
\ifnum\PocetVoleb=1
\else
  \ClassError{ngt}{%
    Prilis mnoho nactenych voleb nebo zadna volba}{}%
\fi
```

4.1.2. Společná nastavení

Třída `ngt` implicitně načítá balíky `lmodern`, `inputenc` a `fontenc`, přitom předpokládáme, že zdrojový soubor bude zapisován v kódování `utf8`. Pro nastavení délkových registrů geometrie stránky načítáme balík `geometry` s nastavením okrajů na 2 cm.

```
\RequirePackage{lmodern}
\RequirePackage[utf8]{inputenc}
\RequirePackage[T1]{fontenc}
\RequirePackage[margin=2cm]{geometry}
```

```
\def\bfdefault{b}
```

Dále zavedeme čítač `\PocetDatabazi`, s jehož pomocí evidujeme celkový počet uvedených databází. Při sazbě eventuálního řešení v databázi nebo písemce nastavujeme vertikální odsazení řešení od předchozího zadání jako `\bigskip`. Pro vyšší přehlednost kódu uvádíme také zkrácené verze primitivů `\expandafter` a `\noexpand`.

```
\newcount\PocetDatabazi
\parindent=0pt
\let\OdsadReseni\bigskip
\let\ea\expandafter
\let\nx\noexpand
```

Pro sazbu případného řešení nyní nadefinujeme jeho návěští, které bude obsahovat jistý grafický element (v případě vícejazyčného dokumentu je tato možnost vhodnější) a příslušné odsazení. Abychom se vyhnuli načítání externích obrázků nebo balíků, které generují grafiku, využíváme možností pdfTEXu disponujícího příkazem `\pdfliteral`, viz Olšák [2]. Analýzu následujícího kódu v definici příkazu `\ReseniIkona` přenecháváme za cvičení.

```
\def\ReseniIkona{%
  \pdfliteral{%
   q .7 g
   .9 0 0 .9 0 0 cm
   0 0 4 4 re f
   1 0 0 1 5 0 cm
   0 0 4 4 re f
   1 0 0 1 0 5 cm
   0 0 4 4 re f
   .7071 .7071 -.7071 .7071 -5 1 cm
   1 0 0 rg
   0 0 4 4 re f Q}}
```

Za účelem velmi častého vykreslování ikony řešení v PDF především při volbě `databaze` (velký počet úloh, a tedy i řešení) je výhodnější vykreslit ikonu pouze jednou a později se na ni odkazovat pomocí `\pdfrefxform`.

```
\setbox 0\hbox to 1.8em{%
  \hskip3pt\vbox to 1em{\vss\ReseniIkona}\hss}
\pdfxform 0
\edef\PDFreseni{\the\pdflastxform}

\def\NavestiReseni{%
  \leavevmode
  \pdfrefxform\PDFreseni}
```

4.1.3. Pomocná makra

V dalším budeme několikrát potřebovat zapisovat a číst jistá data. Pro zjednodušení definujeme makra `\otevri`, `\zapis` a `\zavri`, která zápis kódu příslušejícího těmto úkonům zkrátí.

```
\def\otevri{%
  \immediate\openout }
\def\zapis{%
  \immediate\write }
\def\zavri{%
  \immediate\closeout }
```

Pro pohodlnější čtení argumentů obsahujících lomítko (např. jazyková deklarace `\jazyk`), bude užitečné umět uložit si část před a za lomítkem do odpovídajících maker.

```
\def\PredLomitkem#1/#2,{#1}
\def\ZaLomitkem#1/#2,{#2}
```

Několikrát bude potřeba v zadaném seznamu tokenů spočítat počet výskytů konkrétního tokenu (postačí nám vnější tokeny, tedy neuvedené uvnitř skupiny). Definujeme proto makro `\PocetVyskytu`, které má dva povinné parametry. Prvním parametrem je token, jehož počet výskytů chceme spočítat. Druhým parametrem je čítač, v němž tento počet bude uložen. Za touto specifikací následuje předložený seznam tokenů (zpravidla text), v němž počet výskytů zjišťujeme. Je-li načten token `\UkonciPocitadlo`, je činnost makra `\PocetVyskytu` ukončena.

```
\def\UkonciPocitadlo{}
\def\UkonciPocitadloToken{\UkonciPocitadlo}
\newcount\pocet

\def\PocetVyskytu[#1,#2]{%
  \def\TestovanyZnak{#1}%
  \pocet=0
  \newcount#2%
  \let\pocitadlo#2%
  \let\next\nacti\next}

\long\def\nacti#1{%
  \def\znak{#1}%
  \ifx\znak\UkonciPocitadloToken
    \def\next{\pocitadlo=\pocet}%
  \else
    \ifx\znak\TestovanyZnak
      \advance\pocet by 1
    \fi
```

```
79    \fi
80    \next}
```

4.1.4. Jazyková nastavení

V této části se budeme věnovat popisu maker, která umožňují jazyková nastavení, tj. maker `\jazyky`, `\jazyk` a pomocných maker. V souvislosti s těmito makry zavádíme čítač `\PocetJazyku`, v němž bude uložen celkový počet jazyků uvedený v deklaraci `\jazyky`. Čítač `\PoradiJazyku` má pouze pomocný charakter.

```
81  \newcount\PocetJazyku
82  \newcount\PoradiJazyku
```

Jazyková nastavení dokumentu budou určena tím, zda uživatel uvede jazykovou deklaraci `\jazyky`, resp. specifikaci `\jazyk`, a co do nich zapíše. Pro detekci uvedení těchto nastavení zavádíme logické proměnné `\ifjazyky` a `\ifjazyk`, které nastavujeme na `false`.

