Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Programovací jazyk pro podporu výuky algoritmů

Diplomová práce

Autor: Petr Voborník
Informační management

Vedoucí práce: Doc. RNDr. Eva Milková, Ph.D.

Hradec Králové duben 2006

Prohlášení

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 9.4.2016 **Petr Voborník**

Poděkování

Tímto bych chtěl poděkovat vedoucí diplomové práce Doc. RNDr. Evě Milkové, Ph.D. za odbornou pomoc při tvorbě mé diplomové práce.

Anotace

Tato diplomová práce na téma „Programovací jazyk pro podporu výuky algoritmů" je zaměřena na tvorbu výukového programu, který se již v současné době z části využívá při výuce předmětu Algoritmy a datové struktury na Fakultě informatiky a managementu Univerzity Hradec Králové.

Jako nástroj pro vytvoření programu bylo zvoleno vývojové prostředí Borland Delphi 7, které umožňuje relativně rychlý vývoj podobných aplikací. Prostředí zahrnuje většinu základních vizuálních i nevizuální komponent použitých k vývoji této aplikace, přičemž k realizaci programu Algoritmy bylo použito i několik komponent nadstandardních.

Text předkládané diplomové práce obsahuje podrobný popis ovládání vytvořeného programu, poznatky získané při jeho vývoji a zkušenosti s psaním a zpracováváním algoritmů.

V programu, který je hlavní součástí práce, je kladen důraz na uživatelskou přívětivost, intuitivní ovládání a především na co nejvyšší míru možnosti jeho využití jak ve výuce problematiky algoritmů, tak při jejich samostatném studiu.

Anotation

The thesis on a given subject „Programming language for support of training algorithms" is focused on a training program creation which is partly used in subject Algorithms and data structure teaching at the Faculty of Informatics and Management of the University of Hradec Králové.

Borland Delphi 7 was selected as a tool for a program creating, because it makes a program development possible for similar applications. The environment includes majority parts of basic visual and non-visual components which are useable to a development of this application, if need be some above standard components were used for realization of program Algorithms.

The text of the thesis contains a detail description for a created program control, knowledge acquired over creating it and experience collection over writing and algorithms processing.

The main part of this degree work is the program, which is accentuated on user friendly interface, intuitive control and first of all on the top level usage over teaching and self-training algorithms.

Obsah

1. Úvod

Téma zvolené pro tuto diplomovou práci mě zaujalo již v době, kdy jsem se s předmětem Algoritmy a datové struktury setkal poprvé. Zdálo se mi totiž, že by bylo užitečné, aby studenti tohoto předmětu měli možnost nějakým způsobem vidět, jak vlastně algoritmy, které se zde učí vytvářet, fungují a probíhají, a aby měli při samostudiu nějakou kontrolu, že algoritmus, který napíší, je formálně i funkčně správný. Existující programovací jazyky by sice tyto funkce mohly zastat, leč nesetkal sem se dosud s žádným, jehož ovládání by bylo natolik snadné, že by jeho osvětlení výrazně nebrzdilo samotnou výuku algoritmů, který by umožňoval omezit své funkce pouze na základní probírané algoritmické příkazy, byl v češtině (či lokalizovatelný), podporoval definici pseudokódu a byl volně dostupný zdarma. Absenci takovéhoto nástroje jsem pak nejvíce pocítil ve chvíli, kdy jsem tento předmět doučoval své známé.

Jelikož při svém studiu zároveň pracuji jako programátor, je mi problematika algoritmů velmi blízká. Ve své diplomové práci jsem se tedy rozhodl zužitkovat své znalosti v tomto oboru a zaměřit se na vytvoření programu, který by mohl pomoci začátečníkům v programátorské oblasti a usnadnit jim osvojení si umění číst a psát algoritmy.

Jako vzor mi posloužilo programovací prostředí Borland Delphi, avšak výsledný program měl být výrazně jednodušší, podporovat pouze základní sadu příkazů a především freeware, tedy volně a zdarma dostupný všem zájemcům.

Obsahem mojí diplomové práce bylo tedy vytvoření programu v prostředí programovacího jazyka Borland Delphi 7, který by umožňoval psaní, kontrolu, spouštění a krokování algoritmů v definovatelném pseudokódu, jakožto pomůcku při výuce algoritmů. Samotný text diplomové práce pak je zaměřen na popis ovládání vytvořeného programu, zpracování poznatků při jeho vývoji a stručné osvětlení problematiky algoritmů.

Výsledný Program Algoritmy je již částečně využíván při výuce předmětu Algoritmy a datové struktury na Fakultě informatiky a managementu Univerzity Hradec Králové a to od zimního semestru 2005, jak v prezenční, tak v kombinované formě studia.

Program je neustále zdokonalován, aby plně vyhovoval potřebám tohoto předmětu. Další vývoj programu bude zaměřen i na oblast jeho využití odbornou veřejností. Za tímto účelem byly také zprovozněny stránky k tomuto projektu, které obsahují vždy aktuální informace o vývoji programu, je na nich k dispozici poslední verze programu Algoritmy volně ke stažení. Tyto stránky jsou dostupné každému zájemci na adrese algds.cronos.cz.

2. Glosář

Aktuální okno Okno aplikace, které je právě aktuální, čili vysvícené a uživatel pracuje s jeho prvky.

Build Číslo vydání programu v rámci jedné jeho verze.

Hit Počet unikátních načtení webové stránky. Unikátním načtením se rozumí načtení stránky jedním uživatelem minimálně jednou za 24 hodin. Hity se počítají pro každou stránku zvlášť.

Instance Konkrétní datový prvek v paměti počítače odvozený z nějakého vzoru, kterým může být buď datový typ nebo třída. Instance třídy je objekt dle terminologie objektově orientovaného programování a instance datového typu je datový objekt neboli data, s kterými pracujeme.

Matice Dvojrozměrné pole.

MDI okna *Podokna* aplikace přímo závislá na hlavním oknu, která se zobrazují pouze v rámci jeho plochy, mimo niž je nelze přesunout.

Modální okno *Podokno* aplikace, které po dobu své existence zablokuje přístup ke všem ostatním oknům aplikace.

Nadpříkaz Blokový příkaz algoritmu (příkaz cyklu či větvení), který pod sebou obsahuje jeden či více (blok) *podpříkazů*, pro které je pak tento nadpříkazem.

Originální kód Kód algoritmu napsaný v módu originálního kódu, kterým se rozumí kód s klíčovými slovy převzatými z jazyka Pascal. Jejich znění je neměnné.

Panel nástrojů Lišta pod hlavním menu, obsahující ikony, které zastupují příkazy z menu. Tyto panely nástrojů je možné libovolně přesouvat v rámci okrajů hlavního okna, tažením je uvolňovat do *plovoucích oken*, skrývat je či objevovat, měnit jejich obsah a vytvářet nové.

Plovoucí okno *Podokno* aplikace nepřímo závislé na hlavním okně. Takováto okna je možné v hlavním okně ukotvovat po jeho okrajích, lze je také tažením myši osamostatnit. I tak jsou ale tato okna automaticky skrývána, minimalizována, obnovována a zavírána s oknem hlavním.

Podokno Okno aplikace, které není hlavím oknem.

Podpříkaz Jeden či více (blok) příkazů, které jsou součástí cyklu či větve podmínky algoritmu, jejíž příkaz je pro tyto *nadpříkazem*.

Pole Strukturovaná proměnná, obsahující více hodnot. Pole je definováno svým názvem a hodnoty v něm pak indexem.

Posloupnost Jednorozměrné pole.

Posuvník Pokud se nevejde obsah okna či editoru do jeho okraji vymezeného prostoru, pak svislý (vpravo) a vodorovný (dole) posuvník umožňují posouvat aktuální zobrazení a shlédnout tak tento obsah po částech.

Pseudokód Kód algoritmu napsaný v módu pseudokódu, kterým se rozumí algoritmus s klíčovými slovy, která si uživatel může volitelně měnit.

Rezervovaná slova jsou slova vyhrazená programem pro definici algoritmů a nemohou tedy sloužit jako názvy proměnných a polí. Patří mezi ně především klíčová slova a konstanty.

RSS kanál Really Simple Syndication. RSS umožňuje publikování seznamu odkazů spolu s dalšími informacemi, které blíže popisují daný odkaz. Jedná se o data ve formátu XML, díky nimž specializované programy na čtení těchto formátů či internetové prohlížeče mají vždy aktuální přehled o měnících se částech všech takto sledovaných *webů*.

Schránka Schránka Windows je vyhrazený prostor v paměti, kam lze ukládat označený text (Ctrl+C) a později, případně i v jiné aplikaci, jej ze schránky opět vyzvednout a vložit na aktuální pozici (Ctrl+V).

Stavový řádek Spodní pruh v hlavním okně, který obsahuje nějaké informace týkající se aktuálního stavu aplikace (např. pozici kurzoru, stav změny v editoru apod.). Je-li hlavní okno v normalizovaném stavu (není minimalizováno ani maximalizováno), pak lze tomuto oknu tažením myši za pravý dolní roh stavového řádku měnit jeho velikost.

Stop-značka Breakpoint. Uživatelem umístěná značka na určitém řádku v editoru algoritmu, na kterou když se při svém běhu algoritmus dostane, tak se tento běh automaticky pozastaví.

Syntaxe Formální správnost zapsání algoritmu, na úrovni slov a znaků. Syntakticky správný algoritmus ještě nemusí být algoritmem správně funkčním.

Unit Programová jednotka v jazyce Pascal, umístěné v samostatném souboru s koncovkou *.pas*.

Výraz Aritmetický výraz, který lze početně vyhodnotit a získat tak jednu jedinou hodnotu.

Web Soubor internetových stránek vztahujících se ke stejnému místu a tématu. Adresy těchto stránek mají stejný začátek.

3. Popis programu a jeho ovládání

Nejprve se podíváme na program Algoritmy jako finální produkt z pohledu jeho uživatele a poskytneme mu tak úplný návod jeho ovládání, možností a úskalí.

3.1. Minimální požadovaná konfigurace

Program Algoritmy byl úspěšně testován na operačních systémech Windows XP a 2000, nicméně měl by bez problémů fungovat i ve starších verzích Windows (95, 98, ME). Program na pevném disku potřebuje necelých 5MB.

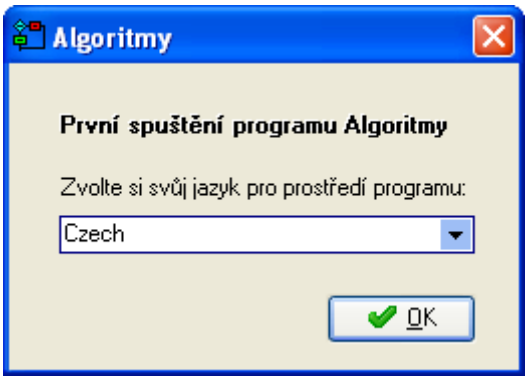
3.2. Instalace

Program Algoritmy má tu výhodu, že nepotřebuje žádnou instalaci. K jeho provozu v podstatě stačí samotný spustitelný soubor *Algoritmy.exe* a program je rovnou schopen samostatného fungování na kterémkoli počítači s Windows. Pro zprovoznění programu z internetu na cílovém počítači jej tedy stačí pouze stáhnout (viz kapitola 6. Stránky projektu, str. 67), rozbalit a spustit.

Program je sice schopen samostatného fungování jako pouhý jeden spustitelný soubor, ale je lepší u něho (ve stejném adresáři) mít i podadresář *Languages*, který obsahuje soubory s různými jazykovými verzemi programu, včetně oficiální verze české. Dále je s programem distribuován i podadresář *Algoritmy*, v němž je přiloženo několik ukázkových algoritmů.

3.3. První spuštění programu

Po prvním spuštění programu Algoritmy se nejprve otevře okno, kde má uživatel možnost zvolit si výchozí jazyk pro prostředí programu (viz Obr. 3.1). Toto okno se zobrazí pouze tehdy, v případě je-li v adresáři programu i podadresář *Languages*, z jehož obsahu je také sestavena nabídka tohoto okna.

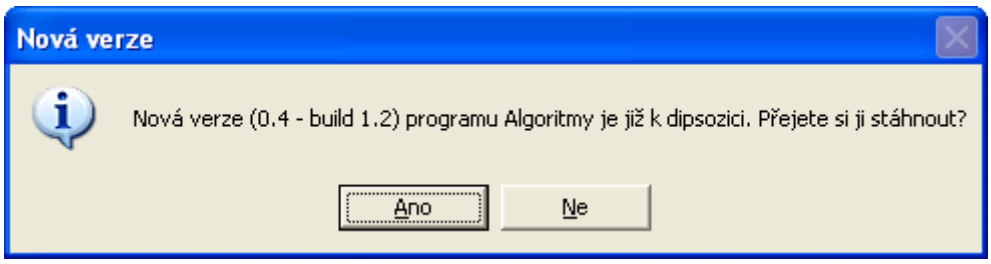


Obr. 3.1 - První spuštění programu

Jazyk, který si zde uživatel zvolí, je pak programu přednastaven a jsou z něho načtena i klíčová slova volitelného pseudokódu.

3.4. Update

Po každém spuštění programu je na pozadí provedena automatická kontrola, není-li na oficiálním webu k programu Algoritmy (viz kapitola 6. Stránky projektu, str. 67) již uveřejněna nová verze ke stažení. V případě že ano, je na to uživatel upozorněn dialogovým oknem (viz Obr. 3.2).



Obr. 3.2 - Upozornění na existenci nové verze programu Algoritmy

Pokud v tomto dialogu uživatel klikne na tlačítko Ano, otevře se mu internetový prohlížeč, který načte stránky projektu. Pokud klikne na Ne, dialog se uzavře a objeví se zase až při dalším spuštění programu Algoritmy.

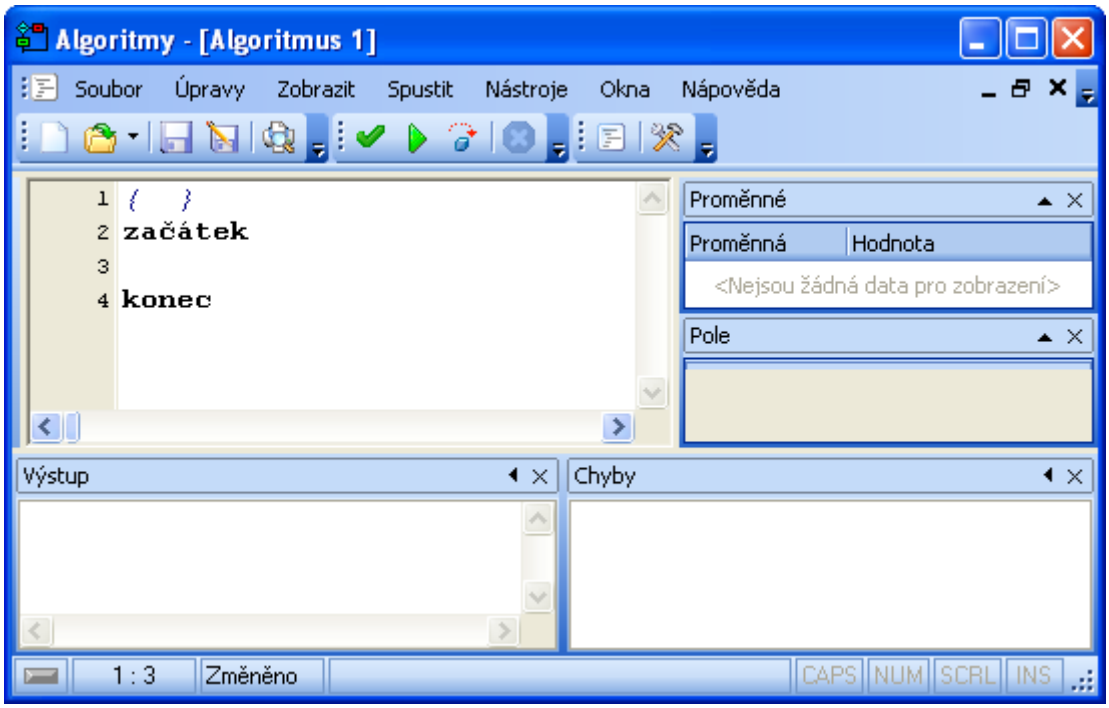
Není-li uživatel připojen k internetu, tato kontrola se neprovádí.

3.5. Prostředí programu

Program Algoritmy je tvořen hlavním oknem programu, které seskupuje MDI podokna s editory kódu algoritmu, dále pak plovoucími okny s výstupy programu a hlavním menu programu. V následujících podkapitolách si probereme základní zacházení s těmito prvky. Veškeré toto ovládání je intuitivní a podléhá nepsaným mezinárodním standardům pro práci s aplikacemi v prostředí Microsoft Windows.

3.5.1. Hlavní okno programu

Hlavní okno programu je nadřazené všem ostatním oknům aplikace (viz Obr. 3.3).



Obr. 3.3 – Hlavní okno programu

Je možné jej libovolně posouvat po obrazovce, či měnit jeho velikost a v něm ukotvená podokna se tomu automaticky přizpůsobí.

Úplně ve spodní části hlavního okna se nachází stavový řádek. Kromě toho, že tažením myši za jeho pravý dolní roh je možné měnit velikost hlavního okna, obsahuje i několik dalších částí.

V první z nich (zleva) je ikonka barevného indikátoru stavu programu. Ta může mít několik barev. Šedivá znamená, že program algoritmu je vypnut, purpurová, že program algoritmu je právě kontrolován, zelená, že program je spuštěn a běží a červená, že program je spuštěn, ale čeká na jeho určitém kroku (řádku), je tady pozastaven.

V další části stavového řádku (druhé zleva) je pak zobrazena pozice editačního kurzoru v aktuálním okně s editorem ve formátu *řádek : sloupec*. Třetí část je pak buď prázdná, nebo v ní je uvedeno, že kód v aktuálním okně s editorem byl uživatelem od posledního uložení změněn. Nejdelší třetí část stavového řádku je pak vyhrazena nápovědným textům, které se zde zobrazí vždy po najetí myši nad určitý objekt mající tuto vlastnost (např. ikony panelů nástrojů).

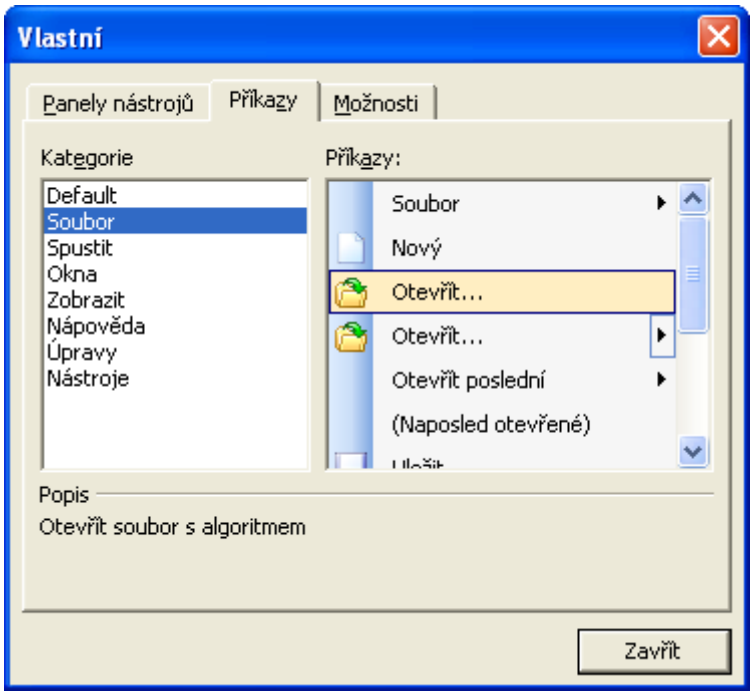
V poslední čtyřdílné části jsou pak indikátory stisknutých voleb na klávesnici, které ovlivňují psaní textu – kódu algoritmu. Jde o CAPS – Caps Lock (velká písmena), NUM – Num Lock (zapnutí číselné části klávesnice), SCRL – Scroll Lock a INS – Insert (přepis textu).

3.5.2. Ovládání menu a panelů nástrojů

Menu programu je naprosto dynamické a proto si jej každý uživatel může přizpůsobit dle svého vkusu a potřeb. Skládá se z hlavního menu (Soubor, Úpravy, Zobrazit, Spustit, Nástroje, Okna, Nápověda) a panelů nástrojů. Standardně je zobrazen pouze panel nástrojů Soubor, Spustit a Nástroje. Ostatní je možné zviditelnit zaškrtnutím přes hlavní menu Zobrazit – Panely nástrojů.

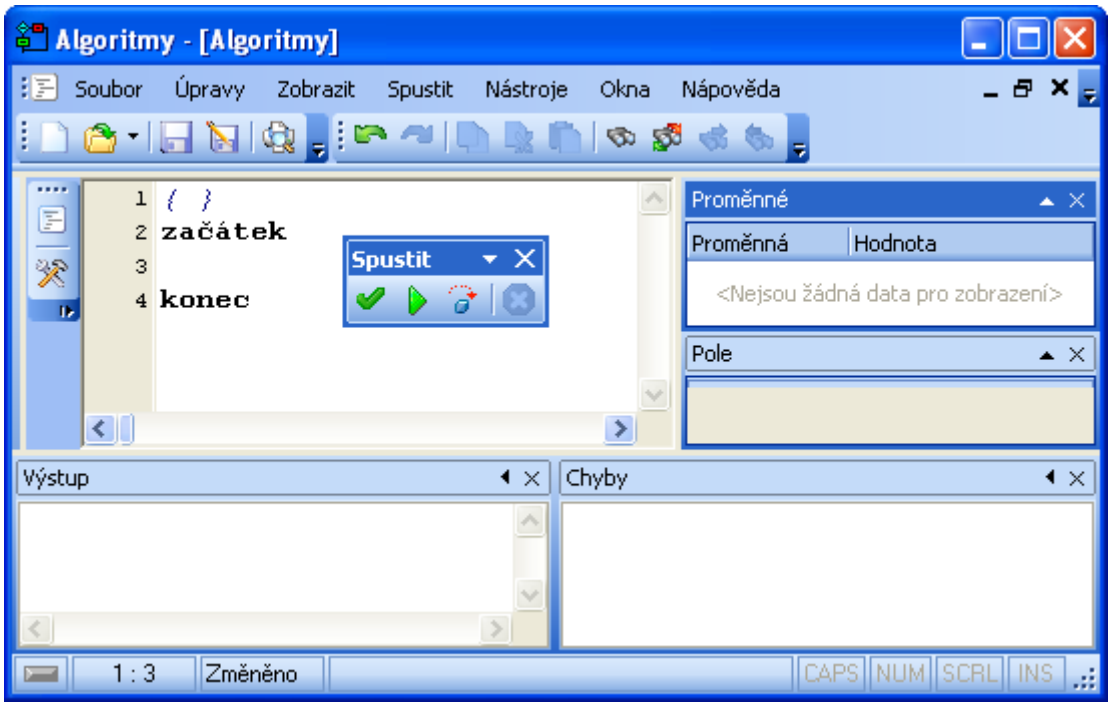
Všechny jednotlivé panely nástrojů je možné myší uchopit za jejich levý okraj a tažením za stále stisknutého levého tlačítka myši je přetáhnout na libovolné místo v okně i mimo něj, do samostatného plovoucího okna. Jednotlivé ikony panelů nástrojů je možno také skrývat či zobrazovat. Po kliknutí na pravý okraj příslušného panelu nástrojů a v zobrazeném vysouvacím menu lze zvolit požadované ikony.