```
83  \newif\ifjazyky
84  \newif\ifjazyk
85    \jazykyfalse
86    \jazykfalse
```

Aby nebylo nutno opakovaně zapisovat shodnou deklaraci `\jazyky` u databáze i písemky, zapíšeme toto uživatelské nastavení uvedené v preambuli databáze do pomocného souboru `jazyky.uzv`.

```
87  \newwrite\nastaveniJazyky
```

Nyní definujeme makro `\jazyky`, které nastaví `\jazykytrue` a v případě, že aktuální volba třídy je `databaze`, je uvedené nastavení zapsáno do pomocného souboru. Ten později při volbě `pisemka` načteme, viz řádek 312.

```
88  \def\jazyky#1 {%
89    \jazykytrue
90    \ifdatabaze
91      \otevri\nastaveniJazyky jazyky.uzv
92      \zapis\nastaveniJazyky{\nx\jazyky #1}%
93      \zavri\nastaveniJazyky
94    \fi
```

Dále spočítáme, jaký počet lomítek uživatel zapsal za `\jazyky`. Pokud je počet lomítek nulový, ukládáme uvedený argument jako `\JedinyJazyk` a nastavujeme počet jazyků na 1.

```
95    \PocetVyskytu[/,\PocetLomitek]#1\UkonciPocitadlo
96    \ifnum\PocetLomitek=0
97      \def\JedinyJazyk{#1}%
98      \PocetJazyku=1
```

Pokud je v argumentu uvedeno alespoň jedno lomítko, bude s argumentem `#1` doplněným o tokeny `,/,` vykonán příkaz `\xjazyky` (viz řádek 107). Pomocí příkazu `\xseznam` (viz řádek 118) se vytvoří seznam zadaných jazyků z částí argumentu `#1` před lomítky a uloží se do `\SeznamJazyku`. Takto vytvořený seznam bude později použit při načtení jazyků v balíku `babel`.

```
99    \else
100     \xjazyky#1,/,
101     \def\SeznamJazyku{\xseznam#1,/,}
```

Pokud je počet uvedených jazyků v deklaraci `\jazyky` roven 1 (s právě jedním uvedeným lomítkem), ukládáme uvedenou jazykovou zkratku za lomítkem jako `\JedinyJzk` a nastavujeme jazykovou specializaci pomocí makra `\jazyk` automaticky na uvedený jazyk.

```
102     \ifnum\PocetJazyku=1
103       \def\JedinyJzk{\ZaLomitkem #1, }
104       \ea\jazyk\JedinyJzk
105     \fi
106   \fi}
```

Mechanismus makra `\xjazyky` zmíněného dříve spočívá v postupném načítání položek oddělených čárkou a uvedených za tímto makrem. V každém cyklu makra se do argumentu načte text až po nejbližší čárku a dále s ním pracuje. Pokud je načteným argumentem smluvně stanovená značka (zde lomítko `/`), je činnost makra ukončena uložením počtu uvedených jazyků do `\PocetJazyku`.

```
107 \def\xjazyky#1,{%
108   \ifx/#1%
109     \PocetJazyku=\the\PoradiJazyku
```

Není-li argumentem lomítko, považujeme jej za jednu z jazykových deklarací, navýšíme hodnotu čítače `\PoradiJazyku` o 1 a vykonáme s aktuálním argumentem příkaz `\PridejJazyk` (viz řádek 124). Speciálně při prvním cyklu makra (`\PoradiJazyku=1`) ukládáme jazykovou zkratku prvního jazyku jako `\PrvniJzk`.

```
110   \else
111     \PridejJazyk#1/%
112     \advance\PoradiJazyku by 1
113     \ifnum\PoradiJazyku=1
114       \def\PrvniJzk{\ZaLomitkem #1, }
115     \fi
116     \ea\xjazyky
117   \fi}
```

Dříve použité makro `\xseznam` pracuje se seznamem jazykových deklarací uvedených za `\jazyky` a načítá do svého argumentu položku až do nejbližší čárky. Z této dílčí deklarace odstraní lomítko a část za ním a připojí čárku. Funkce

makra je ukončena, pokud je načteným argumentem smluvní lomítko, což se stane až na konci seznamu, kde je za tímto účelem přidáno `,/,` (viz řádek 101).

```
\def\xseznam#1,{%
  \ifx/#1%
  \else
    \PredLomitkem #1,,%
    \ea\xseznam
  \fi}
```

Makro `\PridejJazyk` uvedené na řádku 111 má dvě funkce. První funkcí je přiřazení názvu jazyku jeho zkratce (tj. `#2` $\mapsto$ `#1`). Vzhledem k tomu, že uživatel bude zadávat jazykovou verzi pomocí zkratky jazyku, je tento přístup výhodný. Druhou funkcí je definice makra jazykové verze, jehož název je identický s uváděnou zkratkou jazyku. Ve fázi uvedení deklarace `\jazyky` pracují všechna tato makra naprázdno, tj. nic se nevytiskne. Ve finálním dokumentu budeme totiž požadovat zobrazení právě jedné jazykové verze, která bude určena až specifikací `\jazyk`. Makro pro příslušnou jazykovou verzi pak postačí předefinovat.

```
\def\PridejJazyk#1/#2/{%
  \ea\def\csname jazyk:#2\endcsname{#1}%
  \ea\def\csname #2\endcsname##1{}}
```

Následuje definice makra `\jazyk`, které umožní uživateli specifikovat aktuální zobrazený jazyk. Uvedením této specifikace nastavujeme `\jazyktrue` a také definujeme `\AktivniJzk` jako zkratku jazyku načtenou do argumentu. Toto uložení využíváme na řádku 169 při načítání odpovídajících vzorů dělení slov.