Je také možné zcela měnit strukturu jednotlivých panelů nástrojů včetně hlavního menu, či vytvářet vlastní. Po kliknutí na menu Zobrazit – Panely nástrojů – Vlastní, nebo po kliknutí pravým tlačítkem myši do prázdného prostoru menu a volbou položky Vlastní.



Obr. 3.4 - Volitelné nastavení menu

V podokně, které se otevře (viz Obr. 3.4), lze vytvářet nové panely nástrojů (záložka Panely nástrojů), myší do nich (či z nich) přesouvat ikony (záložka Příkazy) ze všech kategorií, a v možnostech pak nastavit obecné chování celého menu.



Obr. 3.5 - Volitelné změny panelů nástrojů

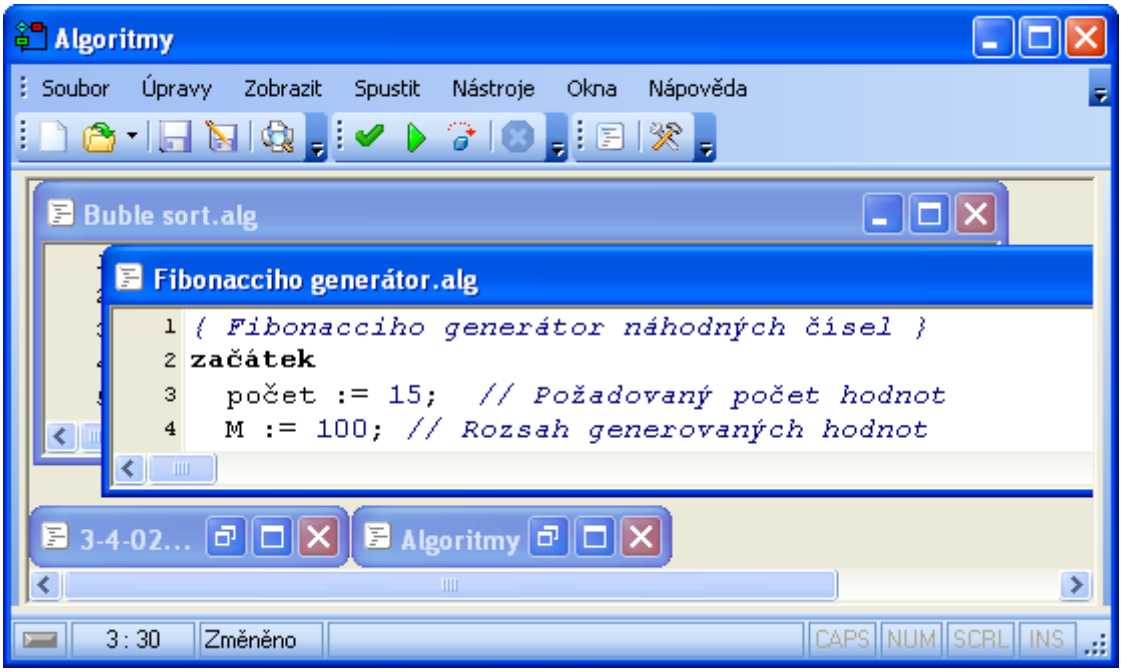
Všechny tyto změny v menu podléhají ukládání a načítání nastavení (viz kapitola 3.12.4 Ukládání nastavení, str. 28), takže budou zachovány i po vypnutí a opětovném spuštění programu.

3.5.3. Ovládání oken editoru

Okna s editorem jsou MDI, tedy „vnořená" do okna hlavního a nemohou být vytažena mimo jeho plochu. Ve výchozím stavu jsou všechna maximalizována do plochy hlavního okna, díky čemuž ani nemusí být na první pohled patrné, že jich může být nad sebou více současně.

Ovládací tři ikonky (minimalizace **–** , obnovení 🗗 a zavřít ✕) aktuálního okna se v případě, že je okno maximalizováno, nacházejí v pravém horním rohu, na stejné úrovni jako hlavní menu. Je-li okno minimalizováno či obnoveno (v normálním stavu), jsou tyto ikony standardně v jeho pravém horním rohu, případně je lze také nalézt v menu Okna – Aktuální okno.

Pokud některé z oken zasahuje mimo oblast hlavního okna, objeví se v hlavním okně posuvník, který umožní posunout aktuální pohled i na tuto skrytou část podokna. Minimalizovaná okna jsou umístěna ve spodní části plochy hlavního okna a jsou postupně řazena vedle sebe (viz Obr. 3.6).



Obr. 3.6 - Volitelné změny oken s editory kódu algoritmu

Úplný seznam všech oken se nachází v poslední části menu Okna, kde je vždy zaškrtnuto právě to aktuální a kliknutím na kterékoli z nich se na něj lze snadno přepnout.

Během krokování či běhu algoritmu je možnost přepínání se mezi okny vypnuta a aktivní zůstává vždy pouze to okno, jehož algoritmus byl spuštěn. Po skončení programu je tato možnost opět obnovena.

V menu Okna se též nacházejí tři možnosti, jak všechna okna s editory automaticky uspořádat na ploše. Jde o kaskádu čili uspořádání sestupně nad sebe, o uspořádání vodorovně nad sebe a horizontálně vedle sebe. Tyto volby fungují korektně pouze tehdy, dovoluje-li to počet oken a velikost plochy.

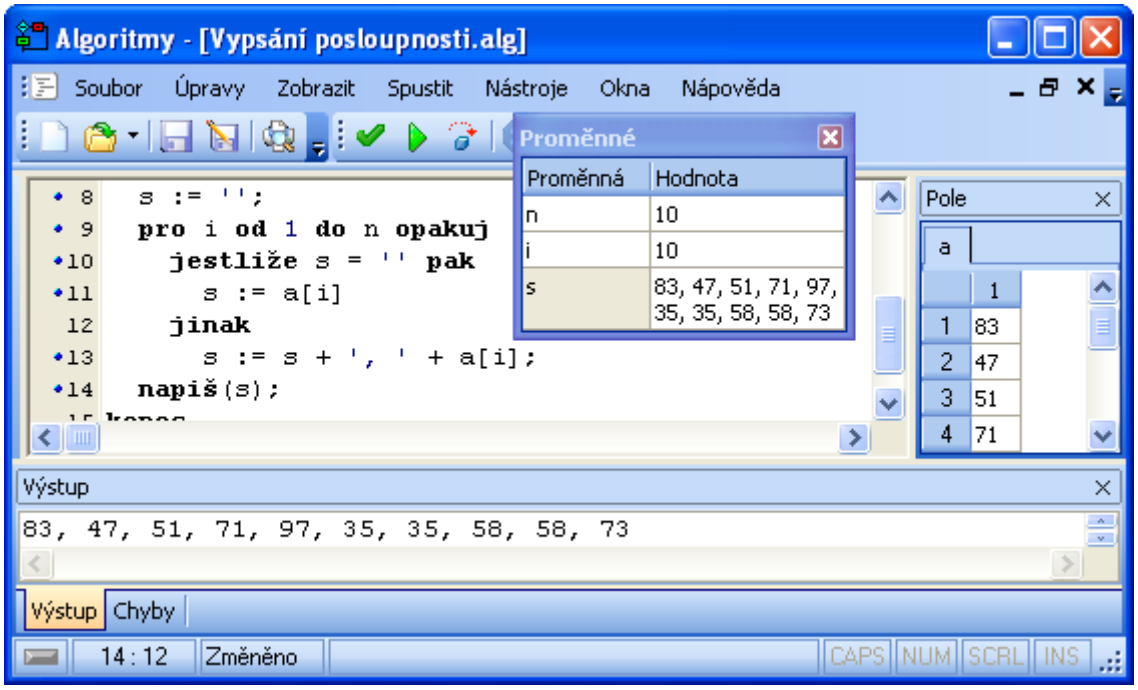
Pokud je aktuální okno s editorem maximalizováno, je název v něm editovaného algoritmu uveden v záhlaví hlavního okna v hranaté závorce (hned za názvem programu Algoritmy v aktuálním jazyce). Není-li toto okno maximalizováno, pak je název algoritmu uveden přímo v jeho záhlaví, které je v tomto stavu již viditelné.

3.5.4. Ovládání oken s výstupy

Okna s výstupy z programu jsou čtyři: Proměnné, Pole, Výstupy a Chyby. Jejich význam je podrobně popsán v kapitole 3.10 Okna s výstupy.

Tato okna se standardně nacházejí ukotvená po dvou v pravé a ve spodní části hlavního okna. S každým z nich však lze pracovat zcela individuálně. Každé je možné myší uchopit za horní lištu s jeho názvem a tažením jej uvolnit z ukotvení v hlavním okně. S těmito plovoucími okny je pak možné volně pohybovat po celé

ploše obrazovky, opět je ukotvovat na všech čtyřech okrajích hlavního okna a také je umísťovat vedle sebe nebo nad sebe či je seskupovat do sebe navzájem a listovat v nich pomocí záložek, které se v takovém případě objeví pod nimi. Za tyto záložky je možné je opět myší uchopit a přetáhnout je do samostatného plovoucího okna.



Obr. 3.7 - Volitelné změny oken s výstupy

Tato okna je také možné zavřít (skrýt) kliknutím na křížek v jejich horním pravém rohu, případně měnit jejich viditelnost v menu Zobrazit.

Všechny tyto změny v oknech s výstupy také podléhají ukládání a načítání nastavení (viz kapitola 3.12.4 Ukládání nastavení, str. 28), takže budou zachovány i po vypnutí a opětovném spuštění programu.

3.6. Práce se souborem

Za soubory jsou v tomto případě považovány samostatné textové soubory na disku obsahující kódy algoritmů spustitelné v programu Algoritmy. Takovéto soubory mají výchozí koncovku *.alg*. Vzhledem k jejich textové povaze není podmínkou, aby byly vytvořeny a napsány přímo v programu Algoritmy.

V programu lze zakládat nové soubory s algoritmy a otevírat již existující . Byl-li aktuální algoritmus otevřený či založený v programu Algoritmy zde změněn, je možné jej uložit do stávajícího souboru. Pokud algoritmus doposud nebyl nikdy uložen, zachová se volba Uložit stejně jako volba Uložit jako, a sice na-

bídne uživateli pomocí standardního dialogu možnost zvolit cílový soubor, do něhož bude algoritmus uložen.

Program Algoritmy si také eviduje, kterých posledních devět souborů bylo otevřených či uložených, a jejich názvy jsou pak seřazeny v menu Soubor – Otevřít poslední nebo pod šipečkou vedle ikonky Otevřít v panelu nástrojů. Kliknutím na kterýkoli ze souborů v seznamu se tento rovnou otevře.

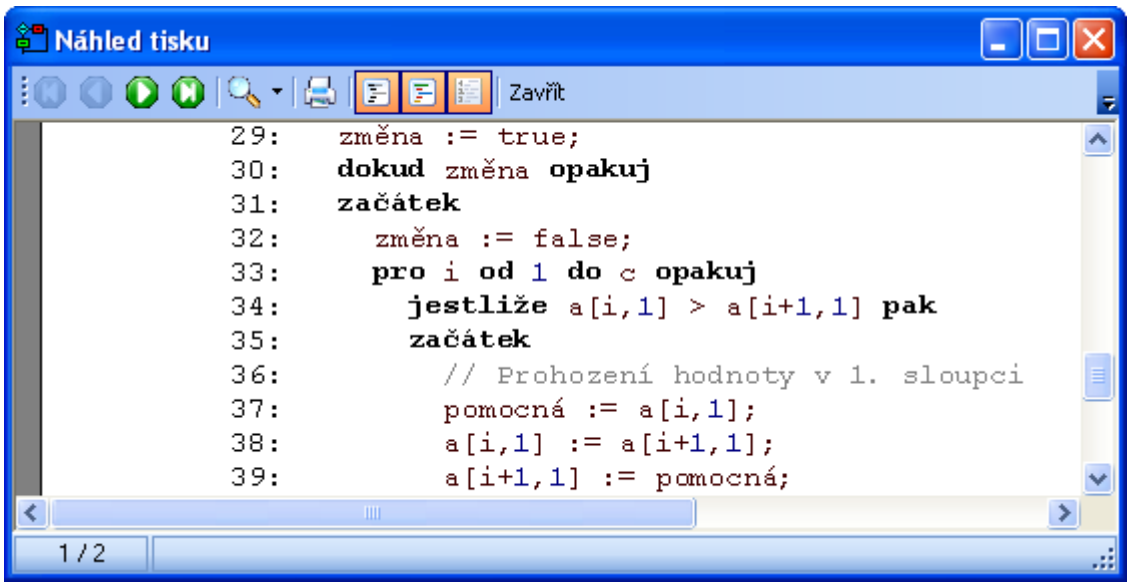
Soubor s algoritmem je vždy otevřen do nového podokna s editorem. Stejně tak tomu je i v případě založení souboru nového.

3.7. Tisk a náhled algoritmů

Algoritmy je samozřejmě možné i vytisknout na tiskárně. Tisk se týká vždy pouze algoritmu v aktuálním okně.

Tisknout je možné dvojím způsobem. Buď přímo odeslat požadovaný algoritmus na tiskárnu, nebo si nejprve zobrazit jeho náhled, tedy zobrazení, jak bude výsledná stránka či stránky vypadat.

Přímý tisk se provádí kliknutím na ikonku Tisk v menu Soubor. Náhled se pak spouští příkazem Náhled v témže menu, či rovnou na panelu nástrojů stejného jména. V případě tisku je uživatel již pouze dotázán standardním dialogovým oknem na cílovou tiskárnu s možnostmi jejího nastavení a po jeho potvrzení tlačítkem OK je zahájen tisk. V případě náhledu se otevře modální okno s náhledem tiskového výstupu (viz Obr. 3.8).



Obr. 3.8 - Náhled tisku

V horní části se nachází panel nástrojů obsahující několik voleb pro změnu zobrazení náhledu tisku. Je zde i několik nástrojů, pomocí kterých lze měnit výsled-

nou podobu tiskového výstupu. Jedná se o stylové zvýraznění syntaxe, barevné zvýrazněním syntaxe a tisknutí či netisknutí čísel řádků. Po zaškrtnutí či odškrtnutí těchto ikonek se v náhledu okamžitě projeví požadovaná změna.

Pomocí ikon umístěných hned vlevo je možné projít všechny stránky k tisku a zmenšit či zvětšit náhled, což již samotný tiskový výstup neovlivní.

Souhlasí-li uživatel se zobrazeným výstupem, který bude shodný s konečným tiskem, klikne zde na ikonu s tiskárnou, zvolí cílovou tiskárnu a klikne na tlačítko OK, které ihned zahájí tisk algoritmu.

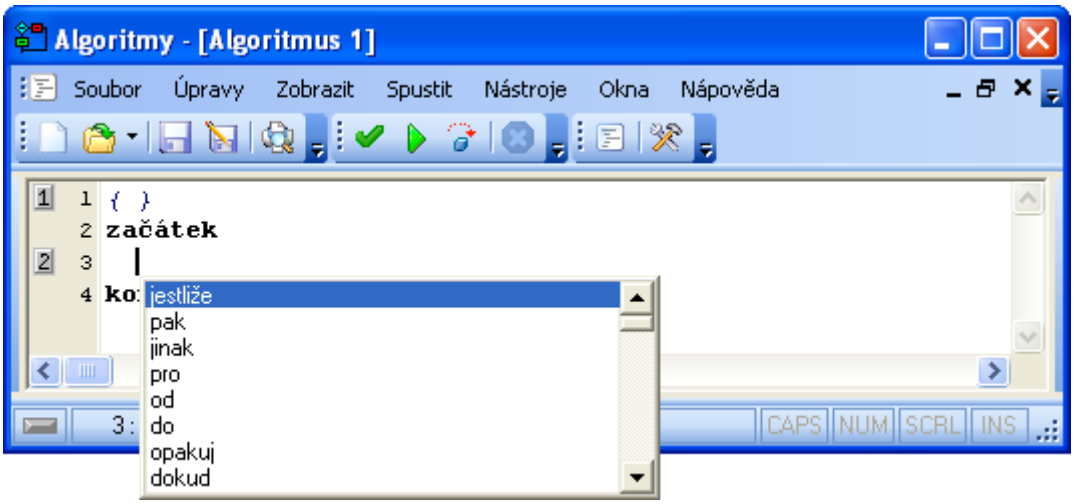
Tlačítko Zavřít zavře okno s náhledem, aniž by k samotnému tisku vůbec došlo.

3.8. Ovládání editoru

Hlavní okno programu pod sebou může mít více podoken s editory. Ovládání oken jako takových je popsáno v kapitole 3.5.3 Ovládání oken editoru. Nyní si přiblížíme možnosti psaní kódu algoritmu v editoru, který se nachází uvnitř těchto podoken.

3.8.1. Psaní kódu

Editor kódu algoritmu funguje téměř shodně jako běžný textový editor typu Poznámkový blok. Má však několik vylepšení, zaměřených především právě na zápis algoritmů.



Obr. 3.9 - Pomůcka při psaní klíčových slov a záložky

Zvýraznění syntaxe

Jedním z nich je zvýraznění syntaxe. To znamená, že klíčová slova, textové řetězce, komentáře a čísla jsou zvýrazněna (mají jinou barvu či styl písma) oproti

zbytku textu. Způsob zvýraznění těchto jednotlivých elementů si může uživatel libovolně nastavit (viz kapitola 3.12 Nastavení programu, str. 24), stejně jakož i barvu pozadí celého editoru.

Klíčová slova

Lze zde také využít pomocníka při pasní klíčových slov. Stačí stisknout kombinaci kláves Ctrl+Mezerník, a to ať již před začátkem psaní nového slova či během jeho psaní, a pod kurzorem se objeví seznam všech klíčových slov (viz Obr. 3.9) podporovaných programem algoritmy v aktuálním módu (viz kapitola 3.11.2 Módy programu, str. 24). Při následném nebo již započatém psaní jsou slova v seznamu filtrována dle již napsaného začátku slova a zůstávají v něm pouze slova mající stejný začátek jako ten napsaný uživatelem.

Klíčové slovo nacházející se v seznamu pak není bezpodmínečně nutné dopsat ručně, ale lze jej vložit do kódu dvojím kliknutím myši na něj, nebo jeho výběrem pomocí šipek nahoru či dolů a následným stiskem klávesy Enter či napsáním znaku oddělujícího slova (např. mezerník, závorka atd.).

Záložky

Další možností, jak se rychle orientovat v delším kódu, jsou záložky. Ty se vkládají vždy na aktuální řádek stisknutím kombinace kláves Ctrl+Shift+[číslo], kde číslem je myšlena číselná klávesa od 0 do 9. Vlevo se hned objeví ikonka s tímto číslem záložky ( - ). Stisknutím kláves Ctrl+[číslo] kdekoli v editoru se pak kurzor okamžitě přesune na řádek se záložkou zvoleného čísla.

Záložku lze zrušit stisknutím stejné kombinace kláves, jakou byla vytvořena (Ctrl+Shift+[číslo]), přičemž kurzor musí být na jejím řádku. Pokud je kurzor na řádku jiném, je tato záložka přesunuta sem.

Je také možné na jeden řádek přidat i několik záložek nad sebe. Záložky slouží pouze pro okamžitou orientaci v algoritmu a neukládají se s ním, takže po jeho zavření a opětovném otevření samy zmizí.

Odsazování

Editor algoritmů také zajišťuje, že ať již mu je nastaven jakýkoli druh písma, budou jednotlivé znaky vždy rozmístěny tak, aby byly vždy přesně nad sebou ve sloupcích. Díky této vlastnosti je možné kód algoritmu přehledně odsazovat mezerami ze začátku řádku tak, aby jednotlivé bloky algoritmu byly vždy na první pohled patrné a celková struktura algoritmu tak byla přehlednější. Jednotka odsazení bloku programu je zde stanovena na dvě mezery.

Editor sám při zahájení psaní nového řádku (Enter) odsadí tento nový řádek na úroveň toho předchozího. K vrácení se k této úrovni při posunu kurzoru o jakýkoli počet znaků (pouze však mezer) stačí vždy jeden stisk klávesy Backspace. Je totiž možné se v editoru libovolně pohybovat pouze pomocí šipek či myši i v místech, kde žádné znaky nejsou (pouze však po existujících řádcích). V případě, že zde uživatel pak něco napíše, editor automatiky doplní před tento znak mezery zleva.

Mezi další usnadnění patří možnost odsadit či zrušit odsazení celého bloku kódu najednou o jednu jednotku. Stačí označit požadované řádky algoritmu (Shift + šipky či myš) a stiskem kombinace kláves Ctrl+Shift+I označené řádky odsadit nebo Ctrl+Shift+U naopak těmto řádkům jedno odsazení umazat. Umazávány jsou pouze mezery zleva každého řádku, takže kód samotný není nijak ohrožen.

Ostatní

Oproti běžnému editoru jsou po levé straně editoru uvedena čísla řádků algoritmu pro snadnější orientaci v něm. Dále je zde vertikální čára za osmdesátým znakem, která nemá jiný než orientační význam. Kód může klidně pokračovat i za ní, ale vzhledem k jeho přehlednosti se to nedoporučuje.

Kopírování označeného textu algoritmu do schránky  , jeho vyjímání  i následné vkládání  jsou zde samozřejmostí. Taktéž je možné do editoru vložit text zkopírovaný do schránky kdekoli jinde v systému (Word, Notepad, Internet Explorer...).

Změny v kódu algoritmu lze vracet pomocí funkce Zpět  (Ctrl+Z) a to až o padesát kroků. Stejně tak je možné vrácení těchto změn i odvolat příkazem Odvolat zpět  (Shift+Ctrl+Z).

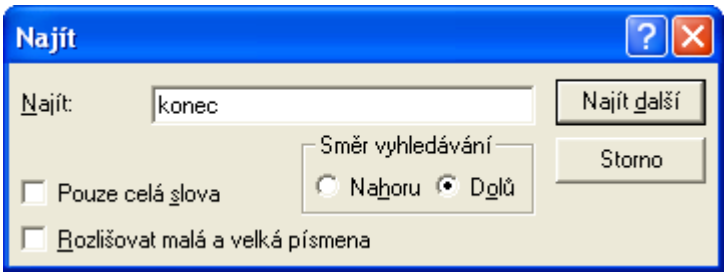
Odstranit aktuální řádek lze stiskem kombinace kláves Ctrl+Y.

3.8.2. Vyhledávání a nahrazování

V rámci jednoho editoru lze také vyhledávat a nahrazovat určité znaky či slova. Všechny tyto funkce je možno najít v menu Úpravy.

Vyhledávání

Vyhledávání slouží k rychlému nalezení požadovaného textu. Pomocí příkazu v menu  či stisknutím klávesové zkratky Ctrl+F se otevře standardní dialogové okno pro vyhledávání (viz Obr. 3.10).



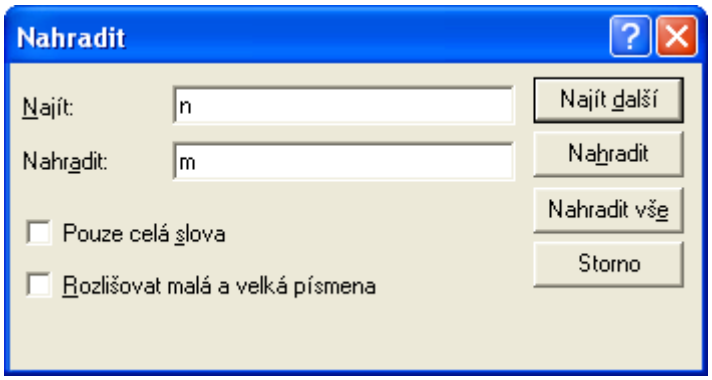
Obr. 3.10 - Vyhledávání

Zde se zadá hledaný text a parametry vyhledávání a kliknutím na tlačítko Najít další se vyhledá požadovaný text, který je nejbližší následující (či předchozí, záleží na volbě směru vyhledávání) od aktuální pozice kurzoru a ten se označí. Dalším stisknutím tohoto tlačítka se najde další atd. až dokud nebude nalezen poslední takový text v algoritmu, pak se již pozice kurzoru nemění.

Po uzavření vyhledávacího dialogu si však program hledaný text pamatuje i na dále a je možné jeho opakované vyhledávání oběma směry bez nutnosti dialog znovu otevírat. Lze tak učinit přes menu Úpravy příkazy Najít další či Najít předchozí nebo pomocí klávesových zkratek F3 (hledání dopředu) a Shift+F3 (hledání dozadu).

Nahrazování

Je-li možné určitý text v editoru najít, pak je také samozřejmě možné jej nechat automaticky nahradit za jiný. K tomu slouží funkce Nahradit (Ctrl+R). Po jejím zvolení se opět otevře dialogové okno, které má však navíc parametr Nahradit (viz Obr. 3.11).



Obr. 3.11 - Nahrazování

Zde ze zadá hledaný text, dále pak text, kterým má být tento nahrazen a parametry vyhledávání. Poté již stačí jen použít správné tlačítko. První z nich, Najít další, pouze nalezne nejbližší následující hledaný text od aktuální pozice kurzoru (shodné s funkcí vyhledávání). Tlačítko Nahradit vykoná to samé, avšak nalezený

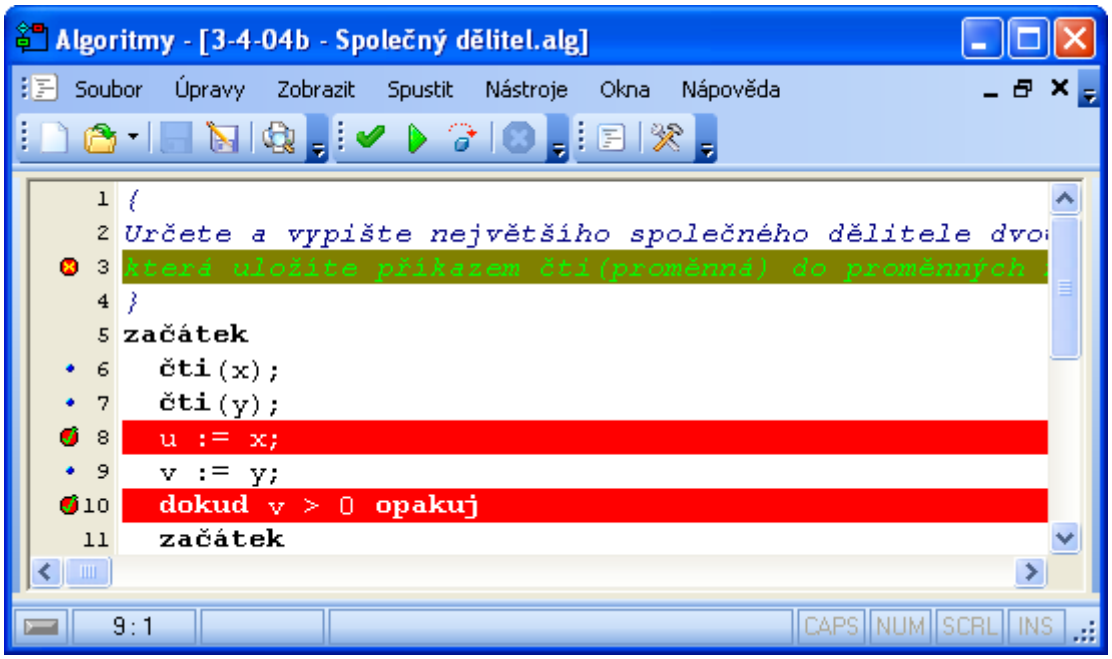
text rovnou nahradí novým. a konečně tlačítko Nahradit vše vyhledá a nahradí všechny hledané texty požadovanými, nacházející se za aktuální pozicí kurzoru. Chce-li uživatel tedy nahradit absolutně vše, musí před použitím této funkce přesunout kurzor na první znak prvního řádku v editoru.

3.8.3. Stop-značky

V editoru algoritmů je také možné definovat takzvané stop-značky. Jde v podstatě o značení řádků, na kterých uživatel chce, aby se běžící algoritmus pozastavil.

Tyto stop-značky se přidávají příkazem Přidat / odebrat stop-značku v menu Spustit, či klávesovou zkratkou F5. Stop-značka je tak přidána vždy na aktuální řádek. Pokud na něm již je, je naopak odstraněna. Další možností přidávání a rušení stop-značek je najet kurzorem myši nad šedivý prostor vlevo vedle řádku (na číslo onoho řádku) jemuž chceme změnit statut stop-značky a kliknout.

Stop-značku lze sice přiřadit kterémukoli řádku, avšak pouze u těch, na nichž se nachází funkční příkaz mají smysl. Řádky, na nichž funkční příkaz není, nebudou při běhu algoritmu dále řešeny a tudíž se na nich ani běh nemůže zastavit.



Obr. 3.12 - Stop-značky

3.9. Spouštění algoritmu

Kromě editoru se spoustou funkcí pro podporu psaní algoritmů program Algoritmy také umožňuje u napsaných algoritmů kontrolovat správnost jejich napsání (syntaxi), spouštět je či krokovat po jednotlivých příkazech, zadávat jim vstupy, sledovat jejich výstupy i hodnoty proměnných v kterékoli části běžícího algoritmu. Právě o těchto funkcích pojednává tato kapitola.

3.9.1. Kontrola syntaxe

Kontrola syntaxe se spouští příkazem Zkontrolovat v menu Spustit, či klávesovou zkratkou Ctrl+F9. Při jejím průběhu se indikátor ve stavovém řádku zbarví purpurově.

Tato kontrola se pokouší v kódu algoritmu najít základní chyby. Jde především o chybějící středníky, neukončené bloky, neznámé výrazy a podobně. V případě nalezení nějaké takovéto chyby v algoritmu je tato skutečnost vypsána v okně s výstupy Chyby (viz kapitola 3.10.2 Chyby, str. 20) a kontrola je ukončena jako neúspěšná.

Pokud nejsou žádné syntaktické chyby v algoritmu nalezeny a kontrola jej celý úspěšně projde, identifikuje veškeré řádky kódu, na kterých se nachází nějaký příkaz. Tyto řádky jsou v editoru označeny ikonou malého puntíku po levé straně (před číslem řádku). Takto je algoritmus připraven ke spuštění, až do doby, kdy v něm nastane jakákoli změna (stačí kamkoli přidat či umazat třeba mezeru). Pak tyto ikonky zmizí až do proběhnutí další úspěšné kontroly.

3.9.2. Spuštění algoritmu

Spuštění algoritmu se provádí příkazem Spustit v menu Spustit, či klávesovou zkratkou F9. Při běhu algoritmu se indikátor ve stavovém řádku zbarví zeleně.

Funkce spustit ještě před samotným spuštěním provede kontrolu syntaxe (nebyla-li již provedena) a pokračuje pouze je-li tato úspěšná.

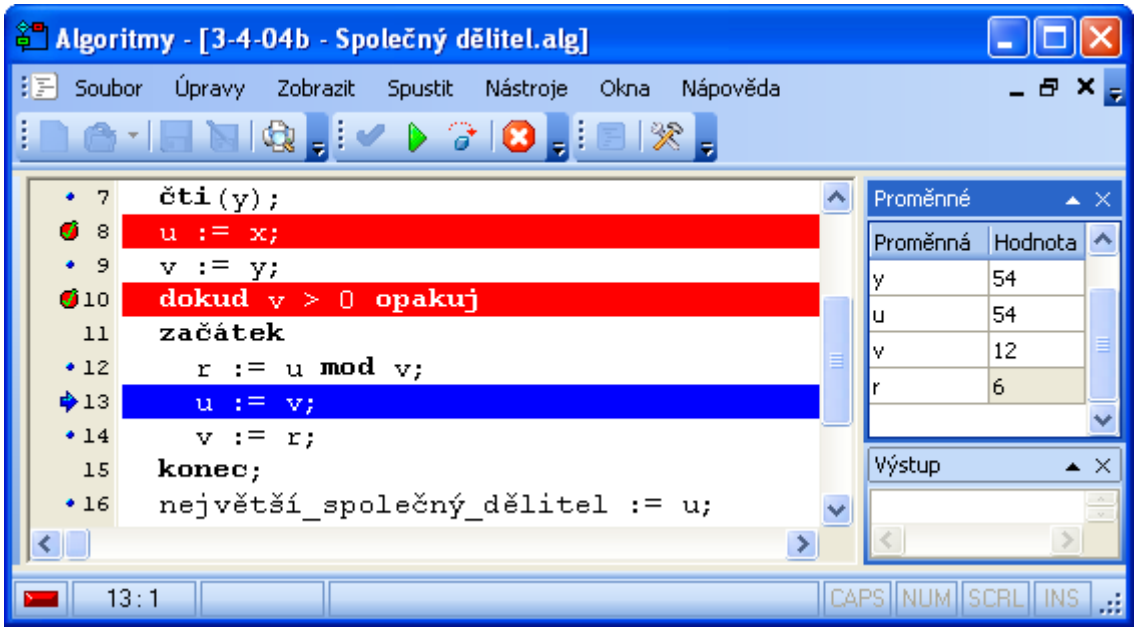
Po spuštění je rychle proveden kód algoritmu, přičemž jsou veškeré změny v hodnotách proměnných, polí či výstupy okamžitě zohledňovány v oknech s výstupy z programu (viz kapitola 3.10 Okna s výstupy, str. 20) po celý průběh algoritmu. V případě, že při vykonávání algoritmu dojde k nějaké chybě, je běh algoritmu zastaven a chyba vypsána v okně s výstupy Chyby (viz kapitola 3.10.2 Chyby, str. 20).

Narazí-li algoritmus při svém běhu na nějaký řádek se stop-značkou, běh algoritmu je před vykonáním příkazu na tomto řádku pozastaven, nikoli však ukon-

čen. V tom případě je možné pokračovat opětovným spuštěním algoritmu (běh pokračuje tam, kde byl pozastaven), posunutím pouze o jeden příkaz (řádek) či běh algoritmu zcela ukončit.

3.9.3. Krokování algoritmu

Krokování algoritmu je funkce, která vždy vykoná pouze jeden následující příkaz (řádek) algoritmu a jeho běh opět zastaví před příkazem následujícím. Není-li algoritmus doposud spuštěn, pak na něm nejprve provede kontrolu syntaxe (není-li již provedena) a až v případě jejího úspěchu spustí program a ihned jej pozastaví před prvním funkčním příkazem. Je-li běh algoritmu pozastaven, indikátor ve stavovém řádku má červenou barvu (viz Obr. 3.13).



Obr. 3.13 - Krokování algoritmu

Opětovným vyvoláním funkce Další krok algoritmu (F8), se tedy vykoná následující příkaz a běh je opět zastaven před příkazem po tomto následujícím, nebyl-li tento příkazem posledním. V tom případě by se běh algoritmu ukončil.

Pokračovat dále lze také spuštěním algoritmu, kdy běh pokračuje od místa, kde byl pozastaven, bez další zastávky. Je také možné běh programu ukončit zcela.

Řádek, na němž je příkaz, který má být právě vykonán je viditelně vyznačen, je na něj vždy automaticky přesunut kurzor a před jeho číslem je ikona s modrou šipkou doprava. Na jednom řádku může být i více funkčních příkazů, ale nedoporučuje se to.

Při pozastaveném běhu programu není možné kód algoritmu jakkoli měnit, či přepínat se mezi okny ostatních algoritmů.

3.9.4. Přerušení běhu programu

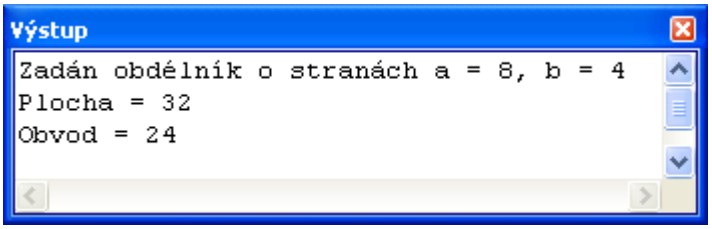
Běh spuštěného algoritmu lze kdykoli ukončit příkazem Přerušit v menu Spustit, či klávesovou zkratkou Ctrl+F2. To lze provést jak u algoritmu přímo běžícího (například v nějakém dlouhém cyklu), tak i u algoritmu pozastaveného. Je-li algoritmus vypnut, indikátor ve stavovém řádku má šedou barvu .

3.10. Okna s výstupy

Obecné ovládání plovoucích oken s výstupy bylo již probráno v kapitole 3.5.4 Ovládání oken s výstupy (str. 10). Nyní se zaměříme na význam a funkci každého z těchto oken.

3.10.1. Výstupy

Toto okno je určeno pro zachycení všech textových informací, které má algoritmus vypsat příkazem `napiš('Text')`. Veškeré takto vypsané texty se tedy v okamžiku zavolání příkazu **napiš** vypíší do tohoto okna (viz Obr. 3.14).



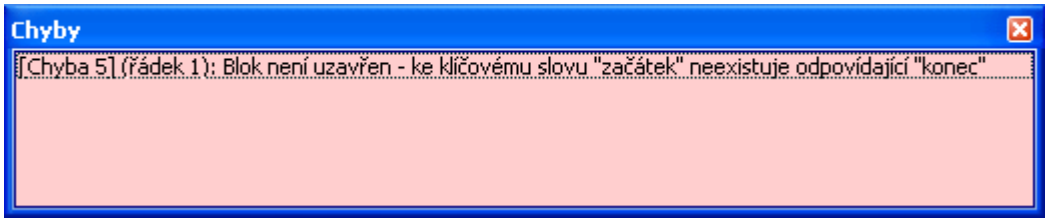
Obr. 3.14 - Okno s výstupy – Výstupy

Za každým samostatným příkazem **napiš** se automaticky zalomí řádek a další příkaz **napiš** tak již vypisuje text na řádek další.

Tento textový výstup je možné označit (celý či jen jeho část), stiskem kombinace kláves Ctrl+C si jej zkopírovat do schránky a poté jej klávesami Ctrl+V vložit do libovolného textového editoru.

3.10.2. Chyby

Jestliže při kontrole syntaxe či běhu algoritmu dojde k nějaké chybě v algoritmu, pak je tato chyba vypsána právě sem (viz Obr. 3.15).



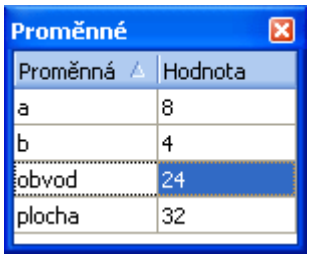
Obr. 3.15 - Okno s výstupy – Chyby

V okamžiku, kdy v tomto okně vypsána nějaká chyba, barva pozadí okna se změní z bílé na světle červenou a okno se automaticky zviditelní (bylo-li předtím skryto).

Seznam všech chyb, které program Algoritmy dokáže rozpoznat, se nachází v kapitole 3.14 Seznam možných chybových hlášení (str. 30).

3.10.3. Proměnné

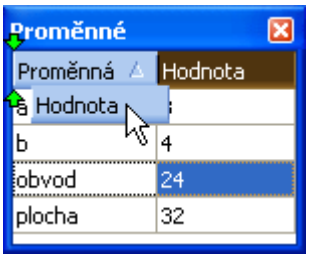
Zde se nachází kompletní seznam proměnných, které již v běžícím (či již proběhlém) algoritmu byly definovány. Tento seznam je rozdělen do dvou sloupců, přičemž v prvním z nich je název proměnné a v druhém její současná hodnota, neboť ta se v různých krocích algoritmu může měnit (viz Obr. 3.16).



Obr. 3.16 - Okno s výstupy – Proměnné

Pořadí proměnných je ve výchozím stavu shodné s pořadím, v jakém byly proměnné v algoritmu definovány. Je však možné tento seznam seřadit i podle hodnot v obou sloupcích a to kliknutím na záhlaví některého z nich. Šipečka, která se za názvem sloupce objeví, naznačuje směr řazení. Při opětovném klinutí na záhlaví již řazeného sloupce se totiž směr řazení obrátí (přepíná se mezi vzestupným a sestupným řazením). Hodnoty v obou sloupcích jsou však brány jako textové a podle toho i řazeny, takže pokud se ve sloupci Hodnota nacházejí pouze číselné hodnoty, i tak jsou řazeny pouze jako text po jednotlivých znacích. Řazení lze vypnout, klikne-li se myší na záhlaví sloupce, podle jehož hodnot se právě řadí, se současným stiskem klávesy Ctrl.

Pořadí sloupců je také libovolně měnitelné. Stačí myší jeden ze sloupců uchopit, přetáhnout jej na požadovanou pozici (viz Obr. 3.17) a pustit.



Obr. 3.17 - Změna pořadí sloupců

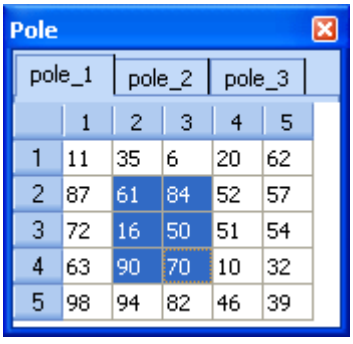
Šířku sloupců lze také libovolně měnit. Stačí myší najet na pravý okraj záhlaví příslušného sloupce, uchopit jej a tažením roztáhnout sloupec do požadované velikosti. Při dvojím kliknutí myší na pravý okraj záhlaví sloupce tento přizpůsobí automaticky svou šířku nejdelšímu textu ve všech svých řádcích.

V obou sloupcích lze též velmi snadno vyhledávat, což se může při větším počtu proměnných docela hodit. Stačí se přepnout do tohoto okna, zvolit sloupec, v němž má být vyhledáváno (je to ten u něhož má ukazatel aktuálního řádku bílé pozadí s pouze tečkovaným orámováním) a začít psát. Ukazatel aktuálního řádku je okamžitě přesunut na řádek, jehož obsah v patřičném sloupci začíná psaným textem.

Libovolnou hodnotu z tohoto seznamu lze též zkopírovat do schránky, a sice označením příslušného řádku a stiskem kláves Ctrl+C a poté ji vložit do kteréhokoli textového či tabulkového editoru klávesami Ctrl+V.

3.10.4. Pole

V tomto okně jsou uvedeny veškeré hodnoty všech polí již definované v běžícím či proběhlém algoritmu. Z pochopitelných důvod zde však jsou zobrazeny pouze jednorozměrná a dvourozměrná pole, a hodnoty polí vícerozměrných se zde, ani nikde jinde, nezobrazují. Hodnoty jednotlivých buněk jsou zde zobrazovány v textovém formátu (viz Obr. 3.18).



Obr. 3.18 - Okno s výstupy – Pole

Používá-li algoritmus více polí, jsou zde tato uvedena jako jednotlivé záložky nahoře, nad samotným přehledem hodnot zvoleného pole. Text v záložkách odpoví-

dá názvu pole v algoritmu. Kliknutím myši na příslušnou záložku jsou tak vypsány hodnoty právě tohoto pole.

Indexy (souřadnice) jednotlivých hodnot jsou uvedeny v prvním modrém sloupci a řádku tabulky. První index hodnoty pole definovaný v algoritmu je zde tedy „číslem" řádku a druhý pak „číslem" sloupce. Hodnoty těchto indexů mohou samozřejmě být i na přeskáčku. U jednorozměrných polí (posloupností) má pak tato tabulka vždy pouze jeden sloupec.