```
\def\jazyk#1 {%
  \jazyktrue
  \def\AktivniJzk{#1}%
```

Pokud odpovídá uvedená zkratka některému jazyku uvedenému v `\jazyky`, předefinujeme příslušné makro pro zápis jazykové verze tak, aby se vysázel jeho argument. Tím je zaručeno, že se zobrazí pouze aktuálně zvolená jazyková verze. Ostatní makra jazykových verzí totiž stále pracují naprázdno.

```
  \ifcsname jazyk:#1\endcsname
    \ea\def\csname #1\endcsname##1{##1}%
```

Pokud uživatelem uvedený jazykový identifikátor nekoresponduje se žádným jazykem uvedeným v `\jazyky`, třída ohlásí při kompilaci chybu.

```
  \else
    \ifjazyky
      \ClassError{ngt}{Neznama jazykova specifikace ‘#1’}{}%
    \fi
  \fi}
```

4.1.5. Předefinování \begin{document}

V následujícím zapíšeme před, resp. za \begin{document}, potřebné nastavení. Za tímto účelem předefinujeme původní příkaz \document definovaný v latex.ltx. Ekvivalent původního \document si pracovně označíme pro pozdější použití jako \origdocument.

```
\let\origdocument\document
\def\document{%
  \endgroup
```

Pokud uživatel použil deklaraci \jazyky s více než jedním jazykem, ale neuvedl specifikaci \jazyk, třída ngt vygeneruje informativní upozornění o automatickém načtení prvního jazyku v deklaraci. Příslušné načtení jazykových vzorů provádíme na řádku 174, tj. za \origdocument.

```
  \ifjazyky
    \ifjazyk
    \else
      \ifnum\PocetJazyku>1
        \ea\jazyk\PrvniJzk
        \ClassWarning{ngt}{%
          Nastaveno \string\jazyk\space `\PrvniJzk'}%
      \fi
    \fi
  \fi
```

Dále načítáme balík babel v závislosti na tom, v jakém formátu byla zadána jazyková deklarace v \jazyky.

```
  \ifjazyky
    \ifnum\PocetLomitek=0
      \usepackage[\JedinyJazyk]{babel}%
    \else
      \usepackage[\SeznamJazyku]{babel}%
    \fi
```

V případě, že uvádíme specifikaci \jazyk bez uvedení deklarace \jazyky, je ohlášena chyba kompilace.

```
  \else
    \ifjazyk
      \ClassError{ngt}{Doplnte deklaraci \string\jazyky}{}%
```

Pokud neuvedeme ani specifikaci \jazyk, je načtena čeština.

```
    \else
      \usepackage[czech]{babel}
    \fi
  \fi
```

V části za `\begin{document}` potom načítáme odpovídající vzory pro dělení slov. Pro celkový počet jazyků roven 1 nemá význam uvádět `\selectlanguage`.

```
\begingroup\origdocument
\ifjazyky
  \ifjazyk
    \ifnum\PocetJazyku=1
    \else
      \ea\ea\ea\selectlanguage
      \ea\ea\ea{\csname jazyk:\AktivniJzk\endcsname}
    \fi
  \else
    \ifnum\PocetJazyku>1
      \ea\ea\ea\selectlanguage
      \ea\ea\ea{\csname jazyk:\PrvniJzk\endcsname}
    \fi
  \fi
\fi}
```

4.1.6. Interní `0/1` přepínače

Kromě jazykového nastavení bude potřeba při volbě `databaze` a především `pisemka` nastavit další parametry sazby. Ty pro přehlednost omezíme na formát, kdy za klíčovou sekvenci uvádíme hodnoty `0` (ne) nebo `1` (ano). Definujeme proto zástupné sekvence `\etiam` a `\nihil`[9] pomocí

```
\def\etiam{1}
\def\nihil{0}
```

Makro `\OIprepinac` definuje popsaný druh ovladače. Do svých dvou povinných argumentů načítá postupně název přepínače a iniciální předvolenou hodnotu.

```
\def\OIprepinac#1/#2 {%
  \def\init{#2}%
```

Po uložení iniciální hodnoty do makra `\init` zavádíme novou logickou proměnnou obsahující název přepínače. V závislosti na hodnotě uložené v `\init` nastavujeme tuto hodnotu na `false`, resp. `true`. Pokud hodnota v `\init` není `0` ani `1`, bude ohášena chyba.[10]

```
  \ea\newif\csname if#1\endcsname
  \ifx\init\etiam
    \csname #1true\endcsname
```

[9] Bohužel nelze použít `\ano` a `\ne`, protože druhá by kolidovala s příkazem pro sazbu $\neq$. Volíme tedy neutrální latinské termíny.

[10] Tuto chybu nemůže uživatel bez editace souboru `ngt.cls` způsobit. Jedná se spíše o pojistku pro autora balíku v případě zavlečení překlepu.

```
    \else
      \ifx\init\nihil
        \csname #1false\endcsname
      \else
        \ClassError{ngt}{%
          Neocekavana inicialni hodnota ‘#2’ v ‘#1’}{}%
      \fi
    \fi
```

Současně připravíme koncovému uživateli makro, jehož název bude identický s názvem přepínače a bude načítat do argumentu jediný token (`0` nebo `1`), který si uložíme do `\OIhodnota`. Ve shodě s touto hodnotou nastavujeme hodnotu příslušné logické proměnné, kterou může uživatel ovlivnit zobrazení výstupu v PDF. V případě, že uložená hodnota není `0` ani `1`, bude ohlášena chyba.