Lze zde označovat celé bloky buněk (Shift + šipky či myš), tyto výběry pak klávesovou zkratkou Ctrl+C kopírovat do schránky a poté ji vložit do kteréhokoli textového či tabulkového editoru klávesami Ctrl+V. Označit celý řádek naráz lze kliknutím na příslušnou buňku jeho prvního modrého sloupce a označení celého sloupce probíhá obdobně, kliknutím na příslušnou buňku jeho prvního řádku. Sloupce lze navíc označovat hromadně s přidržením klávesy Shift či na přeskáčku s klávesou Ctrl. Označení všech hodnot naráz proběhne po stisku klávesové zkratky Ctrl+A.

3.11. Nástroje

Program Algoritmy také disponuje několika nástroji, které poskytují další funkce při práci s algoritmy. O nich pojednává tato kapitola.

3.11.1. Automatické formátování kódu

Tato funkce je dostupná pod příkazem Naformátovat kód  v menu Nástroje. Po jejím spuštění je uživatel nejprve dotázán, přeje-li si skutečně tuto funkci na aktuální algoritmus aplikovat a varování, že pouze validní kód bude naformátován správně. Z tohoto důvodu je také dobré nejprve provést kontrolu syntaxe kódu (viz kapitola 3.9.1 Kontrola syntaxe, str. 18), která v tomto případě neprobíhá automaticky.

Pokud uživatel odsouhlasí použití funkce automatického formátování kódu, je aktuální algoritmus naformátován dle pravidel popsaných v kapitole 5.2 Správné formátování kódu (str. 56).

V některých případech však tento striktní formát nemusí být tím nejvhodnějším a záleží spíše na vkusu a zvyku uživatele, jaký formát kódu algoritmu je pro něj nejpřehlednější. Nicméně této funkce lze velmi dobře využít u nijak či skutečně nepřehledně naformátovaných algoritmů.

3.11.2. Módy programu

Program Algoritmy podporuje dva módy pro jejich spouštění. Tyto módy se týkají pouze jazyka, v němž jsou zpracovávána klíčová slova. Jedná se o tyto módy:

- Volitelný pseudokód
- Originální kód

Originální kód přejímá názvy klíčových slov z jazyka Pascal, kdežto u volitelného pseudokódu lze tato klíčová slova libovolně definovat (viz kapitola 3.12.3 Nastavení klíčových slov, str. 27). Pro každou jazykovou verzi programu Algoritmy jsou definovány výchozí názvy klíčových slov pro volitelný pseudokód. Výchozí mód programu je volitelný pseudokód.

Mezi oběma módy se dá libovolně přepínat v menu Nástroje – Mód, kliknutím (a zaškrtnutím) požadovaného módu. Touto změnou se v algoritmu nic nezmění, pouze budou za klíčová slova považována jiná slova než doposud.

V témže menu se též nachází funkce Převést pseudokód na kód  či Převést kód na pseudokód, záleží na tom, který mód je právě používán (vždy se převádí z aktuálního do toho druhého). Tento příkaz bez dalšího upozornění převede veškerá klíčová slova aktuálního módu v algoritmu, která nalezne, na klíčová slova módu druhého, do něhož se posléze přepne.

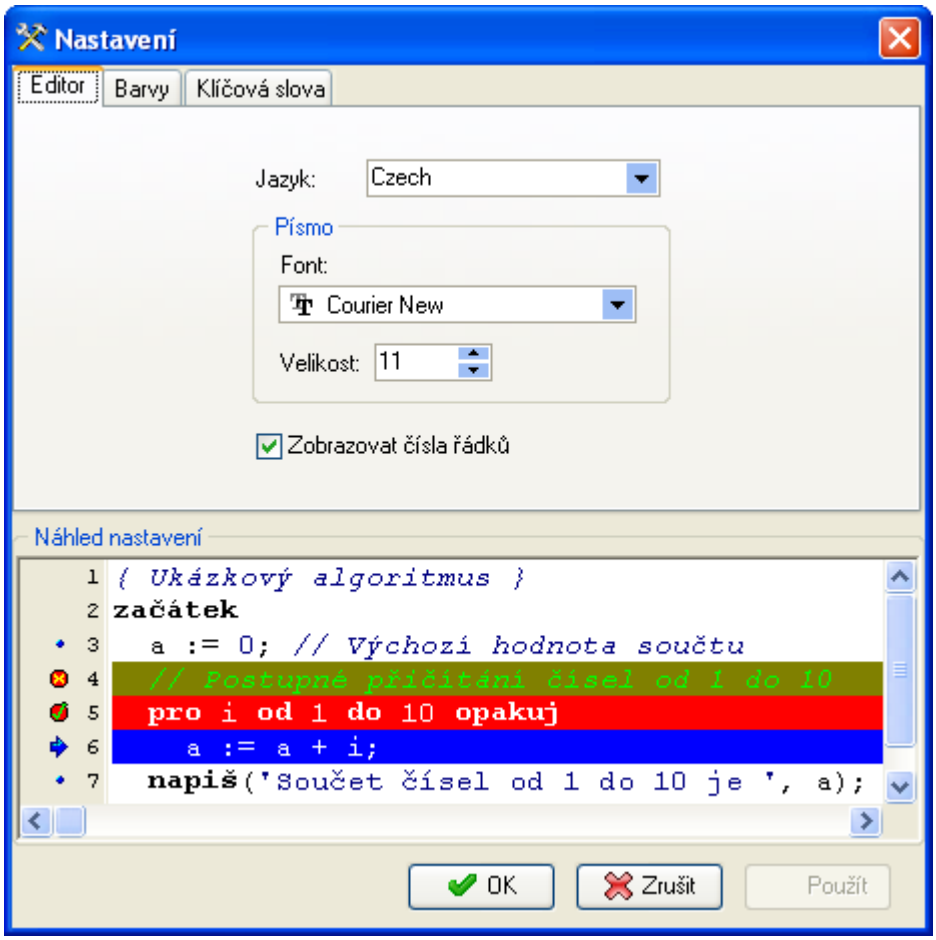
Lze tak algoritmus napsaný v libovolném jazyce kdykoli automaticky převést do originálního a všude univerzálního kódu a z něho pak třeba do kteréhokoli jiného jazyka. Každý uživatel si díky tomu může psát algoritmy v jemu srozumitelné formě a poté je bez problému převést do formy srozumitelné jinému uživateli, ve které mu je může předat, ať již oba užívají jakýkoli jazyk. Tento fakt vychází z předpokladu, že samostatná struktura algoritmů, nehledě na jazyk klíčových slov, je mezinárodní a všude stejná.

3.12. Nastavení programu

Prostředí programu Algoritmy si může každý uživatel libovolně nastavit dle svých představ. Nastavit lze barvy, písmo, jazyk, klíčová slova apod. Většina všech těchto nastavení probíhá v modálním okně, které je dostupné v menu Nástroje jako Nastavení.

3.12.1. Nastavení editoru

První záložka v okně nastavení se týká jazyka programu a písma (fontu) editoru (viz Obr. 3.19).



Obr. 3.19 - Nastavení editoru

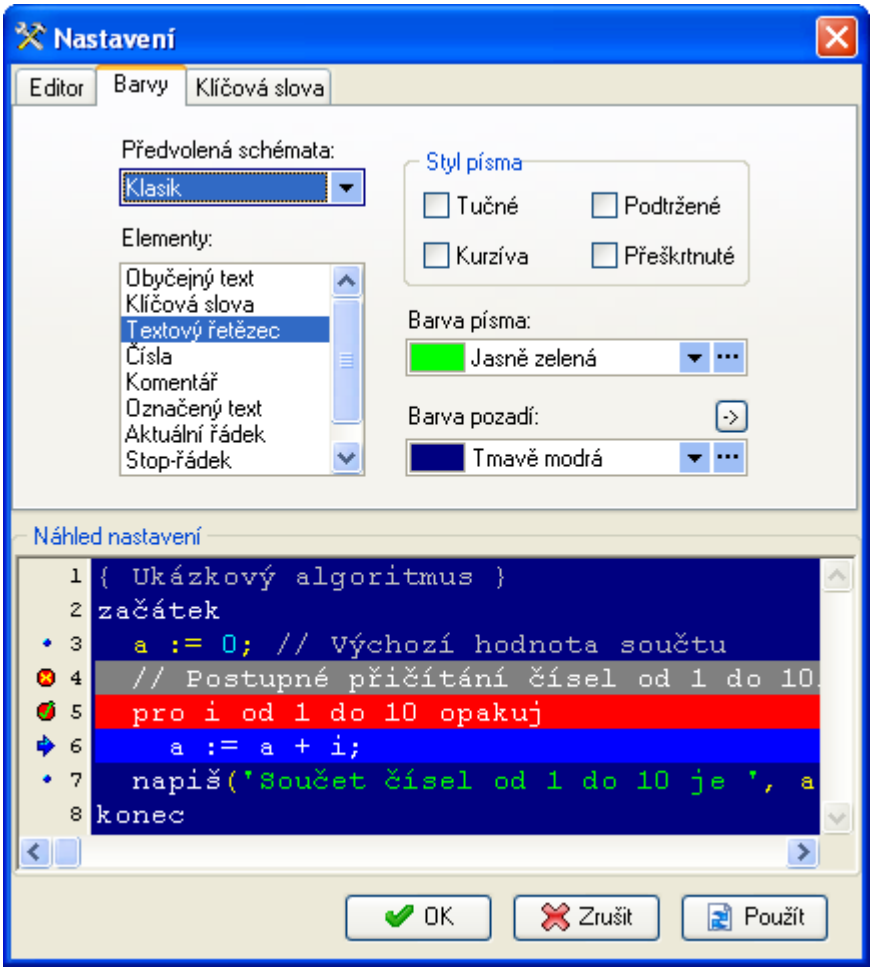
Na výběr zde jsou jazykové verze, jejichž soubory s koncovkou *.lng* se nacházejí v podadresáři *Languages* umístěném se stejném adresáři jako program Algoritmy. Změna jazyka programu se projeví až po kliknutí na tlačítko Použít nebo OK. Rozdíl mezi těmito dvěma tlačítky je, že Použít aplikuje zde provedené změny na celý program, ale okno s nastavením zůstane otevřené, kdežto po stisku OK se po aplikaci změn rovnou zavře. Tlačítko Zrušit neaplikuje žádnou ze zde provedených změn, které jsou díky tomu zrušeny, a pouze zavře okno s nastavením.

U všech ostatních nastavení (kromě jazyka) na všech záložkách, se každá změna projeví okamžitě v náhledu nastavení, které se nachází ve spodní části okna. Tak si je možné lépe a rychleji nastavit co nejvíce vyhovující prostředí v podstatě na první pokus.

Dále se zde dá tedy nastavit druh písma editoru a jeho velikost. Zaškrtávací políčko Zobrazovat čísla řádků určuje, zda uživatel chce či nechce zobrazovat po levé straně editoru čísla jednotlivých řádků.

3.12.2. Nastavení barev

Druhá záložka nastavení se týká výhradně barev a stylu písma jednotlivých elementů v editoru (viz Obr. 3.20).



Obr. 3.20 - Nastavení barev

Jsou zde k dispozici předvolená schémata, která umožní nastavit celou škálu barev všem elementům tak, jak jim bylo autorem programu určeno. Je nutné si na změnu v tomto poli dávat pozor, protože jakmile k ní dojde, jsou veškeré dříve provedené nepoužité změny na všech elementech zrušeny na výchozí hodnotu daného schématu. V případě schématu vlastního, což je hodnota, která se automaticky nastaví při jakékoli změně barvy či stylu písma libovolného elementu, si uživatel může nadefinovat vzhled editoru jaký mu nejlépe vyhovuje.

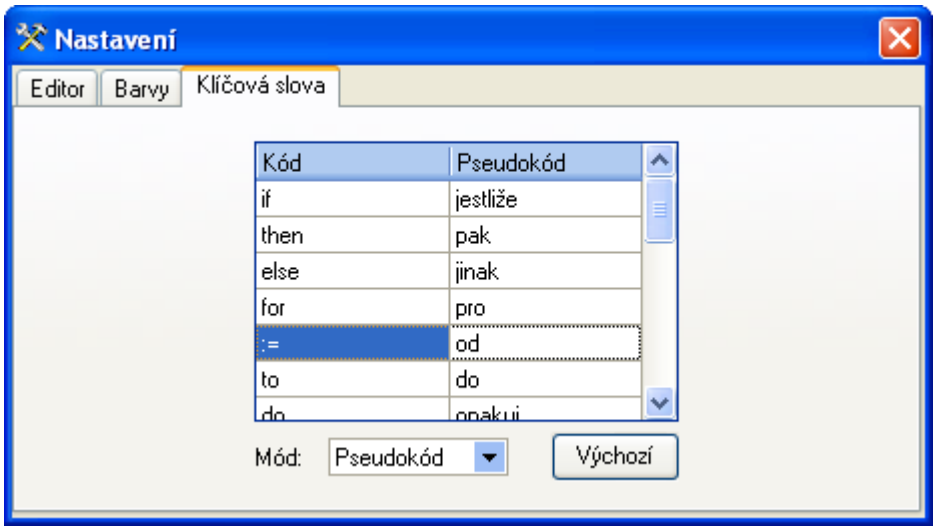
Jednotlivé elementy zde odpovídají všem rozlišovaným slovům či blokům a řádkům v kódu algoritmu. Každému elementu je pak možné nezávisle změnit styl písma (tučné, kurzíva, podtržené či přeškrtnuté), barvu písma i barvu pozadí. Barvy se dají vybírat ze seznamu základní palety barev nebo si namíchat libovolnou vlastní barvu z celé škály dle barevného rozlišení obrazovky.

V případě barvy pozadí se většinou předpokládá, že by měla být u všech elementů stejná a že tyto budou rozlišeny jen svým stylem a barvou písma. Aby uživatel pak nemusel pracně měnit barvu pozadí vždy každému elementu zvlášť na tutéž, je zde tlačítko  (vpravo nad políčkem pro výběr barvy pozadí), které po kliknutí na něj rovnou nastaví právě zvolenou barvu pozadí všem elementům najednou.

Veškeré změny jsou opět okamžitě promítnuty v náhledu nastavení a do celého programu se promítnou teprve až po stisku tlačítka OK nebo Použít.

3.12.3. Nastavení klíčových slov

Na poslední záložce v okně nastavení je možné změnit si klíčová slova pseudokódu (viz Obr. 3.21) a přepnout mód, v němž program vyhodnocuje algoritmy (viz také kapitola 3.11.2 Módy programu, str. 24).



Obr. 3.21 - Nastavení klíčových slov

Seznam klíčových slov je zde uveden v tabulce o dvou sloupcích. V prvním z nich jsou uvedena ve znění originálního kódu převzatého z jazyka Pascal a ta jsou neměnná. V druhém sloupci jsou pak uvedena ve znění volitelného pseudokódu a dají se zde libovolně upravovat. Ve výchozím stavu jsou zde nastavena tak, jak jsou definována v jazykové verzi zvolené při prvním spuštění programu (viz kapitola 3.3 První spuštění programu, str. 5). I když byl později jazyk změněn, klíčových slov pseudokódu se to nijak nedotkne. Je zde však tlačítko Výchozí, na které stačí kliknout a klíčová slova pseudokódu se okamžitě obnoví do výchozí podoby definované pro aktuálně zvolený jazyk (stačí je mít zvolený v okně s nastavením, na záložce Editor, jeho použití předem není nutné).

Jak již bylo řečeno, klíčová slova pseudokódu pouze definují, jaká slova v algoritmu budou v tomto módu považována za klíčová, a to včetně určení jejich významu.

Klíčová slova se mohou skládat pouze ze znaků písmen, číslic či podtržítka („_"), nesmí začínat číslicí a každé z nich musí být jedinečné. Tyto náležitosti jsou při měnění těchto slov programem kontrolovány a změny je možné použít pouze v případě jejich dodržení.

3.12.4. Ukládání nastavení

Veškeré nastavení programu provedené v okně nastavení, ale i to týkající se uživatelské úpravy a rozmístění menu, panelů nástrojů, oken s výstupy, volby módu atd. je programem Algoritmy automaticky ukládáno při jeho vypínání, a to do souboru *settings.dat*, který je umístěn do stejného adresáře jako samotný program. Při dalším spuštění programu jsou pak veškerá nastavení z tohoto souboru zase načtena a aplikována na prostředí, takže to vypadá přesně tak, jako když jej uživatel předtím opouštěl.

Tento soubor lze samozřejmě zálohovat nebo libovolně přenášet (kopírovat) s programem či zvlášť na různá místa jeho užití.

Pokud se z nějakého důvodu chce uživatel vrátit k původnímu nastavení programu, je zde funkce Obnovit výchozí nastavení nacházející se v menu Nástroje. Tato funkce v podstatě smaže soubor *settings.dat* a zajistí, aby program při svém následujícím vypnutí neukládal aktuální nastavení. Díky tomu bude program při jeho dalším spuštění vypadat tak, jako když byl spouštěn poprvé (viz 3.3 První spuštění programu, str. 5).

3.13. Seznam klávesových zkratek

Soubor

Založit nový soubor s algoritmem ..... Ctrl+N
Otevřít existující soubor s algoritmem ..... Ctrl+O
Uložit aktuální změněný algoritmus do stávajícího souboru ..... Ctrl+S
Náhled tisku aktuálního algoritmu ..... Ctrl+W
Tisk aktuálního algoritmu ..... Ctrl+P

Editor

Zrušit změnu v editoru ..... Ctrl+Z
Odvolat zrušení změny v editoru ..... Shift+Ctrl+Z
Kopírovat označený text do schránky ..... Ctrl+C
Vložit text ze schránky na aktuální pozici kurzoru v editoru ..... Ctrl+V
Vyjmout označený text z editoru a uložit jej do schránky ..... Ctrl+X
Odstranit aktuální řádek v editoru ..... Ctrl+Y
Vyhledání textu v editoru ..... Ctrl+F
Najít další hledaný text F3
Najít předchozí hledaný text ..... Shift+F3
Nahradit určitý text v editoru jiným ..... Ctrl+R

Spouštění algoritmu

Přidat / ubrat stop-značku na aktuálním řádku F5
Zkontrolovat syntaxi ..... Ctrl+F9
Spustit aktuální algoritmus F9
Vykonat další krok algoritmu F8
Přerušit běh programu ..... Ctrl+F2

Obecné

Nápověda F1
Ukončit program Algoritmy ..... Alt+F4

3.14. Seznam možných chybových hlášení

Číslo	Chybové hlášení
0	Není zadán žádný algoritmus
1	Algoritmus musí začínat klíčovým slovem "začátek" a končit "konec"
2	Chybí operátor nebo středník
3	Proměnná "X" není v tuto chvíli definována
4	";" nesmí předcházet klíčovému slovu "jinak"'
5	Blok není uzavřen - ke klíčovému slovu "začátek" neexistuje odpovídající "konec"
6	Bylo očekáváno "X" ale bylo nalezeno "Y"
7	Závorka není uzavřena - k "(" schází odpovídající ")"
8	Neznámý příkaz: "X"
9	Chyba při vyhodnocování výrazu: X
10	Prázdné závorky "()"
11	Byla očekávána celočíselná hodnota
12	Hodnota pole "X" se souřadnicemi [x, y] není v tuto chvíli definována
13	Došla paměť
14	Neznámá chyba
15	Chybí hodnota nebo výraz
16	Hledané slovo "X" nebylo nalezeno
17	Byl očekáván název proměnné, ale bylo nalezeno "X"
18	"X" je rezervované slovo

4. Programátorská dokumentace

V této kapitole se podíváme na program Algoritmy z programátorského hlediska. Zaměříme se především na komponentu *TAlgorithm*, jež zpracovává samotné algoritmy a také samozřejmě pak na samotný program, vytvářející k této komponentě grafické uživatelské rozhraní.

4.1. Použité nástroje

Program Algoritmy byl kompletně vytvořen v programovacím nástroji Borland Delphi 7. Samotná komponenta zpracovávající algoritmy (*TAlgorithm*) byla vytvořena pouze za pomocí standardních prostředků a funkcí, které tento vývojový nástroj poskytuje ve verzi Professional.

Program Algoritmy, který pak s touto komponentou pracuje a v podstatě k ní jenom vytváří grafické uživatelské rozhraní, dále kromě standardních prostředí Delphi vestavěných funkcí používá následující komponentové balíčky:

- SynEdit – editor kódu algoritmu, jejich tisk a náhled – viz webová adresa synedit.sourceforge.net
- SynUniHighlighter – univerzální „zvýrazňovač" klíčových slov v editoru – viz webová adresa www.delphist.com/UniHighlighter.html
- Developer Express – menu, plovoucí okna s výstupy, stavový řádek, záložky a editory v okně Nastavení – viz webová adresa www.devexpress.com

4.1.1. Jmenná konvence

Veškeré objekty a komponenty použité v programu Algoritmy podléhají jednotné jmenné konvenci, která standardizuje způsob jejich pojmenovávání. Názvy objektů jsou tedy tvořeny tříznakovým prefixem z malých písmen, který co nejvhodnějším způsobem charakterizuje třídu daného objektu a poté samotným anglickým názvem, vystihujícím funkci objektu v programu. Tento název může být i víceslovný, přičemž první písmeno každého slova je velké, ostatní malá. Například akce pro tisk algoritmu pak má název *actPrint*, akce pro uložení souboru pod jiným názvem *actSaveAs* atd. Prefix *act* mají tedy akce, *lbl* popisky (labels), *btn* tlačítka (buttons) atd.

Názvy souborů unit podléhají stejnému pravidlu, avšak za tříznakový prefix je přidáno ještě podtržítko. Toto opatření je kvůli tomu, že například v případě formulářů je nezbytné, aby se objekt formuláře (*TForm*) jmenoval jinak než soubor (unita) v němž je uložen. Přitom je samozřejmě žádoucí mít tyto názvy byly co

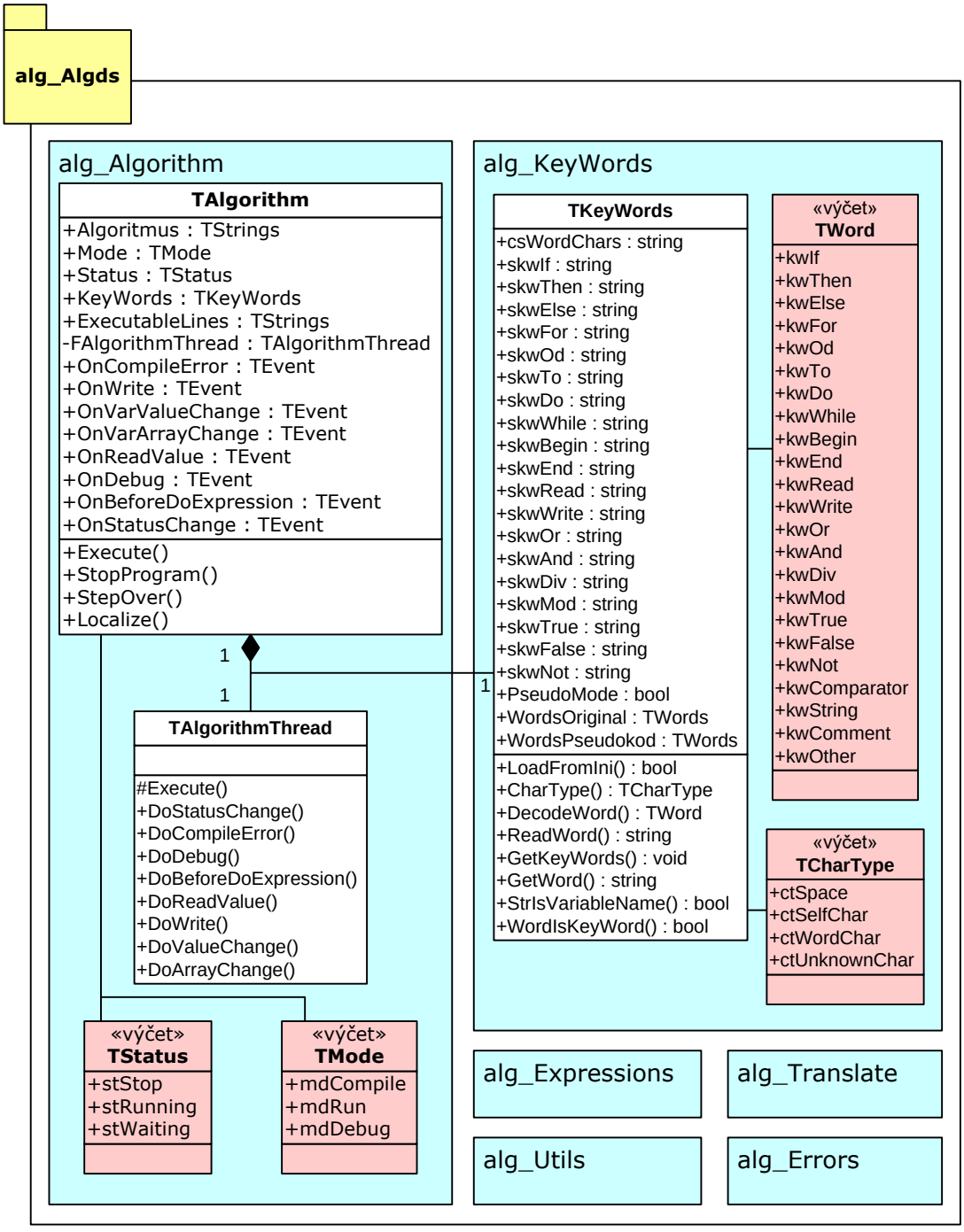
nejpodobnější, ale také je třeba aby z obou bylo okamžitě patrné, který je který. Takže například formulář pro nastavení se jmenuje *frmSetup*, je třídy *TfrmSetup* a uložen je v souboru (unitě) *frm_Setup*. Prefix *frm* je tedy užit u fomulářových unit, prefix *dtm* pak u datamodulů a *unt* u samostatných unit.

V případě unit komponentových je pak prefix jejich názvu volen tak, aby vyjadřoval, že jsou součástí oné komponenty. Čili všechny unity komponenty pro zpracování algoritmů *TAlgorithm* začínají názvem „*alg_*".

V případě konstant je používán pouze dvouznakový prefix, z nichž první znak říká jde-li skutečně o konstantu („*c*") a druhý označuje typ této konstanty (*s – string, i – integer, r – real, a – array...*). Poté následuje název konstanty, definující její význam (např. *csSettingsFileName*).

4.2. Komponentový balíček alg_Algds

Komponentový balíček *alg_Algds.dpk* zapouzdřuje komponentu *TAlgorithm*, která obstarává kompletní zpracování algoritmů, které jsou pro ni v podstatě vstupem. Výstupy pak jsou různé události vyvolávané jako reakce na určité příkazy při zpracování algoritmu.



Obr. 4.1 - Struktura balíčku alg_Algds zachycující základní vztahy jeho unit, tříd a typů

4.2.1. Unit Algorithm

Toto je hlavní unit celého balíčku. Je v něm totiž definována samotná komponenta *TAlgorithm* a veškeré její hlavní funkce. Dalším zde definovaným objektem je *TAlgorithmThread*, což je potomek třídy *TThread*, tedy vlákna. V něm je totiž spouštěn samotný běh algoritmu, díky čemuž další funkce programu zůstávají funkční po celou dobu, kdy algoritmus probíhá.

Komponenta *TAlgorithm* má několik hlavních atributů. Jedním z nich je *Algoritmus*, což je vlastnost typu *TStrings* (seznam textových řetězců – řádků algoritmu). Do ní je třeba zadat vstupní algoritmus. Dále je tu vlastnost *Mode*, což je mód, ve kterém chceme algoritmus spustit. Tyto módy jsou na výběr tři: *mdControl* (na algoritmu má být provedena pouze kontrola syntaxe, viz kapitola 3.9.1 Kontrola syntaxe, str. 18), *mdRun* (algoritmus má být prostě spuštěn, viz kapitola 3.9.2 Spuštění algoritmu, str. 18) a *mdDebug* (algoritmus je spuštěn, avšak před vykonáním každého jeho příkazu je pozastaven a je vyvolána událost *OnDebug*, 3.9.3 Krokování algoritmu, str. 19).

Mezi další vlastnosti komponenty patří události (events), které jí je možné přiřadit. *TAlgorithm* má k dispozici tyto události:

- *OnCompileError* – je vyvolána, dojde-li při zpracování algoritmu k nějaké chybě. Vrací zprávu s popisem chyby, identifikační číslo této chyby (viz kapitola 3.14 Seznam možných chybových hlášení, str. 30), číslo řádku a sloupec, definující místo v algoritmu, kde k chybě došlo.
- *OnWrite* – je vyvolána v okamžiku, kdy je v algoritmu zpracován příkaz `napiš` (`write`) a vrací text, který má být tímto příkazem vypsán.
- *OnVarValueChange* – je vyvolána ve chvíli, kdy byla změněna hodnota nějaké proměnné v algoritmu. Vrací název této proměnné a její novou hodnotu.
- *OnVarArrayChange* – je vyvolána ve chvíli, kdy byla změněna hodnota v nějakém poli v algoritmu. Vrací název tohoto pole, souřadnice změněné hodnoty (jako pole čísel) a její novou hodnotu.
- *OnReadValue* – je vyvolána v okamžiku, kdy algoritmus zpracovává příkaz `čti` (`read`). Vrací název proměnné, jejíž hodnota je načítána a starou hodnotu této proměnné. Je očekáváno, že tato vstupně – výstupní hodnota (*var*) bude během obsluhy této události změněna na novou – tedy takovou, jakou zadá uživatel.
- *OnDebug* – tato událost je vyvolána pouze v případě, že je algoritmus spuštěn v módu *mdDebug* a to ve chvíli, kdy je zpracován libovolný příkaz algoritmu, tedy přesněji před tím, než je proveden. Vrací číslo řádku a sloupce, kde začíná právě zpracovávaný příkaz. Ihned po vyvolání této události je běh algoritmu pozastaven (*suspend*) až do okamžiku kdy je opět spuštěn metodou *StepOver* nebo *Execute* (se změněným módem na *mdRun*).

- *OnBeforeDoCommand* – je vyvolána před vykonáním každého příkazu (nehledě na mód), ještě před *OnDebug* a vrací číslo řádku a sloupce, kde začíná právě zpracovávaný příkaz. Po provedení této události se znovu testuje mód a není-li požadováno ukončení běhu algoritmu, což dává v podstatě možnost algoritmus ukončit, či jej změnou módu pozastavit před vykonáním daného příkazu. Toho je využíváno například narazí-li algoritmus na uživatelem definovanou stop-značku (breakpoint, viz kapitola 3.8.3 Stop-značky, str. 17).
- *OnStatusChange* – je vyvolána ve chvíli, kdy je změněn status běhu algoritmu. Status je atributem *TAlgorithm*, který je pouze pro čtení a může nabývat tří hodnot: *stStop* (běh algoritmu je zcela vypnut), *stRunning* (algoritmus byl spuštěn a běží), *stWaiting* (algoritmus byl spuštěn, ale je právě pozastaven a čeká na své pokračování či ukončení). Změny statusu lze docílit pouze provedením příslušné metody komponenty.

Dalším atributem komponenty *TAlgorithm* je *ExecutableLines*. Toto je opět vlastnost typu *TStrings*, je určena pouze pro čtení a po úspěšném proběhnutí kontroly syntaxe obsahuje seznam čísel řádků algoritmu, které obsahují nějaký funkční příkaz. Tohoto seznamu se pak využívá při identifikaci a vyznačení těchto řádků v editoru a určování platnosti stop-značek.

Komponenta *TAlgorithm* kromě atributů disponuje i několika public metodami. Hlavní z nich je metoda *Execute*, která spouští běh algoritmu ve zvoleném módu ať již od začátku, či od aktuální pozice, na které byl pozastaven (viz kapitola 3.9 Spouštění algoritmu, str. 18). Dále je tu metoda *StepOver*, která algoritmus, v případě, že je pozastaven, opět spustí s tím, že nachází-li se stále v módu *mdDebug*, bude u dalšího příkazu opět pozastaven (viz kapitola 3.9.3 Krokování algoritmu, str. 19). Metoda *StopProgram*, pak okamžitě zcela ukončí běh algoritmu (viz kapitola 3.9.4 Přerušení běhu programu, str. 20). Další metoda *Localize*, která požaduje na vstupu název ini souboru s jazykovou verzí programu, pak na základě tohoto souboru načte veškeré texty, které používá, a to v jazyce definovaném v tomto souboru.

Postup zpracování algoritmu

Nyní si rozebereme, jakým způsobem je algoritmus zpracováván při svém běhu. Jak již bylo řečeno, algoritmus je zadán přes atribut *Algoritmus*, vlastností *Mode* je nastaven mód, ve kterém má běžet a metodou *Execute* je spuštěn.

Metoda *Execute* nejprve provede kontrolu, je-li algoritmus zadán a poté buď obnoví činnost vlákna *TAlgorithmThread* (*resume*), která je vždy po skončení běhu

algoritmu zastavena (*suspend*). V případě módu *mdControl* (pouze pro kontrolu syntaxe) je funkce jinak volaná vláknem spuštěna přímo, neboť tato kontrola má velmi rychlý průběh a není ani žádoucí během ní nechávat aplikaci nezávisle na průběhu kontroly uživateli dostupnou. Takto je tedy buď přes vlákno či rovnou spuštěna funkce *ExecuteCode*.

Tato funkce nejprve nastaví status na *stRunning*. V dalším kroku zkontroluje že prvním slovem algoritmu (komentář se nepočítá) je slovo **`začátek`**. Poté procedurou *ReadBlock* zjistí pozici znaku (v řetězci algoritmu) kde začíná tento první začátek a k němu odpovídající **`konec`** (čímž zároveň ověří souhlasí-li počty všech začátků a konců v celém algoritmu). S těmito zjištěnými souřadnicemi je pak zavolána procedura *RunCode* (v případě kontroly *ControlCode*), která má tedy na vstupu dvě čísla – pozici od a do které má zpracovat algoritmus.

Procedura *RunCode* v cyklu postupně zpracovává všechny příkazy, které ve vymezeném prostoru nalezne. Nejprve je funkcí *ReadCommand* přečten další příkaz od aktuální pozice. Tato funkce nejdřív dle klíčového slova rozpozná druh příkazu a podle něho načte i jeho parametry, které vrací ve zvláštní proměnné typu *TCommand*. Není-li první předčtené slovo žádným z klíčových, je toto považováno za název proměnné či pole, jemuž má být přiřazena nějaká hodnota. Ta je následně přečtena, většinou ve formě výrazu, který je zakončen buď středníkem či klíčovým slovem **`jinak`**. Pokud toto zakončení před dalším klíčovým slovem chybí, je to považováno za chybu, která je okamžitě vyvolána a běh algoritmu ukončen. K tomuto samozřejmě může dojít i při nesprávném pořadí parametrů jakéhokoli jiného příkazu.

Hodnota vrácená funkcí *ReadCommand* obsahuje veškeré informace potřebné k vykonání kteréhokoli příkazu. To již probíhá zpět v proceduře *RunCode*. Zde je nejprve vyvolána událost *OnBeforeDoCommand* a *OnDebug*, přičemž po průběhu každé z nich je testováno, není-li tu požadavek běh algoritmu ukončit.

Poté je podle druhu načteného příkazu provedena jeho obsluha. U příkazů blokových, které pod sebou obsahují další části kódu vykonávaných určitým způsobem (u větvení jen pokud je splněna podmínka, u cyklů je volání celého bloku opakované) se využívá rekurze a celý blok je volán opět funkcí *RunCode*, jejíž působnost je nyní vymezena pouze na daný blok. U příkazů přímých (**`čti`** a **`napiš`**) je pak pouze vyvolána událost (*OnReadValue*, *OnWrite*), která je obslouží zvenčí.

V případě že má dojít k nastavení hodnoty nějaké proměnné či pole, je nejprve vyhodnocen výraz který má určovat tuto nově přiřazovanou hodnotu funkcí *ValueOfExpression* a jí vrácená hodnota je pak procedurou *SetVariable* přiřazena proměnné. Proměnné jsou uchovávány v lineárním spojovém seznamu definovaným typem *TVariable*. Ten požaduje pouze název proměnné, její hodnotu

a ukazatel na další prvek seznamu. Procedura *SetVariable* začne u první hodnoty, na níž má komponenta uchovaný ukazatel a postupně projde celý tento seznam, přičemž kontroluje, není-li v něm již proměnná s tímto názvem. Pokud ano, změní pouze její hodnotu, pokud ne, je tato k seznamu přidána na jeho konec.

V případě hodnot polí je princip podobný, pouze procedura *SetArray* navíc pracuje se souřadnicemi pole, kteréžto jsou po svém vyhodnocení jakoby druhým jménem proměnné.

Za zmínku v tomto procesu stojí ještě zmíněná funkce *ValueOfExpression*, která postupně projde každé slovo výrazu a v případě že se jedná o název proměnné, je za něho do řetězce doplněna jeho hodnota v textovém tvaru. Také víceznakové operátory nahradí jednoznakovými, aby se s nimi dle lépe pracovalo. Po tomto doplnění hodnot je zavolána funkce *EvaluateExpression* z unity *alg_Expressions*, která již libovolný korektní výraz z konkrétními hodnotami dokáže vyhodnotit (viz kapitola 4.2.3 Unit Expressions, str. 39).

Takto procedura *RunCode* zpracovává příkaz po příkazu až do konce, který má vymezen na vstupu. V případě kontroly je místo ní volána obdobná procedura *ControlCode*, která má však několik podstatných rozdílů. Předně nejsou vyvolávány žádné události, mimo *OnCompileError*, která nastane pouze při nalezení chyby. Dále jsou veškeré příkazy pouze čteny nikoli prováděny, takže například u cyklů je jejich blok vždy pouze jedenkrát přečten, čímž je zkontrolována jeho správnost. U větvení (příkazu **`jestliže`**) je jeho blok přečten vždy, včetně bloku alternativního (při použití **`jinak`**). Hodnoty proměnných a polí nejsou ukládány, je pouze kontrolována korektnost výrazů, které je definují.

Po ukončení běhu procedury *RunCode* a všech jejích rekurzí funkce *ExecuteCode* vrací *true*. *False* by vrátila pouze v případě, že byl běh *RunCode* přerušen výjimkou, která by zde byla zachycena, neboť se přepokládá, že příslušná zpráva o této chybě již byla podána (*OnCompileError*). Poté je vyprázdněn seznam všech proměnných a hodnot polí a status běhu algoritmu je nastaven na *stStop*.

Celý tento proces samozřejmě dbá na veškerá pravidla při používání vláken a všechny externí projevy jsou synchronizovány (*synchronize*).

4.2.2. Unit KeyWords

V této unitě je vymezena třída *TKeyWords*, která se stará o definici, nastavení a základní funkce pro práci s klíčovými slovy. Komponenta *TAlgorithm* používá instanci této třídy.

Atributy této třídy jsou především jednotlivá klíčová slova, operátory a konstanty, jenž jsou následující:

Klíčová slova

- if jestliže
- then pak
- else jinak
- for pro
- := od
- to do
- do opakuj
- while dokud
- begin začátek
- end konec
- read čti
- write napiš

Operátory

- or nebo
- and a
- div celočíselné dělení
- mod zbytek po dělení
- not negace

Konstanty

- true pravda
- else nepravda

Veškeré tyto výrazy jsou nastavovány přes metodu *Set*, která vždy zajistí, aby byly uloženy pouze ve formátu malých písmen.

Dalšími atributy je pak seznam znaků, z nichž mohou být tvořena slova a nastavení módu (3.11.2 Módy programu, str. 24).

Kromě těchto atributů třída *TKeyWords* ještě obsahuje několik funkcí pro práci se „slovy" v algoritmu. Zde jsou ty nejdůležitější z nich:

- *LoadFromIni* – Načte klíčová slova pseudokódu ze zadaného vstupního ini souboru.
- *CharType* – Zjistí typ vstupního znaku.
- *DecodeWord* – Ze vstupního řetězce zjistí o jaký typ slova se jedná.
- *ReadWord* – Přečte ze zadaného řetězce další slovo od zadané pozice, kterou zvýší.

- *GetKeyWords* – Vrátí seznam všech klíčových slov aktuálního módu jako *TStrings*.
- *GetWord* – Ze zadaného typu slova vrátí toto slovo jako textový řetězec v aktuálním módu.
- *StrIsVariableName* – Zjistí, je-li zadaný řetězec názvem proměnné.
- *WordIsKeyWord* – Zjistí, je-li vstupní slovo klíčovým, či jiným typem slova.

4.2.3. Unit Expressions

Jedinou funkcí této unity je vyhodnocování výrazů. K tomu slouží jediná statická public funkce *EvaluateExpression*. Tato funkce má na vstupu textový řetězec obsahující výraz, který má být vyhodnocen. Výstupem je pak výsledek tohoto výrazu, což je jediná hodnota typu variant.

Vstupní výraz musí být platným výrazem v infixové formě obsahující již pouze konkrétní hodnoty. Ty musí být tedy dosazeny za případné proměnné vyskytující se v tomto výrazu ještě před zavoláním této funkce.

Funkce dokáže kromě číselných hodnot pracovat také s hodnotami textovými a logickými. U hodnot logických jsou tyto vyznačeny jedním speciálním znakem, kde „F" je logická nepravda (false) a „T" logická pravda (true). V případě hodnot textových musí být řetězec z obou stran vymezen apostrofy či uvozovkami. Všechny tyto úpravy vstupního výrazu musí být provedeny před zavoláním funkce *EvaluateExpression*, což tedy zajišťuje komponenta *TAlgorithm*.

Princip vyhodnocování výrazů je postaven na převodu infixové formy do postfixové, jejíž vyhodnocení je již celkem bezproblémové. Přitom jsou samozřejmě zohledňovány priority jednotlivých operátorů a závorky.

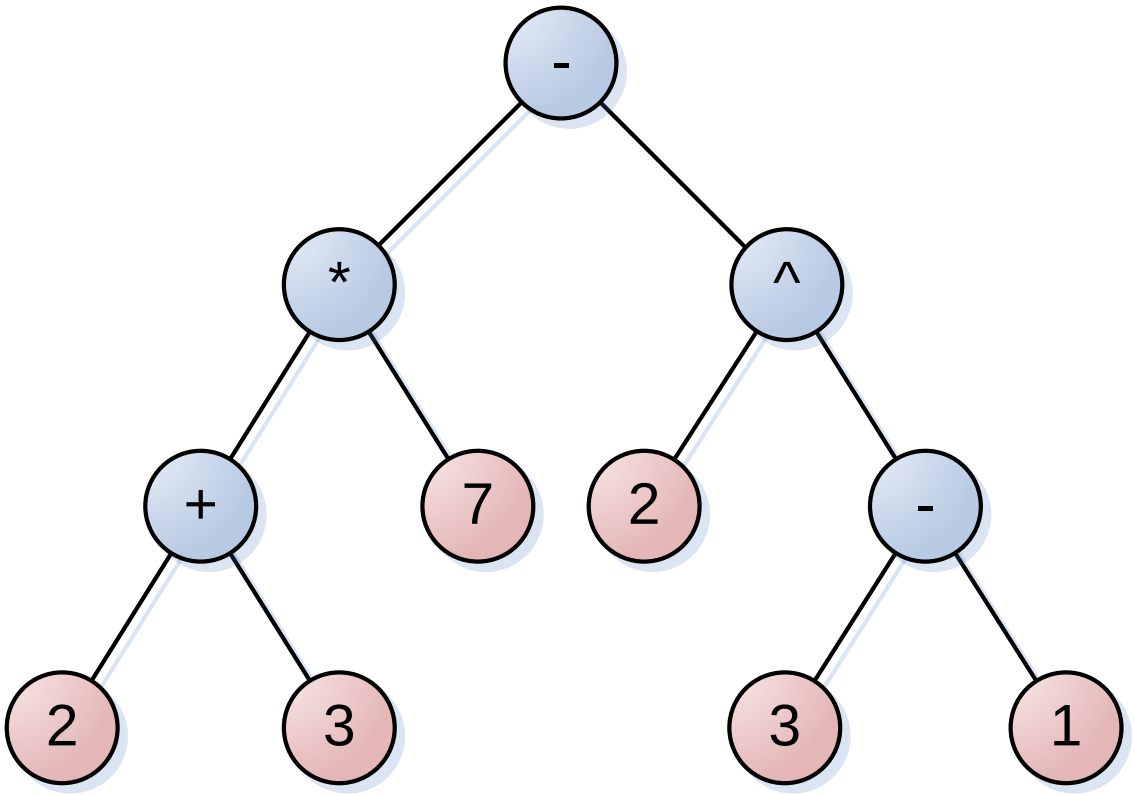
Priorita operátorů

1. not
2. ^
3. *, /, div, mod
4. +, -
5. <, >, <=, >=, =, <>
6. and
7. or

Příklad

Převod vstupního infixového výrazu do postfixového a jeho vyhodnocení si předvedeme na jednoduchém příkladu.