```
    \ea\def\csname #1\endcsname##1 {%
      \def\OIhodnota{##1}%
      \ifx\OIhodnota\etiam
        \csname #1true\endcsname
      \else
        \ifx\OIhodnota\nihil
          \csname #1false\endcsname
        \else
          \ClassError{ngt}{Neocekavana hodnota (#1)}{}%
        \fi
      \fi}}
```

V případě, že uživatel neuvede žádné nastavení, bude implicitně načteno

```
\OIprepinac vyres/1
\OIprepinac novy/1
\OIprepinac body/1
```

Tímto nastavením jsme současně definovali přepínače `\vyres`, `\novy` a `\body`.

4.1.7. Jiná pomocná makra a nastvení

Pro zvýšení přehlednosti kódu volíme v některých případech definice maker, jejichž koncovým separátorem je konec řádku. Tato sympatická praxe má své zastoupení třeba v systému OPmac P. Olšáka popsané v [2, s. 46]. Jeho makro `\eoldef` jsme pro naše účely převzali v nezměněné podobě.

```
\def\eoldef#1{%
  \def#1{\begingroup \catcode‘\^^M=12 \eoldefA#1}%
   \ea\def\csname\string#1:M\endcsname}
{\catcode‘\^^M=12 \gdef\eoldefA#1#2^^M{%
  \endgroup\csname\string#1:M\endcsname{#2}}}
```

Často je vhodné vědět, jaký je první token, který následuje za řídící sekvencí. Pro účely tohoto specifického rozdělení načítaného argumentu nějakého makra definujeme `\RozdelArgument`, které rozloží původní argument a uloží si první token, resp. zbylé tokeny, do odpovídajících maker.

```
\long\def\RozdelArgument#1#2***{%
  \def\PrvniToken{#1}%
  \long\def\ZbyleTokeny{#2}}
```

Posledním přípravným krokem bude případné načtení souboru `nastaveni.tex` s uživatelským nastavením (v případě, že existuje). Za tímto účelem přebíráme ještě jedno makro P. Olšáka (viz [3, s. 288]), `\softinput`.

```
\newread\testin

\def\softinput #1 {%
  \let\next\relax
  \openin\testin=#1
  \ifeof\testin
  \else
    \closein\testin
    \def\next{\input #1 }%
  \fi
  \next}

\softinput nastaveni.tex
```

Pro zajištění vertikálního odsazení mezi jednotlivými úlohami kromě prvních úloh v aktuální databázi nebo subdatabázi (tj. oddílu), definujeme logickou proměnnou `\ifoddil`.

```
\newif\ifoddil
```

Tu nastavíme na `true` na začátku sazby každé databáze a subdatabáze (viz řádky 251 a 261).

4.2. Nastavení pro volbu `databaze`

```
\ifdatabaze
```

4.2.1. Všeobecná nastavení

V této části jsou nastaveny základní parametry sazby a některé pomocné prostředky. Především zavádíme vertikální odsazení úlohy jako `\bigskip` a také související čítače pro číslování úloh a dílčích databází (sekcí).

```
  \def\OdsadUlohu{\vskip1.5em}
  \newcount\PoradiUlohy
```

```
\newcount\sekce
```

Pro zápis zadání (a pokud existují, i řešení) úloh deklarujeme identifikátor souboru pro zápis `\zadaniSoubor`. V souvislosti s tímto zápisem deklarujeme registry `\zadaniTokeny`, resp. `\reseniTokeny`, do nichž budeme ukládat jednotlivé definice příkazů pro zadání, resp. řešení úloh. Soubor, do něhož bude zápis proveden, nazveme `databaze.pkd`, kde `pkd` značí překódovaný. Cílem je totiž jednoznačně přiřadit každé položce databáze (úloze, případně řešení) uspořádanou trojici $(\texttt{X}, m, n)$, tj. zakódovat ji. Zde `X` označuje typ položky (`U` – úloha, `R` – řešení), číslo m pořadí databáze a n pořadí úlohy v rámci m-té databáze. Každou úlohu, nebo případně její řešení, si tak po zápisu těchto definic budeme moci zavolat pouze zadáním typu objektu `X` a čísel m, n.

```
\newwrite\zadaniSoubor
  \newtoks\zadaniTokeny
  \newtoks\reseniTokeny
  \otevri\zadaniSoubor databaze.pkd
```

Aby nebyl zápis definic těžkopádný, připravíme si makro `\ZapisDefinici` se třemi parametry, identifikátorem souboru pro zápis, názvem makra a ukládaným obsahem.

```
\def\ZapisDefinici#1#2#3{%
  \zapis#1{\nx\ea\def\nx\csname#2\endcsname{#3}}}
```

Pro sazbu návěští úlohy nadefinujeme malý čtvereček, za nímž následuje pořadí databáze a pořadí úlohy v rámci databáze.