Vstupní infixový výraz: **(2 + 3) * 7 - 2 ^ (3 - 1)**, tedy $(2+3)\cdot 7-2^{3-1}$, čehož výsledek je **31**. Tento výraz je v přirozené podobě, jak jej všichni znají, zadávají a jsou schopni vypočítat. Po rozkreslení tohoto výrazu do binárního stromu dle priorit, dostaneme Obr. 4.2.



Obr. 4.2 - Výraz rozkreslený do binárního stromu

Postfixovou formu získáme při procházení tohoto binárního stromu metodou postorder, kdy procházíme od vrchního bodu postupně všechny uzly. Nejprve vstupujeme vždy do levého a jeho hodnotu přidáme do zásobníku až v okamžiku, kdy daným uzlem procházíme naposled (opouštíme jej).

Tento převod v programu Algoritmy řeší funkce *InfixToPostfix*, jejímž vstupem je infixový výraz a výstupem výraz postfixový, oba ve formě textového řetězce. Tato funkce převádí infix na postfix postupným přepisem hodnot do výsledného tvaru. Pro operátory je použit pomocný zásobník, kam jsou odkládány v pořadí v jakém byly přečteny ze vstupního řetězce s ohledem na jejich prioritu. Je-li tedy ze vstupu přečten operátor, jsou z pomocného zásobníku nejprve přečteny (zprava) všechny operátory s vyšší nebo stejnou prioritou a přidány do výsledného řetězce. Poté je tento operátor přidán do pomocného zásobníku.

Do tohoto pomocného zásobníku jsou odkládány i otevírací závorky. Je-li pak ze vstupního výrazu přečtena závorka uzavírací, jsou ze zásobníku postupně přečteny všechny operátory (zprava) až po první otevírací závorku a tyto jsou přidány do výsledného výstupního výrazu. Takto se pokračuje až dokud není vstupní infixový řetězec přečten celý. Nakonec jsou do výsledného výrazu ještě přidány veškeré operátory, které zůstaly v pomocném zásobníku (v opačném pořadí, neboť jsou čteny od zadu).

Názorný postup tohoto převodu je naznačen zde:

Infixový řetězec	**Pomocný zásobník**	**Postfixový řetězec**
`(2 + 3) * 7 - 2 ^ (3 - 1)`		
`2 + 3) * 7 - 2 ^ (3 - 1)`	`(`	
`+ 3) * 7 - 2 ^ (3 - 1)`	`(`	`2`
`3) * 7 - 2 ^ (3 - 1)`	`(+`	`2`
`) * 7 - 2 ^ (3 - 1)`	`(+`	`2 3`
`* 7 - 2 ^ (3 - 1)`		`2 3 +`
`7 - 2 ^ (3 - 1)`	`*`	`2 3 +`
`- 2 ^ (3 - 1)`	`*`	`2 3 + 7`
`2 ^ (3 - 1)`	`-`	`2 3 + 7 *`
`^ (3 - 1)`	`-`	`2 3 + 7 * 2`
`(3 - 1)`	`-^`	`2 3 + 7 * 2`
`3 - 1)`	`-^(`	`2 3 + 7 * 2`
`- 1)`	`-^(`	`2 3 + 7 * 2 3`
`1)`	`-^(-`	`2 3 + 7 * 2 3`
`)`	`-^(-`	`2 3 + 7 * 2 3 1`
	`-^`	`2 3 + 7 * 2 3 1 -`
		`2 3 + 7 * 2 3 1 - ^ -`

Výsledkem je tedy postfixový výraz: **2 3 + 7 * 2 3 1 - ^ -**.

Dalším krokem je vyhodnocení tohoto postfixového výrazu. O to se stará funkce *EvaluatePostfix*. Vyhodnocení je zde provedeno tak, že se začnou číst hodnoty po jedné (zleva) a přepisovat do pomocného zásobníku. V momentě, kdy je místo hodnoty přečten operátor, jsou z pomocného zásobníku vyzvednuty dvě poslední hodnoty (zprava), je mezi nimi provedena příslušná operace (dle operátoru) a výsledná hodnota je opět přidána na konec pomocného zásobníku. Takto se pokračuje až k poslednímu znaku řetězce postfixového výrazu. Výsledek je pak hodnotou (jedinou), která zbude v pomocném zásobníku. Tento postup je naznačen zde:

Postfixový řetězec	Pomocný zásobník
2 3 + 7 * 2 3 1 - ^ -	
3 + 7 * 2 3 1 - ^ -	2
+ 7 * 2 3 1 - ^ -	2 3
7 * 2 3 1 - ^ -	5
* 2 3 1 - ^ -	5 7
2 3 1 - ^ -	35
3 1 - ^ -	35 2
1 - ^ -	35 2 3 1
- ^ -	35 2 3 1
^ -	35 2 2
-	35 4
	31

Výsledkem výrazu je tedy číselná hodnota 31.

V případě hodnot textových jsou možné pouze operace sčítání (+) a ostré porovnávání (=, ≠). Je-li některá tato operace vykonávána mezi číselnou hodnotou a textovou, je ta číselná nejprve převedena na textovou. Pokud dojde k pokusu o vykonání jiné operace s textovými hodnotami, funkce vyvolá výjimku se zprávou vysvětlující její původ. O tu už se však stará komponenta *TAlgorithm*, která ji převede na chybové hlášení, které zobrazí uživateli.

4.2.4. Unit Translate

Tato unita obsahuje pouze dvě statické procedury: *Translate* a *FormatCode*.

Translate

Účelem této procedury je přeložit algoritmus z jednoho módu do druhého. Zajišťuje tedy funkci překládání módů z uživatelského hlediska popsanou v kapitole 3.11.2 Módy programu (str. 24).

Na vstupu tedy požaduje ukazatel na algoritmus (*TStrings*) a na existující a nadefinovaný objekt *TKeyWords* (viz kapitola 4.2.2 Unit KeyWords, str. 37). Z objektu klíčových slov si zjistí, který mód je právě aktuální, poté projde celý algoritmus a každé klíčové slovo v aktuálním módu nahradí klíčovým slovem módu druhého. Ostatní slova nechá ve jejich původním znění. Nakonec již pouze přepne mód na ten druhý, neboť nyní je algoritmus v jeho znění.

FormatCode

Procedura *FormatCode* naformátuje algoritmus podle pravidel definovaných v kapitole 5.2 Správné formátování kódu (str. 56). Zajišťuje tedy funkci automatického formátování kódu algoritmu z uživatelského hlediska popsanou v kapitole 3.11.1 Automatické formátování kódu (str. 23).

4.2.5. Unit Utils

Obsahuje řadu pomocných statických funkcí, převážně pro práci s textovými řetězci, které komponenta využívá. Mohou je také využívat i další programy, a to i v případě že přímo nevyužívají komponentu *TAlgorithm*, stačí když si ji přidají mezi používané unity (*uses*).

4.2.6. Unit Errors

Tato unita definuje a zpřístupňuje veškerá chybová hlášení, které je komponenta schopna rozpoznat a reagovat na ně. Za tímto účelem obsahuje dvě statické procedury: *DefineCsErrors* a *LocalizeErrors*.

DefineCaErrors

Výchozí znění jednotlivých chyb definuje procedura *DefineCaErrors*, která v podstatě naplní pole *caErrors*, z něhož jsou již pak přímo chybová hlášení načítána. Definuje je tedy pozice (index) v tomto jednorozměrném poli. Jejich kompletní seznam je uveden v kapitole 3.14 Seznam možných chybových hlášení (str. 30). Také je zde definováno pole *caExpressionErrors*, což jsou chybová hlášení objevující se v případě selhání určité matematické operace při vyhodnocování výrazů (viz kapitola 4.2.3 Unit Expressions, str. 39).

LocalizeErrors

Kromě definice výchozího znění těchto chybových hlášení, kterým je čeština, je zde i funkce *LocalizeErrors*, která dokáže ze zadaného názvu ini souboru a názvu jeho sekce načíst nová znění všech chybových hlášení. Při lokalizaci pak stačí jen zavolat tuto funkci a veškerá chybová hlášení jsou tak automaticky „přeložena" do jazyka definovaného ve vstupním ini souboru. Ten musí samozřejmě obsahovat příslušné hodnoty v předem definovaném formátu (viz kapitola 4.3.5 Lokalizace, str. 45).

4.2.7. Unit Register

Tato unita pouze zajišťuje registraci a přidání komponenty *TAlgorithm* do palety komponent Delphi při své kompilaci. Komponenta je tedy přidána pod záložkou *Algorithms*.

4.3. Program Algoritmy

Program Algoritmy tvoří v podstatě pouze grafické uživatelské rozhraní při práci s komponentou *TAlgorithm*. Nicméně i tak musí zajišťovat řadu nezbytných funkcí, které budou naznačeny v této kapitole.

4.3.1. Hlavní okno

Hlavní okno programu Algoritmy zajišťuje drtivou většinu všech hlavních funkcí celé aplikace. Především tedy obsluhuje komponentu *TAlgorithm*, nastavuje její vlastnosti a obsluhuje její události. Dále se stará o MDI okna s editory, plovoucí okna s výstupy, menu, ukládání a načítání nastavení do souboru *settings.dat*, aplikace změn nastavení za běhu, evidence naposled otevřených souborů, zobrazování změny statusu ve stavovém řádku, vyhledávání a nahrazování výrazů v editorech atd.

4.3.2. Editor kódu

Jako editor kódu algoritmů je tedy použita komponenta *TSynEdit*. Ta je přímo určena pro tento účel a poskytuje také nadstavbu pro podporu „označování" řádků, vhodnou pro grafické znázornění krokování algoritmu a umísťování stop-značek (*TDebugSupportPlugin*). Také umožňuje automatické kompletování slov díky komponentě *TSynCompletionProposal*, jež potřebuje znát pouze jejich seznam.

Dále toto okno zajišťuje ukládání a načítání souborů s algoritmy a hlídá, došlo-li k jeho změně. S hlavním oknem pak komunikuje pomocí událostí, které vyvolává, přičemž hlavní okno může reagovat na změny přímo, neboť má do všech svých podoken umožněn přístup, opačně však nikoli.

4.3.3. Nastavení

V okně s nastavením prostředí programu je především obsluha vizuálních komponent umožňujících toto nastavení uživateli měnit a promítání těchto změn do editoru s náhledem nastavení ve spodní části okna. Také jsou zde definovány přednastavené barevné styly celého programu, které se přednastaví při jejich výběru na záložce barvy.

Na záložce Klíčová slova je pak třeba ohlídat pravidla jejich definice, tedy jedinečnost a korektnost názvu. Korektnost je kontrolována vždy po jejich napsání, před uložením každého z nich, jedinečnost pak před použitím změn v nastavení, funkcí *CheckUniquesKeyWords*.

Změny nastavení jsou v programu projeveny až po stisku tlačítka OK nebo Použít. Tuto aplikaci nastavení však má na starosti hlavní okno programu, neboť jedině to má přístup samo k sobě a ke všem svým podoknům, kde je třeba tyto změny promítnout. Tato funkce hlavního okna je vlastně reakcí na událost okna s nastavením. Ta je definována v proměnné *OnApplyChanges* a je vyvolána vždy, když si uživatel přeje použít zde provedené změny do celého programu.

4.3.4. Náhled tisku

Okno s náhledem tisku využívá komponenty *TSynEditPrintPreview*, která je součástí stejného komponentového balíčku jako editor kódu *TSynEdit*. Při ovládání zobrazení tohoto náhledu jsou pak již pouze využity vlastnosti a metody této komponenty.

Nastavení tiskárny při tisku z tohoto okna je realizováno standardní komponentou Deplhi *TPrintDialog*, která je však umístěna na hlavním formuláři a oknu s náhledem je předáván pouze odkaz na ni. Toto opatření je učiněno z důvodu, aby při opakovaném tisku různými způsoby (bez náhledu nebo s náhledem) stačilo tiskárnu nastavit pouze jednou (poprvé) a poté již bylo toto nastavení vždy předvoleno.

4.3.5. Lokalizace

Jelikož program Algoritmy umožňuje změnu nastavení své jazykové verze přímo za běhu, bylo nutné zde použít jiný mechanismus pro lokalizaci než-li užití resourcestringů. Jako zdroj lokalizací byla zvolena forma ini souborů (ač se změněnou koncovkou na *.lng*), neboť jejich jednoduchá struktura umožňuje zkušenějším uživatelům jejich libovolnou modifikaci a poskytuje tak prostor pro vznik dalších jazykových mutací programu a jejich volnou šiřitelnost.

Samotný průběh změny jazyka programu dle obsahu těchto souborů je pak rozdělen do několika částí. Co se týče komponenty *TAlgorithm*, tak ta dokáže lokalizovat sama sebe zavoláním její public metody *Localize*, jejímž jediným vstupním parametrem je název ini souboru s lokalizací.

Dále je třeba přeložit veškeré popisky a další textové atributy všech komponent na všech formulářích. K tomu slouží procedura *LocalizeForm* v unitu *unt_Localization*, které stačí na vstupu předat ukazatel na objekt formuláře, jehož komponenty mají být lokalizovány. Tato procedura poté postupně projde veškeré

komponenty umístěné na tomto formuláři a pokud některá z nich má textovou vlastnost z definovaného seznamu (*Caption*, *Hint*, *Title*...), a existuje-li k ní v souboru s lokalizací příslušná hodnota, pak je tato hodnota oné vlastnosti nastavena.

Tato procedura je při změně jazyka postupně provedena na všechny aktuálně existující formuláře v aplikaci a stejně tak i při vytváření jakéhokoli okna nového.

Další částí lokalizace jsou pak hlášení různých dialogových oken, která program za určitých situací zobrazuje. Tyto zprávy jsou uchovány v globální proměnné *InternalMessages* typu *TStrings*, která je společná a dostupná celé aplikaci. Formát uchování těchto zpráv byl zvolen stejný jako je tomu v ini souborech, tedy „klíč=hodnota", kde klíčem je interní neměnný název jednoznačně identifikující každou ze zpráv. Libovolnou z nich je pak možné načíst pomocí funkce *GetMessage*, které je předán pouze onen interní název zprávy. Výchozí české znění těchto zpráv je nastaveno procedurou *SetDefaultMessages*, která je volána pouze po spuštění programu. Při změně jazyka je pak díky tomuto formátu načtena celá sekce těchto zpráv do *InternalMessages* naráz.

Poslední částí lokalizace je zde pak proedura *LocalizeMenu*, která přeloží resourcestringy komponent z balíčku DevExpress.

Seznam lokalizací programu, který je uživateli nabízen k výběru, sestavuje procedura *LoadLanguageList*, jíž je potřeba definovat pouze adresář, v němž jsou soubory s lokalizacemi umístěny. Tato procedura jej prohledá a vrátí seznam názvů souborů s koncovkou *.lng* v něm obsažených. Je tedy třeba aby název souboru odpovídal názvu jazykové verze, která je v něm uložena.

4.3.6. Kontrola aktualizace

Program Algoritmy při každém svém spuštění provádí kontrolu existence nové verze. Tato kontrola je realizována v unitu *dtm_Images*, který je součástí datamodulu obsahujícím jinak pouze seznamy ikon (*TImageList*) používaných v celém programu. Jelikož je potřeba mít tyto ikony k dispozici po celou existenci programu a dostupnou všem oknům, je tento datamodul vytvářen vždy při startu programu a uvolňován až při jeho vypnutí. Jako takový je ideální právě pro tento úkol.

Aby zjišťování aktualizace z internetu nezdržovalo uživatele při každém spouštění programu, probíhá na pozadí v samostatném vlákně *TUpdateThread*, paralelně s během programu. Zde je využito standardní komponenty v Delphi 7 *TIdHTTP*, která dokáže načíst obsah zadané webové adresy. Ta tedy načte speciální stránku z oficiálního webu k projektu, která vrací pouze číslo verze, která je právě umístěna na tomto webu. Poté je již jen porovnáno toto číslo (po složkách) s číslem

verze právě běžícího programu a je-li to z internetu vyšší, znamená to, že nová verze je k dispozici.

Na tuto skutečnost je poté uživatel upozorněn dialogem, který mu umožňuje přepnout se automaticky do internetového prohlížeče na stránky projektu, kde je ona nová verze k dispozici ke stažení.

V případě že během komunikace se serverem, na němž jsou stránky projektu umístěny, dojde k chybě (náhlé odpojení, příliš dlouhá odezva, absence spojení s internetem vůbec...), vzniklá výjimka je pohlcena blokem *try – except*, a uživatel není při práci s programem rušen vůbec.

5. Algoritmy

Program Algoritmy, jak již ostatně jeho název napovídá, je primárně určen především k pohodlnému psaní, kontrolování, spouštění a krokování algoritmů. Slouží tedy jako učební pomůcka k problematice algoritmů.

V této kapitole se zaměříme právě na ně. Probereme zde každý podporovaný příkaz, jaké musí mít náležitosti, jaká je jeho podstata a správný formální zápis. Ukázky algoritmů zde budou uvedeny ve znění výchozí verze českého pseudokódu.

5.1. Podporované příkazy

Na začátku je třeba upozornit, že program Algoritmy je *case insensitive*, tedy nerozlišuje malá a velká písmena, načež proměnná s názvem **proměnná** je považována za stejnou jako například **ProMěNNá**.

Nejprve si probereme jednotlivé příkazy podporované programem Algoritmy.

5.1.1. Uvození bloku příkazů

Blokem algoritmu se rozumí jeden či více libovolných příkazů jdoucích za sebou v daném pořadí, kteréžto jsou vymezeny klíčovými slovy **začátek** (před prvním příkazem bloku) a **konec** (po posledním příkazu bloku).

```
začátek
  příkaz;
  ...
  příkaz;
konec
```

Do bloku je vždy třeba uzavřít celý algoritmus. Jinými slovy prvním slovem algoritmu musí být **začátek** a posledním **konec**. Do bloku se také uzavírají skupiny příkazů u větvení a cyklů (zpravidla pouze v případě, že jich je více než jeden, ale není to podmínkou). Tedy následuje-li pod těmito typy příkazů slovo **začátek**, platí jejich funkce (podmínka větvení či multiplicita cyklů) na celý tento blok až po slovo **konec**.

Počet všech začátků v algoritmu musí odpovídat počtu všech konců!

Klíčová slova **začátek** a **konec** sama o sobě nejsou příkazy, jelikož nevykonávají žádnou činnost, mají pouze vymezovací funkci.

5.1.2. Příkazy větvení

Větvením se rozumí omezení vykonání určitého příkazu (či bloku příkazů) jistou podmínkou. V programu, v souladu s výukou algoritmizace na FIM UHK rozlišujeme dva typy příkazu větvení: úplné a neúplné. Příkazy podmínkou uvozené jsou tedy provedeny pouze v případě jejího splnění (platnosti výrazu s výslednou logickou hodnotou *true* - pravda).

Příkazem neúplného větvení je zde příkaz začínající klíčovým slovem **jestliže**. Za tímto klíčovým slovem musí následovat podmínka (např. **x > 3**), poté klíčové slovo **pak**, a poté již příkaz, který bude v případě splnění podmínky vykonán a v případě opačném přeskočen. Je-li těchto příkazů více, je třeba je uvést do bloku.

Příkaz větvení tedy může vypadat takto:

```
jestliže x > 3 pak
  příkaz;
```

Podmínka může být samozřejmě libovolně složitá, nezbytné však je, aby jejím výsledkem byla logická hodnota reprezentovaná buď *true* / *false* (získaná např. porovnáním dvou hodnot) nebo číselně (0 pak zastupuje *false* a vše ostatní je *true*).

Úplným větvením je možnost použití klíčového slova **jinak**. To může následovat pouze těsně za příkazem (či za slovem **konec** v případě bloku), jenž je omezen podmínkou větvení a za ním musí následovat příkaz další, který bude vykonán pouze v případě, že podmínka splněna nebude. Může to tedy vypadat například takto:

```
jestliže x > y pak
  vetší := x
jinak
  větší := y;
```

5.1.3. Příkazy cyklu

Rozlišujeme dva způsoby zápisu příkazu cyklu: příkazy začínající klíčovým slovem **dokud** a příkazy začínající klíčovým slovem **pro**, přičemž příkaz **pro** je speciálním případem příkazu **dokud**, ale v určitých případech je výhodnější a úspornější používat spíše příkaz **pro**.

Pro

Příkaz **pro** je určen pro případy, kdy chceme použít nějaké příkazy ve známém počtu opakování (např. 10x). Parametrem za klíčovým slovem **pro** je proměnná, v jejíž hodnotě se bude počítat kolikáté je to opakování, za klíčovým slovem **od** následuje počáteční hodnota této proměnné a za klíčovým slovem **do** hodnota, kterou když proměnná přesáhne, tak cyklus končí. Příkaz je pak uzavřen klíčovým slovem **opakuj**, za nímž již následuje rovnou příkaz (či blok příkazů), jež má být opakován. V každém zopakování je tedy k proměnné automaticky přičteno +1. Pro deset opakování by tedy tento příkaz mohl vypadat takto:

```
pro i od 1 do 10 opakuj
  příkaz;
```

Zde **i** je proměnná, z jejíž hodnoty se dá u příkazů v cyklu kdykoli zjistit, kolikáté je toto opakování. Obě hodnoty (od a do) mohou být samozřejmě definovány jinou proměnnou či výrazem, přičemž pokud je hodnota **od** větší než **do**, cyklus není vykonán ani jednou. Cyklu **pro** se využívá nejvíce při procházení hodnot polí.

Dokud

Příkaz **dokud** je obecnější než příkaz **pro**, a to především co do podmínky pro ukončení cyklu. Příkaz **dokud** totiž má za parametr logický výraz, který se před každým zopakováním takto uvozených příkazů znovu otestuje a pouze v případě, že jeho výsledkem je logická pravda (*true*) je cyklus znovu zopakován, v opačném případě je ukončen. Za touto podmínkou následuje klíčové slovo **opakuj** a za ní již příkaz (či blok příkazů), jež bude opakován. Cyklus by tedy mohl vypadat například takto:

```
dokud x > 0 opakuj
  x := x - 2;
```

Je zde třeba dbát na to, aby nedošlo k tomu, že podmínka tohoto cyklu bude splnitelná vždy, neboť v tom případě by cyklus nebyl nikdy ukončen a probíhal tak donekonečna (např. při 1 = 1).

5.1.4. Jednoduché příkazy

Jednoduché funkční příkazy jsou takové, které jsou vykonány pouze samy o sobě a neobsahují již žádné další podpříkazy, za kterézto se však nepovažuje vy-

hodnocování výrazů. Vezměme si každý z nich podporovaný programem Algoritmy jednotlivě.

Čti

Tento příkaz umožňuje uživateli definovat hodnoty některých proměnných v průběhu běžícího algoritmu. Tímto příkazem definované proměnné se nazývají vstupy algoritmu, jelikož za předpokladu, že by byl tento algoritmus samostatně spustitelným programem, by pak tento příkaz byl jedinou možností, jak získávat vstupy zvenčí.

Příkaz **čti** má v závorkách pouze jediný parametr a tím je proměnná, do níž bude přečtená hodnota zadána. Zápis příkazu vypadá takto:

```
čti(x);
```

Tento příkaz se tedy zeptá uživatele na hodnotu, kterou po jejím zadání přiřadí do proměnné **x**.

Napiš

Příkaz **napiš** je naopak jedinou možností, jak realizovat výstupy z programu (sledování aktuálních hodnot proměnných, které program Algoritmy umožňuje nelze považovat za plnohodnotný výstup). Díky němu lze vypsat libovolný text, hodnotu kterékoli proměnné nebo pole či přímo výrazu.

Příkaz **napiš** má tedy v závorkách jeden či více parametrů oddělených čárkami. Jeho zápis pak může vypadat například takto:

```
napiš('Násobkem hodnot', x, ' a ', y, ' je hodnota ', x * y, '.');
```

Tento příkaz tedy v případě že hodnota **x** je 3 a hodnota **y** je 4 vypíše následující text: **Násobkem hodnot 3 a 4 je 12.**

Přiřazení

Příkaz přiřazení není uvozen žádným klíčovým slovem ale obsahuje přiřazující dvojznak „:=" (dvojtečka a rovná se). Prvním slovem je však vždy název proměnné, do které bude hodnota za tímto dvojznakem přiřazena. Následuje := a název proměnné, pole, hodnota či výraz, jehož výsledek bude proměnné před dvojznakem přirazen.

Zápis přiřazení pak může vypadat například jako jeden z následujících příkladů:

```
x := 1;
y := x + 1;
z := 10 * (2 * x) ^ y - 74 * (x - y / 2);
```

Po provedení těchto tří příkazů tedy bude platit, že **x = 1**, **y = 2** a **z = 40**.

Při práci s kteroukoli proměnnou je třeba dbát na to, aby její hodnota byla v některém z předchozích příkazů definována (přiřazením nebo příkazem **čti**). V opačném případě totiž dojde k chybě a běh algoritmu je automatiky ukončen.

5.1.5. Konstanty, operátory, textové řetězce a komentáře

Zbývá ještě probrat slova, která nejsou přímo klíčovými slovy, ale pouze slovy rezervovanými. To znamená, že je, stejně jako klíčová, nelze použít pro názvy proměnných, neboť jejich využití je především při definici jiných proměnných a vyhodnocování výrazů.

Konstanty

Konstanty jsou zde rezervovaná slova, mající stejný význam jako hodnota, kterou zastupují. Program Algoritmy zná tato:

- **true** - Logická pravda (shodná například s výrazem **1 = 1**).
- **false** - Logická nepravda (shodná například s výrazem **1 = 2**).
- **pi** - Ludolfovo číslo (pí, 3.14159265358979).
- **exp** - Eulerovo číslo (e, 2.71828182845905).
- **random** - Není konstantou, ale pokaždé jiným náhodným reálným číslem mezi 0 a 1 z rovnoměrného rozdělení R(0,1).

Operátory

Dalším případem rezervovaných slov jsou operátory. Většina z nich však ani nejsou slova, ale znaky, jejichž význam je na první pohled zřejmý. Přesto si tu však každý z nich pro jistotu osvětlíme. Pořadí operátorů je řazeno dle jejich priority.

- **not** – Negace logického výrazu. Je-li hodnota následující za **not** logická pravda, bude výsledkem nepravda a naopak. Např. platí, že **(not true) = false**.
- **^** - Mocnina. Hodnota před tímto operátorem je umocněna mocnitelem, který následuje za tímto operátorem. Např. platí, že **2 ^ 3 = 8**.
- ***** - Násobení. Vynásobí hodnoty před a za tímto operátorem. Např. platí, že **2 * 3 = 6**.

- **/** - Dělení. Vydělí hodnotu před tímto operátorem hodnotou za ní. Např. platí, že **6 / 3 = 2**.
- **div** - Celočíselné dělení. Vrátí kolikrát se hodnota za tímto operátorem celá vejde do hodnoty před ním. Např. platí, že **7 div 3 = 2**.
- **mod** - Zbytek po dělení. Vrátí zbytek po dělení hodnoty před tímto operátorem hodnotou za ním. Např. platí, že **7 mod 3 = 1**.
- **-** - Odečítání. Odečte od hodnoty před tímto operátorem hodnotu za ním. Např. platí, že **3 – 2 = 1**.
- **+** - Sčítání. Vrátí součet hodnot před a za tímto operátorem. Např. platí, že **2 + 3 = 5**.
- **<=** - Menší nebo rovno. Vrátí logickou hodnotu `true` v případě, že hodnota před tímto operátorem je menší nebo rovna než ta za ním. V opačném případě je výsledkem `false`. Např. platí, že **(4 <= 3) = false**.
- **>=** - Větší nebo rovno. Vrátí logickou hodnotu `true` v případě, že hodnota před tímto operátorem je větší nebo rovna než ta za ním. V opačném případě je výsledkem `false`. Např. platí, že **(4 >= 3) = true**.
- **<** - Menší než. Vrátí logickou hodnotu `true` v případě, že hodnota před tímto operátorem je menší než ta za ním. V opačném případě je výsledkem `false`. Např. platí, že **(4 < 3) = false**.
- **>** - Větší než. Vrátí logickou hodnotu `true` v případě, že hodnota před tímto operátorem je větší než ta za ním. V opačném případě je výsledkem `false`. Např. platí, že **(4 > 3) = true**.
- **=** - Porovnání dvou hodnot. Vrátí logickou hodnotu `true` pouze v případě, že hodnoty před a za tímto operátorem jsou naprosto stejné, jinak je výsledkem `false`. Např. platí, že **4 = 4**.
- **<>** - Nerovná se. Vrátí logickou hodnotu `true` pouze v případě, že hodnoty před tímto operátorem není stejná jako hodnota za ním, jinak je výsledkem `false`. Např. platí, že **4 <> 3**.
- **and** - A zároveň. Vrátí logickou hodnotu `true` pouze v případě, že logické hodnoty před i za tímto operátorem jsou `true`. Pokud je jen jedna z nich či obě `false`, výsledkem je `false`. Např. platí že **(true and false) = false**.

- **or** - Nebo. Vrátí logickou hodnotu `true` pouze v případě, že alespoň jedna logická hodnota (případně obě) před či za tímto operátorem je `true`. Např. platí že **(true or false) = true**.

Ne všechny operátory však lze použít na všechny typy hodnot. Program Algoritmy rozlišuje totiž tři typy hodnot: logické, číselné a textové. Pro každý z nich lze použít pouze tyto operátory:

- logické hodnoty: not, =, <>, and, or
- číselné hodnoty: not, ^, *, /, div, mod, -, +, <=, >=, <, >, =, <>, and, or
- textové řetězce: +, =, <>

Operátory `not`, `and` a `or` lze u číselných hodnot použít díky tomu, že 0 je brána jako `false` a všechny ostatní číselné hodnoty jako `true`. Výsledkem však již bude logická hodnota `true` či `false` (např. **not 3 = false**).

Textové řetězce

Textový řetězec je typ hodnoty. Takovéto hodnoty mohou být přiřazovány proměnným a vykonávány s nimi i některé operace (sčítání a přímé porovnávání).

Veškeré textové řetězce přímo použité v algoritmu musí být umístěny mezi apostrofy či uvozovky (např. **'text'** a **"text2"** jsou textové řetězce). Vše co bude umístěno mezi jedním z těchto párů znaků je považováno za textový řetězec a jako s takovým je s ním i nakládáno. Dva druhy vymezení textových řetězců jsou použity mimo jiné i pro to, aby bylo možné do řetězců zahrnout i oba tyto znaky (např. "McDonald's" nebo 'Řekl: "Ahoj!"').

Jsou-li sčítány dvě hodnoty tohoto typu operátorem +, nejedná se o sčítání v pravém slova smyslu, nýbrž o sloučení těchto dvou textů do jednoho (např. **'Ahoj' + 'Karle' = 'AhojKarle'**). Sčítáme-li textový řetězec a číselnou hodnotu, bude tato hodnota převedena také na textový řetězec, což bude i typem výsledku takovéhoto součtu (např. **250 + ' Kč' = '250 Kč'**).

V případě porovnávání je srovnáván každý jednotlivý znak obou textových řetězců (při *case sensitive*, tedy rozlišování malých a velkých znaků jako různých) a výsledkem je logická hodnota určující, jsou či nejsou-li tyto dva řetězce stejné (např. výsledkem výrazu **'honza' = 'jan'** je **false**, stejně jako u výrazu **'honza' = 'Honza'**). Při srovnávání číselné a textové hodnoty je opět ta číselná nejprve převedena na textovou a až poté dojde k porovnání (tedy např. **platí** že **2 = '2'** ale **neplatí** že **2 = 'dvě'**). To samé, ale s opačnými výsledky platí i pro operátor **<>**.

Komentáře

Je velmi dobrým zvykem, ne-li přímo nepsaným pravidlem, do kódu algoritmů vkládat také komentáře. Jedná se o vymezené části obsahující libovolný text, který nemá žádný vliv na běh algoritmu. Komentáře jsou však velmi užitečné pro celkové pochopení algoritmu, ať třetí osobou, tak samotným autorem algoritmu, vrátí-li se k němu po čase.

Komentáře mohou být v programu Algoritmy dvojího typu: blokové a řádkové. Blokové jsou vymezeny složenými závorkami **{** a **}**. Všechno mezi těmito závorkami, včetně jich samotných, je považováno za komentář a jako takové při zpracování kódu zcela ignorováno. Takto vymezený blok komentáře může být jak přes několik řádků, tak i pouze uprostřed řádku s jinak platným příkazem, viz následující ukázka:

```
{
  Toto je komentář
  na více řádcích.
}
a := 3 * 4 { + 5};
```

V této ukázce by hodnota proměnné a byla 12, neboť **{+ 5}** je pouhý komentář.

Řádkové komentáře na rozdíl od blokových mají pouze dvojznak který je začíná, neboť jejich konec je vymezen koncem řádku. Tímto dvojznakem jsou dvě lomítka za sebou (**//**). Vše za nimi následující až do konce daného řádku je také považováno za komentář a jako takové při zpracování algoritmu ignorováno.

```
a := 3 * 4; // a bude od teď 12
```

5.1.6. Středníky

Nyní si ještě vysvětlíme problematiku psaní či nepsaní středníků v algoritmu. Jsou totiž jejich důležitou součástí a pokud nejsou uvedeny správně, neprojde algoritmus ani základní kontrolou. Středníky v algoritmech určují konec příkazu a v zásadě platí, že by měly následovat za každým z nich. Existují ovšem i výjimky od tohoto pravidla, takže to vezměme pěkně popořádku.

Za jednoduchými příkazy (**čti**, **napiš** a přiřazení) se středníky píší vždy, pokud za nimi nenásleduje klíčové slovo **jinak**. Za příkazy cyklu (tedy za klíčovým slovem **opakuj**) a větvení (za klíčovým slovem **pak**) se středníky nepíší nikdy, neboť za nimi následuje další příkaz či klíčové slovo **začátek**. Za klíčovým slovem **za-**

čátek se středník také nepíše, neboť není příkazem. Za klíčovým slovem **konec** se naopak středník píše vždy (nenásleduje-li **jinak**), protože jím v podstatě končí příkaz, který stojí nad tímto blokem (cyklus či větvení). Za posledním klíčovým slovem **konec**, jež zakončuje celý algoritmus se nepíše buď nic nebo tečka.

V podstatě tedy platí, že středník se píše za každým příkazem, nenásleduje-li klíčové slovo **jinak**, přičemž „nadpříkazy" (cyklus a větvení) končí až za svým posledním „podpříkazem". Před (ani za) **jinak** se středník nepíše nikdy, protože příkaz větvení, jehož je **jinak** pokračováním, končí až za blokem této alternativní podmínky.

Následuje názorný příklad:

```
{ Největší ze tří }
začátek
  // Načtení tří hodnot
  čti(x);
  čti(y);
  čti(z);
  // Porovnání
  jestliže x > y pak
    jestliže x > z pak
      největší := x
    jinak
      největší := z
  jinak
    jestliže y > z pak
      největší := y
    jinak
      největší := z;
  // Vypsání výsledku
  napiš('Největší zadané číslo je ', největší);
konec
```

5.2. Správné formátování kódu

Formátování kódu algoritmu je v podstatě pouze jeho formální úprava, která nemá na jeho běh žádný vliv a díky tomu si každý může psát algoritmy takovým stylem, jaký mu nejlépe vyhovuje. Nicméně vhodné formátování kódu mnohdy až několikanásobně zvyšuje jeho přehlednost a dobře naformátované algoritmy je poté celkem snadné „číst", studovat jejich podstatu, vyhledávat v nich chyby, opravovat

je a učit se z nich. Z těchto důvodů si zde nastíníme jeden z (v dlouhodobé praxi prověřených) příhodný způsob, jak algoritmy vhodně formátovat.

Základním pravidlem správného formátování je, že každý příkaz by měl být na zvláštním řádku. To zvyšuje nejen přehlednost, ale také jedině v tomto případě je možné algoritmus efektivně krokovat.

Správně

```
začátek
  příkaz;
  příkaz;
  příkaz;
konec
```

Nesprávně

```
začátek
  příkaz;  příkaz;  příkaz;
konec
```

Dalším podstatným pravidlem je odsazování podpříkazů. Odsazováním se rozumí přidání mezer na začátek každého řádku s příkazem tak, aby tento začínal až na příslušné úrovni. Tato úroveň je určena tím, kolik nadřazených příkazů (mohou jimi být pouze příkazy cyklu či větvení) je nad tím právě řešených. Počet mezer je pak násobkem čísla této úrovně a odsazovací jednotky, která je v programu Algoritmy stanovena na dvě mezery. Hlavní blok programu, který sám začíná na nulté úrovní, se do toho také počítá.

Správně

```
začátek
  příkaz;
  jestliže x > 3 pak
    příkaz
  jinak
    příkaz;
konec
```

Nesprávně

```
začátek
příkaz;
jestliže x > 3 pak
příkaz
jinak
příkaz;
konec
```

Klíčová slova uvozující blok (`začátek` a `konec`) mají úroveň stejnou jako příkaz nadřazený tomuto bloku.

Správně

```
začátek
  pro i od 1 do 5 opakuj
  začátek
    příkaz;
    příkaz;
  konec;
konec
```

Nesprávně

```
začátek
  pro i od 1 do 5 opakuj
    začátek
    příkaz;
    příkaz;
    konec;
konec
```

Pokud je nějaký řádek příliš dlouhý, ve vhodném místě jej zalomíme a pokračujeme v něm na řádku dalším, který bude odsazen o dvě jednotky (čtyři mezery) více, než je příkaz, jehož je toto pokračováním.

Správně

```
začátek
  x := ((3 * 4) - 2 + 1) *
       (10 + 1) * 12 / 2 ;
konec
```

Nesprávně

```
začátek
  x := ((3 * 4) - 2 + 1) *
  (10 + 1) * 12 / 2 ;
konec
```

U zalamování řádků je však více než kde jinde vhodné neřídit se zcela tímto pravidlem a přizpůsobit si úroveň odsazení tak, aby nějakým způsobem logicky začínala na co nejvhodnějším místě, usnadňujícím orientaci v tomto dlouhém příkazu. Například aby začínala na úrovni závorky, v níž se nachází.

```
začátek
  x := ((((3 + 4) * (7 + 2)) /
          (((5 + 2) * (4 - 3)) * 2)) ^
         (3 + 2));
konec
```

U hodně dlouhých příkazů se však doporučuje si je rozdělit na více kratších a ty pak najednou vyhodnotit.

Dobrým zvykem je také vkládání mezer mezi operátory ve výrazech. V některých případech je samozřejmě vhodnější tyto mezery vynechat (například u mocnění), ale v zásadě je spíše lepší tyto mezery dodržovat. V každém případě to však platí u dvojznaku přiřazení „:=“.

Správně

```
začátek
  čti(d);
  S := pi * d^2 / 4;
konec
```

Nesprávně

```
začátek
  čti(d);
  S:=pi*d^2/4;
konec
```

U čárek v příkazu `napiš` a v souřadnicích pole platí to samé, jako při psaní obyčejného textu, tedy že před ní mezera není a za ní naopak ano.

Správně

```
začátek
  pole[1, 1] := 1;
  napiš('První je ', pole[1, 1]);
konec
```

Nesprávně

```
začátek
  pole[1 ,1] := 1;
  napiš('První je ',pole[1,1]);
konec
```

Co se týče komentářů, tak ty je vhodné psát několika různými způsoby. Jedním z nich je při nadpisu části algoritmu odsadit jej na stejnou úroveň jako má následující příkaz, který je prvním ze série příkazů tímto komentářem popisovaných.

```
začátek
  // Načtení hodnot
  čti(a);
  čti(b);
  // Porovnání
  jestliže a > b pak
    větší := a
  jinak
    větší := b;
  // Vypsání výsledku
  napiš('Větší je ', větší);
konec
```

Pokud chceme komentovat pouze jeden příkaz je vhodnější tento komentář napsat za něj na tentýž řádek. Toto je zvlášť vhodné, komentujeme-li takto několik řádků za sebou, každý samostatně. V tom případě, je-li to z kapacitních důvodů možné, je vhodné i tyto komentáře zarovnat pod sebe.

```
začátek
  čti(r);                          // Načteme poloměr
  o := 2 * pi * r;                 // Vypočítáme obvod
  S := pi * r ^ 2;                 // Vypočítáme obsah
  napiš('o = ', o, ', S = ', S);   // Vypíšeme výsledky
konec
```

Je také dobrým zvykem ještě před samotný hlavní blok algoritmu na první řádek (či více) vložit blokový komentář s textem stručně charakterizujícím každý algoritmus. Díky tomu je pak možné si jeho význam přečíst zde a ne se po něm složitě pídit ve struktuře samotného algoritmu. Je tu také prostor pro jméno autora algoritmu, a to buď přímo v jeho úvodním popisu, nebo dole za finálním slovem `konec`.

```
{ Součet dvou čísel. }
začátek
  čti(a);
  čti(b);
  c := a + b;
  napiš('Součtem čísel ', a, ' a ', b, ' je číslo ', c);
konec
{ Autor: Jan Novák }
```

Výjimečné postavení má někdy klíčové slovo **`jinak`**. V základu na něj lze samozřejmě aplikovat všechna výše uvedená pravidla, nicméně existuje několik celkem častých případů, kdy je vhodné k němu přistupovat individuálně.

Jedním z těchto případů je vícenásobné větvení, kdy je například potřeba jednu hodnotu otestovat na několik (více než dvě) možností, přičemž jen jedna z nich může platit. Pokud by se tedy tyto příkazy větvení odsazovali klasicky stále dál a dál, vznikla by velmi nepřehledná a z okrajů obrazovky doprava mizející „jitrnice“. Proto je vhodné vždy rovnou za slovo **`jinak`** napsat na tentýž řádek **`jestliže`** s další podmínkou.

```
začátek
  čti(x);
  jestliže x > 0 pak
    jestliže x < 10 pak
      napiš('Zadané číslo má jednu cifru.')
    jinak jestliže x < 100 pak
      napiš('Zadané číslo má dvě cifry.')
    jinak jestliže x < 1000 pak
      napiš('Zadané číslo má tři cifry.')
    jinak jestliže x < 10000 pak
      napiš('Zadané číslo má čtyři cifry.');
konec
```

Další specialitou klíčového slova **jestliže** je možnost ušetření jednoho až dvou řádků, což u delší algoritmů může být přínosné. Vejdou-li se totiž díky tomu celé na jednu obrazovku či papír, stávají se lépe čitelnými. Konkrétně jde o případ, kdy je před klíčovým slovem **jinak** slovo **konec**, nebo za ním následuje slovo **začátek**, či obojí. V těchto případech je možné tato tři slova (případně jen dvě) napsat na jediný řádek.

```
začátek
  čti(x);
  jestliže x = 0 pak
  začátek
    napiš('Zadané číslo je nula.');
    sign := 0;
  konec jinak
    jestliže x > 0 pak
    začátek
      napiš('Zadané číslo je kladné.');
      sign := 1;
    konec jinak začátek
      napiš('Zadané číslo je záporné.');
      sign := -1;
    konec;
konec
```

Program Algoritmy disponuje funkcí, která dokáže na aktuální kód algoritmu aplikovat základní pravidla formátování (viz kapitola 3.11.1 Automatické formátování kódu, str. 23). Je však určitě lepší rovnou psát algoritmus dle těchto pravidel, což umožňuje i aplikovat ony drobné výjimky, jež program identifikovat nedokáže.

5.3. Ukázkové algoritmy

V této kapitole je uvedeno pouze několik hotových algoritmů, jakožto ukázka syntaktické i formální správnosti.