```
\def\ctverecek{%
  \pdfliteral{%
   q .7 g 0 0 6 6 re f Q}}

\setbox 0\hbox to 1em{\vbox to 1em{\vss\ctverecek}\hss}
\pdfxform 0
\edef\PDFctverecek{\the\pdflastxform}
\def\NavestiUlohy{%
  \leavevmode
  \llap{\pdfrefxform\PDFctverecek\hskip5pt}{%
    \bfseries\the\sekce.\the\PoradiUlohy}~~}

```

Následuje definice makra `\databaze`, které má dva povinné parametry oddělené lomítkem, koncovým separátorem je písmeno `b`. Pokud je hodnota čítače `\sekce` větší než nula, zapíšeme do souboru `databaze.pkd` počet úloh předchozí sekce jako obsah makra `\csname PocetUloh|<poradi_sekce>|\endcsname`. Počet úloh konkrétní dílčí databáze (tj. pořadí její poslední úlohy) tak provádíme hned na začátku zavolání nejbližšího následujícího `\databaze`, kdy hodnota čítače `\sekce` ještě není změněna.

```
\def\databaze#1/#2b{%
  \ifnum\sekce>0
    \ZapisDefinici\zadaniSoubor{%
      PocetUloh|\the\sekce|}{\the\PoradiUlohy}%
  \fi
```

Na začátku databáze vynulujeme čítač `\PoradiUlohy`, nastavíme `\oddiltrue` a sázíme `\section` známé z LaTeXu. Současně pokládáme hodnotu čítače `\sekce` rovnu hodnotě čítače `\thesection` číslující v LaTeXu sekce.

```
  \PoradiUlohy=0
  \section{#1}
  \oddiltrue
  \sekce=\arabic{section}%
```

Do souboru `databaze.pkd` si ještě uložíme počet bodů příslušející úlohám dané dílčí databáze (makro `\csname Body|<poradi_sekce>|\endcsname`).

```
  \ZapisDefinici\zadaniSoubor{Body|\the\sekce|}{#2}}
```

Pokud bude potřeba vytvořit jemnější strukturu dílčí databáze, definujeme pro tento účel makro `\subdatabaze`, které je klasickým `\subsection` z LaTeXu. Pokud je bezprostředně před zapsaným `\subdatabaze` uvedeno `\databaze`, je nastaveno ještě před sazbou subdatabáze `\oddiltrue`. V tomto případě nadpis subdatabáze od nadpisu databáze neodsazujeme. Při `\oddilfalse` naopak odsadíme nadpis subdatabáze (od předchozí úlohy).

```
\eoldef\subdatabaze#1{%
  \ifoddil\else\OdsadUlohu\fi
```

Po vysázení nadpisu subdatabáze nastavíme `\oddiltrue`.

```
  \subsection{#1}
  \oddiltrue}
```

4.2.2. Definice maker pro sazbu úloh a řešení

Při zápisu úlohy s řešením budeme zadání od řešení oddělovat separátorem `\reseni`. Při pozdějším zobrazení verze s výsledky nebo bez nich (ovladač `\vyres`) bude vhodné uložit si zvlášť část před, resp. za separátorem, jako `\xzadani`, resp. `\xreseni`.

```
\long\def\RozdelZadaniReseni#1\reseni#2***{%
  \long\def\xzadani{#1}%
  \long\def\xreseni{#2}}
```

Pro účely ignorování úlohy při sazbě databáze a při generování testu definujeme příkaz `\hvezda` jako token `*`.

```
\def\hvezda{*}
```

Nyní následuje definice důležitého makra `\uloha`. Nejdříve si rozdělíme argument (tj. zadání úlohy s případným řešením) na první token a zbylé tokeny a také v něm spočítáme počet výskytů tokenu `\reseni`, který ukládáme do čítače `\PocetReseni`.

```
\long\def\uloha#1***{%
  \RozdelArgument#1***
  \PocetVyskytu[\reseni,\PocetReseni]#1\UkonciPocitadlo
```

Pokud je první token `*`, činnost makra je ukončena a nic se nevysází. V opačném případě navýšíme čítač `\PoradiUlohy` o 1.

```
  \ifx\PrvniToken\hvezda
  \else
    \advance\PoradiUlohy by 1%
```

Současně vertikálně odsadíme každou úlohu, která nenásleduje bezprostředně po vysázení nadpisu databáze nebo subdatabáze. Pokud úloha následuje bezprostředně po uvedeném typu nadpisu, nastavujeme `\oddilfalse`, aby bylo zajištěno odsazení dalších úloh dané databáze nebo subdatabáze.

```
    \ifoddil
      \oddilfalse
    \else
      \OdsadUlohu
    \fi
```

Dále se činnost makra větví podle toho, kolikrát napsal uživatel do argumentu `#1` separátor `\reseni`. Pokud není uveden ani jeden separátor `\reseni`, uložíme celý argument do registru `\zadaniTokeny`, vysázíme návěští úlohy se zadáním, uložíme text zadání do makra `\csname U|<m>|<n>|\endcsname` a zapíšeme tuto definici do souboru `databaze.pkd`.

```
    \ifcase\PocetReseni
      \zadaniTokeny={#1}
      \NavestiUlohy #1\par
      \ZapisDefinici\zadaniSoubor{%
        U|\the\sekce|\the\PoradiUlohy|}{\the\zadaniTokeny}
```

Pokud je uveden právě jeden separátor `\reseni`, rozdělíme načtený argument na část `\xzadani`, resp. `\xreseni` a uložíme je do registrů `\zadaniTokeny`, resp. `\reseniTokeny`. Opět provádíme analogický kódovací zápis definic do souboru `databaze.pkd`.

```
    \or
      \RozdelZadaniReseni#1***
      \ea\zadaniTokeny\ea=\ea{\xzadani}
      \ea\reseniTokeny\ea=\ea{\xreseni}
      \ZapisDefinici\zadaniSoubor{%
```

```
290             U|\the\sekce|\the\PoradiUlohy|}{\the\zadaniTokeny}
291           \ZapisDefinici\zadaniSoubor{%
292             R|\the\sekce|\the\PoradiUlohy|}{\the\reseniTokeny}
```

Dále již sázíme návěští úlohy, a pokud uživatel nastavil ovladač `\vyres` na hodnotu `1` (tj. `\resenitrue`), sázíme také její řešení s příslušným odsazením a návěštím.

```
293           \NavestiUlohy\xzadani\par
294           \ifvyres
295             \OdsadReseni
296             {\footnotesize\NavestiReseni\xreseni}\par
297           \fi
```

Pokud uživatel zadal více než jeden separátor `\reseni`, bude tento stav vyhodnocen jako chyba.

```
298         \else
299           \ClassError{ngt}{%
300             V prostredi \string\uloha\space muzete pouzit
301             \string\reseni\space nejvyse jednou}{}%
302         \fi
303       \fi}
```

Zcela závěrem ještě zapíšeme počet úloh poslední uvedené databáze, uložíme počet databází a ukončíme zápis do souboru `databaze.pkd`.

```
304     \AtEndDocument{%
305       \ZapisDefinici\zadaniSoubor{%
306         PocetUloh|\the\sekce|}{\the\PoradiUlohy}
307       \zapis\zadaniSoubor{\PocetDatabazi=\the\sekce}
308       \zavri\zadaniSoubor}
```

4.3. Nastavení pro volbu `pisemka`

```
309   \else
```

4.3.1. Všeobecná nastavení

Kromě úpravy horního okraje pro hlavičku načítáme překódovanou databázi úloh a jazykové nastavení uvedené v řídicím souboru, což je v naší ukázce soubor `banka_uloh.tex` (pokud bylo nějaké uvedeno). Definujeme také odpovídající vertikální odsazení úlohy na písemce.

```
310     \advance\topmargin by -20pt
311     \input databaze.pkd
312     \softinput jazyky.uzv
```

```
313
314     \def\OdsadUlohu{%
315       \bigskip\bigskip}
```

4.3.2. Nastavení sazby datumu

Pro usnadnění zadávání datumu bez nutnosti jeho ručního stanovení definujeme několik pomocných maker. Nejdříve definujeme makro `\DnuVMesici`, které expanduje na počet dnů v daném měsíci.[11]

```
316     \def\DnuVMesici{%
317       \ifcase\month
318       \or 31
319       \or\ifnum\numexpr\year/4*4-\year=0 29 \else 28 \fi
320       \or 31 \or 30 \or 31 \or 30
321       \or 31 \or 31 \or 30 \or 31
322       \or 30 \or 31 \fi}
```

Nyní definujeme makro `\PrictiDny`, které zajistí úpravu aktuálního datumu vpřed nebo vzad podle hodnoty v argumentu `#1`.

```
323     \def\PrictiDny#1{%
324       \advance\day#1\relax
325       \ifnum#1>0
326         \OpravDatumVpred
327       \else
328         \OpravDatumVzad
329       \fi}
```

Makra `\OpravDatumVpred` a `\OpravDatumVzad` pracují obě na shodné bázi jako cykly. Vzhledem ke skutečnosti, že makro `\PrictiDny` upravuje hodnotu čítače `\day`, je nutno v závislosti na přičítaném počtu dnů a na počtu dnů v jednotlivých měsících zjistit, jaký bude po úpravě měsíc (čítač `\month`) a rok (čítač `\year`).

```
330     \def\OpravDatumVpred{%
331       \ifnum\day>\DnuVMesici
332         \advance\day by -\DnuVMesici
333         \advance\month by 1
334         \ifnum\month>12
335           \month=1
336           \advance\year by 1
337         \fi
338       \ea\OpravDatumVpred
339       \fi}
```

[11] Uživatelé třídy `ngt` v roce 2100 musí počítat s malým překvapením.

```
340
341     \def\OpravDatumVzad{%
342       \ifnum\day<1
343         \advance\month by -1
344         \ifnum\month<1
345           \month=12
346           \advance\year by -1
347         \fi
348         \advance\day\DnuVMesici
349         \ea\OpravDatumVzad
350       \fi}
```

Pro sazbu datumu připravíme uživateli makro `\Datum`, které v tuto chvíli definujeme jako `\today`.

```
351     \def\Datum{\today}
```

Pokud bude chtít uživatel upravit datum (tj. `\Datum`) na jiné, provede potřebné nastavení pomocí makra `\datum`, které načítá argument do konce řádku. Další činnost makra `\datum` spočívá ve specifickém předefinování `\Datum` s větvením podle prvního tokenu v zadaném argumentu.

```
352     \eoldef\datum#1{%
353       \RozdelArgument#1***
354       \gdef\Datum{%
```

Pokud je prvním tokenem `+`, provede se úprava datumu přičtením dnů podle hodnoty v `\ZbyleTokeny`. Pokud je prvním tokenem `-`, provede se analogická úprava odečtením.

```
355         \if\PrvniToken+
356           {\PrictiDny{\ZbyleTokeny}\today}%
357         \else
358           \if\PrvniToken-
359             {\PrictiDny{-\ZbyleTokeny}\today}%
```

Pokud není první token ani `+` ani `-`, vysází se původní argument.

```
360           \else#1%
361           \fi
362         \fi}}
```

4.3.3. Hlavička písemky

Vzhledem k okolnostem plynoucím z předpokládané vícejazyčné sazby dokumentu je vhodné textové popisky v hlavičce redukovat na minimum. Namísto jména a příjmení budeme sázet ikonu osoby příkazem `\JmenoIkona`. Analýzu kódu v `\pdfliteral` přenecháváme opět za cvičení.

```
\def\JmenoIkona{%
  \pdfliteral{%
   q 1 G .5 g 2 w 0 0 m
    0  5  5 10 10 10 c
   15 10 20  5 20  0 c h B
   .3 G .4 w 30 0 m
   200 0 l s
   .7 G .7 g 10 10 m
    4 10    5 17.5 10 17.5 c 10 17.5 m
   15 17.5 16 10   10 10   c h f Q}}
```

Protože jméno a příjmení nejsou vždy jednoznačným identifikátorem studenta, je často nutné požadovat jeho osobní číslo nebo podobný údaj. Návěští pro tento údaj bude zastoupeno specifickou ikonou sázenou příkazem `\OsobniCisloIkona`.

```
\def\OsobniCisloIkona{%
  \pdfliteral{%
   q .5 G .5 g 2 w
   2.5 0 m  5   10 l s
   6   0 m  8.5 10 l s
    .7 3 m 10    3 l s
   1.5 7 m 10.7  7 l s Q}}
```

Sazba hlavičky pomocí `\Hlavicka` je ovlivněna nastavením ovladače pro sazbu výsledků (`\vyres`). Při `\vyres 1`, tj. verze zpravidla pro pedagoga, není nutno sázet identifikační ikony. Ostatní prováděné úkony sazby hlavičky jsou zřejmé.

```
\def\Hlavicka{%
  \ifvyres
  \else
    \hrule height 1.5pt
    \par\vskip30pt
    \leavevmode\JmenoIkona
    \par\medskip
    \leavevmode\OsobniCisloIkona\hskip10pt\OsobniCisloIkona
  \fi
  \hfill
  \Datum\medskip\par
  \hrule\par\vskip 2pt
  \hrule height 1.5pt\bigskip}%
```

Příkaz `\hlavicka` pro úpravu hlavičky dokumentu pracuje v závislosti na prvním tokenu. Pokud je jím token `0`, chová se `\hlavicka` jako `\relax`. V opačném případě se chová jako `\xhlavicka #1`, který pouze předefinuje `\Hlavicka`.

```
\def\hlavicka#1{%
  \ifx#10
```

```
395         \let\Hlavicka\relax
396       \else
397         \xhlavicka#1%
398       \fi}
399
400     \long\def\xhlavicka#1***{%
401       \def\Hlavicka{{#1}}%
```

4.3.4. Zobrazení bodových zisků

Bodové zisky uložené v souboru `databaze.pkd` jsou obecně rozdílné pro každou úlohu. Zavedeme proto pomocný čítač `\PoradiDatabaze`, pomocí nějž budeme indexovat bodové zisky v závislosti na pořadí aktuální databáze.

```
402     \newcount\PoradiDatabaze
```

Nejdříve vytvoříme box `\MujBox`, do nějž vložíme bodový zisk. Dále definujeme makro `\BodyBox`, které změří šířku boxu obsahujícího bodový zisk (registr `\sirka`) a horizontálně jej vycentruje na aktuální pozici sazby.

```
403     \newdimen\sirka
404
405     \def\BodyBox{%
406       \hbox to 0pt{%
407         \hss\scriptsize\sffamily
408         \csname Body|\the\PoradiDatabaze|\endcsname\hss}}
```

Protože předpokládáme možnost sazby testu v libovolném jazyce, vyhýbáme se opět textovým popiskům a využíváme vlastní ikony. Pro účely zobrazení bodů definujeme makro `\BodyIkona`. To vykreslí jednoduchý měšec, do kterého je umístěn bodový zisk. Nakonec bodový zisk s ikonou vytiskneme vlevo od aktuálního bodu sazby.

```
409     \def\BodyIkona{%
410       \llap{%
411         \pdfliteral{%
412          q .8 0 0 .8 0 0 cm
413         .85 g .5 G .7 w -10 0 m
414         -10 5 -3 12 0 12 c
415         3 12 10 5 10 0 c
416         10 0 10 -5 0 -5 c
417         -10 -5 -10 0 -10 0 c B
418         -2 11 m -5 15 l
419         0 13 0 18 5 15 c
420         2 11 l B
421         .2 G 1.5 w -2.5 11 m
```

```
422         0 11.5 0 11.5 2.5 11 c B Q}%
423         \BodyBox\hskip 2.5em}}
```

4.3.5. Návěští úlohy a některé pomocné prostředky

Návěští úlohy v písemném testu definujeme jako tučně vysázené pořadové číslo úlohy.

```
424     \def\NavestiUlohy{%
425       \leavevmode
426       \ifbody
427         \BodyIkona
428       \fi
429       {\bfseries\the\PoradiDatabaze. }}
```

Při generování testu bude potřeba vytisknout jednotlivé úlohy a v případě povolených řešení i tato řešení. Oba typy dat jsme již dříve zapsali do souboru `databaze.pkd` a uložili do maker typu `\csname X|<m>|<n>|\endcsname`, kde `X` značí buď `U` nebo `R`. Pro krátkost a přehlednost definujeme příkaz `\tiskni`, který zadáním specifikace `X|<m>|<n>|` vytiskne takový objekt.

```
430     \def\tiskni #1 {%
431       {\csname #1\endcsname}\par}
```

Pokud uživatel nastaví ovladač vypnutí náhodného generování na `\novy 0`, bude při příští kompilaci potřeba znát náhodnou volbu úloh z minulé kompilace. Proto definujeme příkaz `\ZapisPamet`, který takový zápis do paměti provádí. Objekt typu `X|<m>|<n>|` bude pracovně uložen v souboru s příponou `pmt` jako `XP|<m>|` (v případě `X` = `R` pouze tehdy, pokud je uveden v souboru `databaze.pkd`, viz `\ifcsname` na řádku 435). Pro příslušný zápis deklarujeme odpovídající identifikátor souboru `\pametSoubor`.

```
432     \newwrite\pametSoubor
433
434     \def\ZapisPamet #1|#2|#3|{%
435       \ifcsname #1|#2|#3|\endcsname
436         \zapis\pametSoubor{%
437           \nx\ea\def\nx\csname #1P|#2|\endcsname{%
438             \nx\csname #1|#2|#3|\endcsname}}
439       \fi}
```

Nyní zavádíme čítač `\NahodneCislo` a definujeme makro `\generujNC` generující pseudonáhodné číslo v daném rozsahu a uloží ho do čítače `\NahodneCislo`. Horním ohraničením rozsahu pro náš náhodný výběr je zřejmě počet úloh dané databáze.

```
440     \newcount\NahodneCislo
441
442     \def\generujNC{%
```

```
443       \NahodneCislo=\pdfuniformdeviate
444         \csname PocetUloh|\the\PoradiDatabaze|\endcsname
445       \advance\NahodneCislo by 1}
```

4.3.6. Generování testu

Pro náhodné vygenerování testu definujeme příkaz `\generujTEST`. Nejprve pokládáme `\PoradiDatabaze=1` a v případě, že uživatel nastavil `\vyres 0`, bude vypnuto číslování stránek dokumentu (předpokládáme, že zadání testu je na jediné straně, zatímco při zobrazení řešení může být těchto stran více). Následně sázíme hlavičku testu.

```
446     \def\generujTEST{%
447       \PoradiDatabaze=1
448       \ifvyres
449       \else
450         \thispagestyle{empty}
451       \fi
452       \Hlavicka
```

Pokud je nastaveno `\novy 1`, otevřeme paměťový soubor pro zápis aktuálního náhodného výběru. Pokud je nastaveno `\novy 0`, paměťový soubor naopak načítáme (předpokládáme, že byl vytvořen při předchozí kompilaci).

```
453       \ifnovy
454         \otevri\pametSoubor vyber.pmt
455       \else
456         \input vyber.pmt
457       \fi
```

Nyní spouštíme cyklus přes všechny databáze a provádíme náhodný výběr jednotlivých úloh.

```
458       \loop\ifnum\PocetDatabazi<\PoradiDatabaze
459       \else
460         \OdsadUlohu
```

Při `\novy 1` generujeme náhodné číslo uložené v čítači `\NahodneCislo`, a to v mezích daných počtem úloh aktuální databáze. Dále tiskneme návěští i vylosovanou úlohu a zapisujeme zadání úlohy a případně i řešení do paměťového souboru.

```
461         \ifnovy
462           \generujNC
463           \NavestiUlohy
464           \tiskni U|\the\PoradiDatabaze|\the\NahodneCislo|
465           \ZapisPamet U|\the\PoradiDatabaze|\the\NahodneCislo|
466           \ZapisPamet R|\the\PoradiDatabaze|\the\NahodneCislo|
```

Pokud je nastaveno `\vyres 1` a řešení vylosované úlohy existuje, vytiskneme toto řešení s ostatními náležitostmi.

```
          \ifvyres
            \ifcsname%
              R|\the\PoradiDatabaze|\the\NahodneCislo|\endcsname
              \OdsadReseni
              {\footnotesize\NavestiReseni
               \tiskni R|\the\PoradiDatabaze|\the\NahodneCislo|}
            \fi
          \fi
```

Pokud je nastaveno `\novy 0`, tiskneme úlohu dané dílčí databáze uloženou při předchozím náhodném výběru. Pokud je navíc uvedeno `\vyres 1`, sázíme při existujícím řešení i toto řešení.

```
        \else
          \NavestiUlohy
          \tiskni UP|\the\PoradiDatabaze|
          \ifvyres
            \OdsadReseni
            \ifcsname RP|\the\PoradiDatabaze|\endcsname
              {\footnotesize\NavestiReseni
               \tiskni RP|\the\PoradiDatabaze| }
            \fi
          \fi
        \fi
```

Nyní navyšujeme proměnnou `\PoradiDatabaze` a cyklus opakujeme až do splnění podmínky `\PoradiDatabaze=\the\PocetDatabazi`.

```
        \advance\PoradiDatabaze by 1
      \repeat
```

Pokud je nastaveno `\novy 1`, ukončíme ještě na konci zápis do paměťového souboru.

```
      \ifnovy
        \zavri\pametSoubor
      \fi}\fi
```

Odkazy

1. HÀN THÉ, Thành et al. The pdfTEX user manual [online]. 2018 [cit. 2020-03-24]. Dostupné z: `http://texdoc.net/texmf-dist/doc/pdftex/manual/pdftex-a.pdf`.
2. OLŠÁK, Petr. *TEX pro pragmatiky.* CSTUG, 2016.
3. OLŠÁK, Petr. *TEXbook naruby.* Konvoj, 2001.

Summary: Multilingual pseudorandomly generated tests from databases

The aim of this paper is the description of the class `ngt.cls` that was created to simplify the preparation of written tests for educators with common user knowledge of LaTeX. The described simplification consists mainly in pseudorandom generation of tests from a prepared database of problems. Further advantages of the created system include the ease of the control for the end user and the possibility of creating a version with or without results. Writing the problems in the database file is designed to work with any number of language versions in a single source file with easy switching between them.

Marian Genčev, VŠB – Technická univerzita Ostrava