Průměr ze zadaných čísel

```
začátek
  suma := 0;
  počet := 0;
  x := 1; // Aby to poprvé prošlo podmínkou cyklu dokud
  dokud x <> 0 opakuj
  začátek
    x := 0; // Výchozí hodnota, aby pro konec stačilo jen Enter
    čti(x);
    suma := suma + x;
    počet := počet + 1;
  konec;
  počet := počet - 1;      // Poslední byla 0, která se nepočítá
  průměr := suma / počet;  // Výpočet průměru
  napiš('Průměr ze zadaných čísel je ', průměr);
konec
```

Fibonacciho generátor náhodných čísel

```
začátek
  počet := 15; // Požadovaný počet hodnot
  M := 100;    // Rozsah generovaných hodnot
  // Libovolně zvolené první dvě hodnoty
  x[1] := 32;
  x[2] := 63;
  // Vygenerování "náhodných" čísel od 0 do M-1
  pro i od 3 do počet opakuj
    x[i] := (x[i-1] + x[i-2]) mod M;
konec
```

Lehmerův generátor pseudonáhodných čísel

```
začátek
  počet := 15;  // Požadovaný počet hodnot
  // Parametry generátoru (nastavena volba ESSL)
  a := 16807;
  M := 2^31-1;
  // Libovolně zvolím první hodnotu
  x[1] := 123456;
  // Vygenerování "náhodných" čísel od 0 do M-1
  pro i od 2 do počet opakuj
    x[i] := (a * x[i-1]) mod M;
konec
```

Vypsání posloupnosti

```
začátek
  n := 10; // Počet hodnot
  // Definice posloupnosti náhodných hodnot (od 1 do 99)
  pro i od 1 do n opakuj
    a[i] := ((random * 100) div 1) mod 100;
  // Převedení hodnot posloupnosti do jednoho textového řetězce
  s := '';
  pro i od 1 do n opakuj
    jestliže s = '' pak
      s := a[i]
    jinak
      s := s + ', ' + a[i];
  // Vypsání textového řetězce
  napiš(s);
konec
```

Algoritmus řazení - Buble sort

```
začátek
  n := 15; // Počet hodnot
  // Vygenerování pseudo náhodných hodnot (od 1 do 30)
  a[1] := 19; // První libovolně zvolená hodnota
  pro i od 2 do n opakuj
    a[i] := (3*a[i-1]) mod 31;
  // Seřazení hodnot
  c := n - 1;
  změna := true;
  dokud změna opakuj
  začátek
    změna := false;
    pro i od 1 do c opakuj
      jestliže a[i] > a[i+1] pak
      začátek
        pomocná := a[i];
        a[i] := a[i+1];
        a[i+1] := pomocná;
        změna := true;
      konec;
  konec;
konec
```

Odmocnina z X (celá část jejího čísla)

```
začátek
  čti(x);
  s := 0;
  c := 1;
  dokud s <= x opakuj
  začátek
    s := s + c;
    c := c + 2;
  konec;
  s := c div 2 - 1;
  napiš('Odmocnia z ', x, ' = ', s, '. (Skutečná honota je ',
      x^0.5, '.)');
konec
```

Nalezení minimální hodnoty v matici

```
začátek
  // Definice hodnot v matici
  m := 4;
  n := 5;
  pro i od 1 do m opakuj
    pro j od 1 do n opakuj
      a[i, j] := ((random * 100) div 1) mod 100;
  // Nalezení minimální hodnoty v poly a jejího počtu
  min := a[1, 1];
  počet := 0;
  pro i od 1 do m opakuj
    pro j od 1 do n opakuj
      jestliže a[i, j] <= min pak
        jestliže a[i, j] = min pak
          počet := počet + 1
        jinak začátek
          min := a[i, j];
          počet := 1;
        konec;
  // Vypsání vásledků
  napiš('Minimální hodnota = ', min, '.');
  napiš('Počet minimální prvků = ', počet, '.');
konec
```

Násobení matic

```
začátek
  // Rozměry matic A(m x n), B(n x r), C(m x r)
  n := 4;
  m := 5;
  r := 3;
  // Nastavení generátoru pseudonáhodných hodnot
  x := 19;
  rozsah := 31;
  // Definice hodnot matice a pseudonáhodnými hodnotami
  pro i od 1 do m opakuj
    pro j od 1 do n opakuj
    začátek
      x := (3*x) mod rozsah;
      a[i, j] := x;
    konec;
  // Definice hodnot matic B pseudonáhodnými hodnotami
  pro i od 1 do r opakuj
    pro j od 1 do n opakuj
    začátek
      x := (3*x) mod rozsah;
      b[j, i] := x;
    konec;
  // Vynásobení matic C = a x B
  pro i od 1 do m opakuj
    pro j od 1 do r opakuj
    začátek
      s := 0;
      pro k od 1 do n opakuj
        s := s + a[i, k] * b[k, j];
      c[i, j] := s;
    konec;
konec
```

6. Stránky projektu

Vývoj programu Algoritmy je projekt, který určitě zasluhuje vlastní webové stránky. Jakožto freeware učební pomůcku je totiž žádoucí učinit jej volně dostupným všem možným zájemcům, poskytnout jim základní informace o něm ještě před jeho stažením a ty více skeptické zaujmout a nalákat k jeho používání. Právě k tomu tedy slouží tyto oficiální stránky projektu.

6.1. Umístění

Nejdůležitější informací o stránkách je samozřejmě jejich adresa. Ta tedy zní algds.cronos.cz.

Tyto stránky byly vytvořeny v jazyce PHP s využitím databázového systému MySQL a jejich konečná XHTML podoba je validní.

6.2. Jednotlivé stránky

Samotný web obsahuje několik samostatných stránek, které si uživatelé mohou prohlédnout. Přepínání mezi nimi se provádí kliknutím myší na příslušné tlačítko v hlavním menu v horní části stránky.

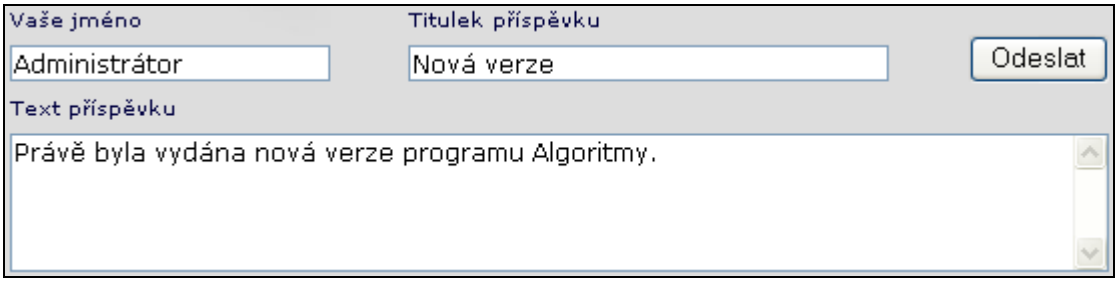
Obr. 6.1 - Hlavní menu na stránkách projektu Algoritmy

6.2.1. Diskuze

Na stránkách jakéhokoli projektu by neměla scházet otevřená diskuze. Ta je dostupná i zde. Díky ní se uživatelé programu Algoritmy mohou veřejně vyjadřovat ke svým zkušenostem s ním, upozorňovat na problémy, pokládat v plén své nápady a diskutovat s autorem programu. Všichni tak mohou dostat velmi rychlé odpovědi na své otázky, kteréžto i nadále zůstávají dostupné i všem ostatním.

Přispívání do diskuze je velmi snadné. Není k tomu třeba ani žádné registrace. Slouží k tomu formulář situovaný nahoře, nad samotným výpisem z diskuze (viz Obr. 6.2). Stačí vyplnit své jméno či přezdívku, titulek příspěvku (není povinný) a text samotného příspěvku. V textu příspěvku je možné používat tyto tagy:

-
-
- <u></u>
- <code></code>
- <a>



Obr. 6.2 - Přispívání do diskuze

6.2.2. Obrázky

Tato stránka je zde spíše jako propagační a slouží především osobám, které se rozhodují, jestli program Algoritmy stáhnout a vyzkoušet či nikoli. Najdou zde totiž několik screenshotů („vyfocených" oken programu), které poslouží pro lepší utvoření názoru o celém programu bez nutnosti jeho přímého spuštění.

6.2.3. Historie verzí

Zde se nachází seznam všech verzí programů (od nejstarší po aktuální) a u každé je uvedeno, k jakým změnám v ní v programu Algoritmy došlo.

Číslo verze se skládá ze čtyř samostatných číslic. První dvojice je číslem verze (hlavní a vedlejší) a druhá dvojice je číslem buildu (nové kompilace programu s pouze drobnými opravami). V rámci jedné verze tedy může být i několik buildů. U každého čísla verze je také uvedeno datum, kdy byla vydána.

Verze jsou řazeny sestupně, stejně jako změny v nich provedené. Nejnovější informace jsou tedy vždy nahoře. Titulek poslední verze je zároveň odkazem k přímému stažení programu Algoritmy.

6.2.4. Stáhnout

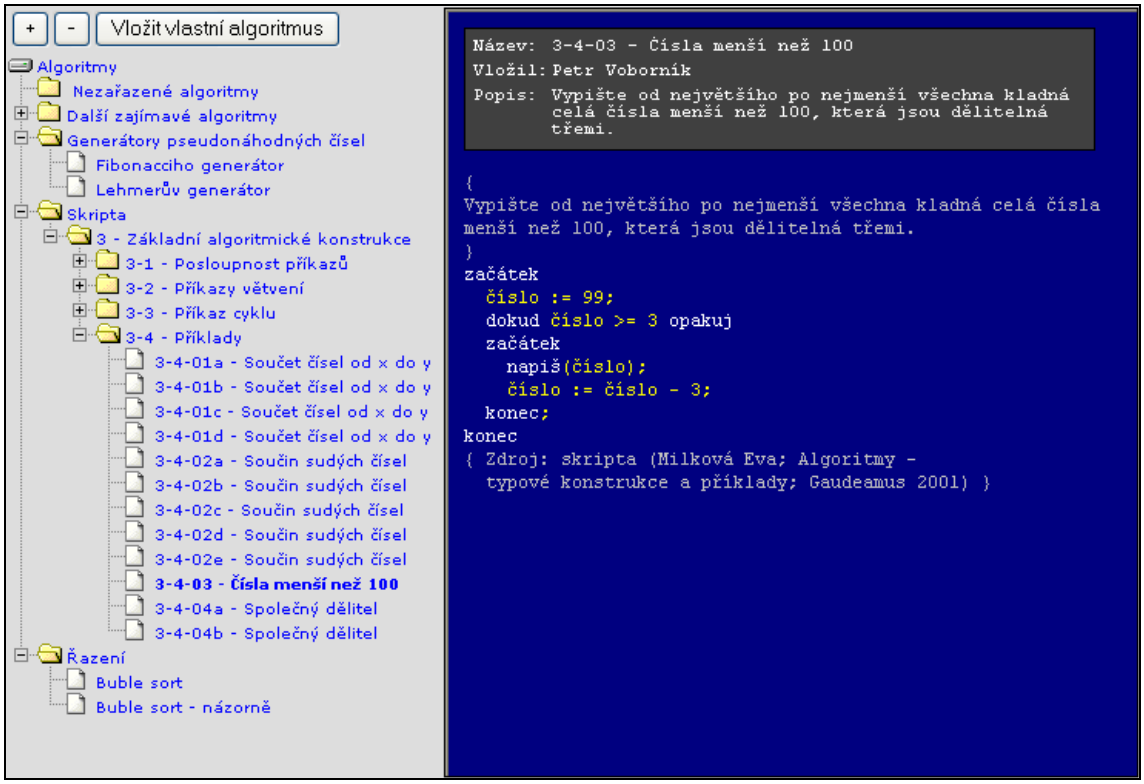
Tato stránka slouží pouze jako snadno identifikovatelný odkaz pro stažení aktuální verze programu Algoritmy. Program je ke stažení v komprimované formě jako soubor *algoritmy.zip*. Kromě samotného programu je v archivu ještě přibalen podadresář *Languages*, obsahující soubory s různými jazykovými verzemi programu, a podadresář *Algoritmy*, v němž se nachází řada hotových algoritmů, spustitelných právě v programu Algoritmy.

6.2.5. Algoritmy

Na této stránce se nachází databáze hotových algoritmů na ukázku. Ty jsou seřazeny do stromové struktury složek, dle jejich funkce a původu (viz Obr. 6.3).

Tento strom byl realizován s pomocí hotového volně šiřitelného řešení Triga Tree Menu (viz http://www.softcomplex.com/products/tigra_tree_menu) a jedná se o JavaScript řešení. Díky tomu veškeré otevírání a zavírání složek nevyžaduje opětovné načítání stránky a pohyb v něm je celkem rychlý. Tlačítka + a -, která jsou nad tímto stromem, umožňují jediným kliknutím otevřít kompletní strom včetně všech podsložek (+), či je naopak naráz všechny pozavírat (-).

Po kliknutí na název vybraného algoritmu v konkrétní složce se tento algoritmus načte do prostoru v pravé části stránky, který je pro ně vyhrazený. Odtud si je možné algoritmus zkopírovat přes schránku (označit jej a stisknout Ctrl+C) přímo do programu Algoritmy, či jakéhokoli jiného textového editoru (přepnout se do něj a stisknout Ctrl+V).

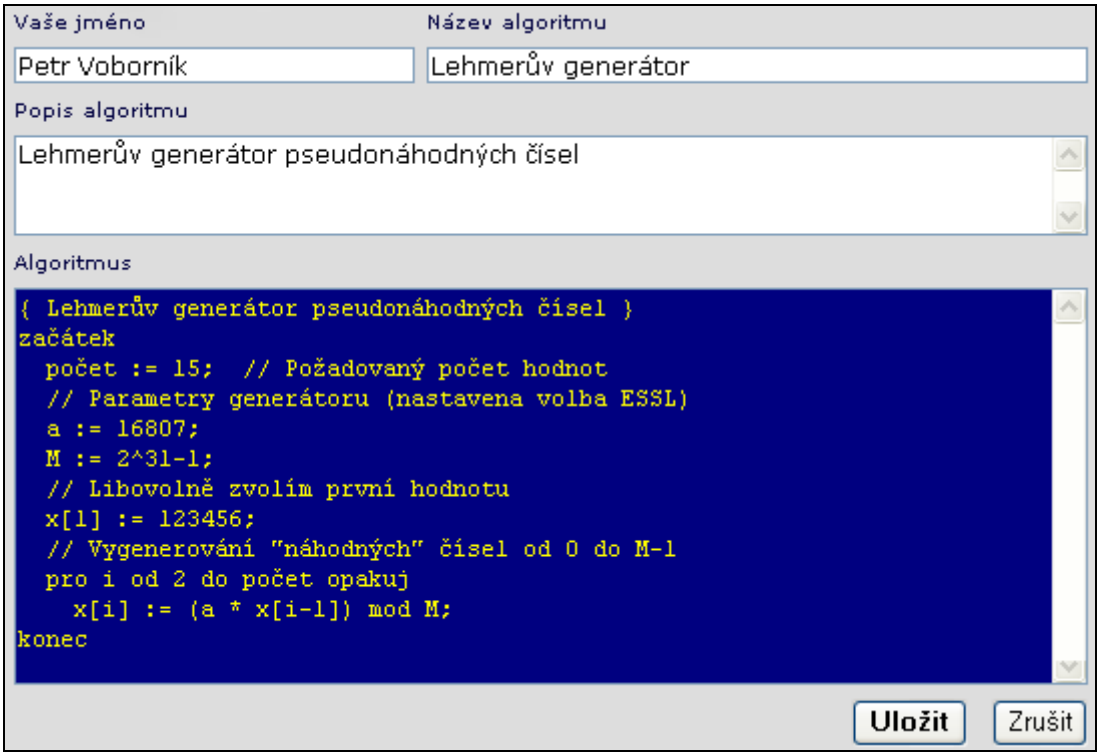


Obr. 6.3 - Stránka s ukázkovými algoritmy

Kromě prohlížení a kopírování algoritmů mají uživatelé také možnost sem vkládat vlastní algoritmy. Za tímto účelem je zde tlačítko Vložit vlastní algoritmus, které po kliknutí na něj otevře formulář pro přidání vlastního algoritmu (viz Obr. 6.4).

Zde stačí vyplnit své jméno, název vkládaného algoritmu, jeho stručný popis a samotný algoritmus. Je dobré si jej nejprve vyzkoušet v programu Algoritmy, je-li skutečně funkční a měl by také dodržovat správné formátování (viz kapitola 5.2 Správné formátování kódu, str. 56). Do algoritmu není povoleno vkládat jakékoli tagy. O zvýraznění klíčových slov se postará funkce až v okamžiku jeho zobrazení.

Takto nově vložené algoritmy se automaticky uloží do složky *Nezařazené algoritmy*. O jejich správné zařazení se pak již postará administrátor webu.



Obr. 6.4 - Vložení vlastního algoritmu

6.3. RSS kanály

Stránky také podporují RSS kanály. Je zde jeden společný, který obsahuje veškeré změny na webu v různých kategoriích a tři samostatné pro každou z těchto kategorií. Na stránce každé kategorie mající vlastní RSS kanál se nachází úplně vespod ikona RSS, která je zároveň odkazem na tento kanál. Výstupy těchto RSS kanálů jsou ve formátu 2.0 a jsou přímo určeny pro libovolnou RSS čtečku.

Individuální kategorie RSS kanálů na těchto stránkách jsou tedy tyto:

- Diskuze http://algds.cronos.cz/w_rss.php?sec=discusion
- Historie verzí http://algds.cronos.cz/w_rss.php?sec=history
- Algoritmy http://algds.cronos.cz/w_rss.php?sec=algorithm
- *Souhrnná* http://algds.cronos.cz/w_rss.php

6.4. Redakční systém pro administraci stránek

Aby bylo možné tyto stránky rychle, efektivně a odkudkoli spravovat, mají vlastní redakční systém pro jejich administraci. Ten zahrnuje správu diskuze, obrázků, historií verzí, souboru ke stažení a algoritmů. Díky ní je veškeré tyto sekce možné spravovat přímo přes webové rozhraní, po přihlášení se do této administrace (pomocí skrytého odkazu, jména a hesla).

6.4.1. Statistiky

Další funkcí po přihlášení se do administrátorského režimu stránek je zpřístupnění tabulky se statistikami návštěvnosti jednotlivých stránek. Díky ní je možné zjistit, kolik unikátních přístupů (hitů) jednotlivá stránka měla a kdy k tomu naposledy došlo. Následující ukázka této tabulky je z 26.1.2006.

Datum poslední	URL	Počet
26.01.2006 23:16:12	hity celého webu	441
26.01.2006 09:51:46	index.php?sec=about	378
26.01.2006 01:33:48	index.php?sec=download	223
23.01.2006 23:21:06	w_algorithm.php	282
23.01.2006 23:20:45	index.php?sec=algorithm	136
18.01.2006 23:08:39	index.php?sec=history	170
16.01.2006 16:00:58	index.php?sec=screens	86
16.01.2006 16:00:56	index.php?sec=discusion	136
16.01.2006 15:59:50	index.php?sec=contact	55
12.01.2006 16:18:07	w_screen.php	74
20.12.2005 13:52:06	w_rss.php?sec=discusion	8
20.12.2005 13:52:06	w_rss.php?sec=algorithm	11
20.12.2005 13:52:06	w_rss.php?sec=history	8
20.12.2005 13:52:06	w_rss.php	7

7. Výsledky

Výsledkem je především vytvořený program Algoritmy, plně vyhovující ve všech potřebných bodech, tedy pohodlné psaní algoritmů, jejich kontrola, spouštění a krokování. Vzhledem k jeho úspěšnému využití ve výuce a stále masivnějšímu užívání studenty pro samostudium, lze považovat jeho implementaci za úspěšnou.

Webové stránky k tomuto projektu pak činí program Algoritmy a informace o něm dostupnými širší veřejnosti a umožňují jeho pravidelné aktualizace.

Součástí diplomové práce je také podrobný popis ovládání programu, bez kterého by řada funkcí nejspíš zůstala běžnému uživateli skryta.

V práci se též podařilo zdokumentovat vnitřní strukturu programu a osvětlit princip, kterým algoritmy zpracovává. Dále pak práce pojednává o způsobu a formálnosti psaní těchto algoritmů a poskytuje řadu doporučení, jak k této činnosti přistupovat.

8. Závěry a doporučení

Program Algoritmy, je výbornou pomůckou pro začátečníky v oboru algoritmizace, na které si mohou zdarma vyzkoušet a osvojit techniku psaní a fungování algoritmů.

Za jeden z největších přínosů programu lze považovat možnost nasazení při výuce, kdy bude s jeho pomocí možné více názorněji prezentovat průběh jednotlivých algoritmů a studenti tak snadněji pochopí jejich princip. Další důležitou výhodou programu je také jeho užití při samostatném studiu.

V neposlední řadě je díky tomuto programu možné názorně demonstrovat a zpřístupnit širší veřejnosti v jednotné funkční formě určité specifické algoritmy, znalost jejichž principu je nezbytná k pochopení podstaty látky probírané i v jiných předmětech (např. matematika, statistika apod.).

Díky webovým stránkám k tomuto projektu lze přes diskusní fórum či e-mail získávat cennou zpětnou vazbu od koncových uživatelů programu – studentů. Pomocí ní je pak možné program i nadále vylepšovat tak, aby stále lépe vyhovoval jejich potřebám.

Program Algoritmy zatím plní očekávání v něj kladená a i do budoucna má v sobě velký potenciál. Především se jedná o jeho význam při spolupráci s jinými univerzitami po celém světě, představování programu na konferencích, jeho publikování v odborných časopisech, případně i vydání v rámci publikace zaměřené na problematiku algoritmů a její výuky v tomto programu.

9. Seznam obrázků

10. Seznam použité literatury

1. **Milková, E.:** *Algoritmy, typové konstrukce*. Gaudeamus, Hradec Králové, 2001

2. **Milková, E.:** *Algoritmy v příkladech*. Gaudeamus, Hradec Králové, 2004

3. **Wróblewski, P.:** *Algoritmy, datové struktury a programovací techniky*. 1. vyd. Přel. M. Michalek, B. Kiszka. Computer press, Brno, 2004. Kapitola 5.6.1, Binární stromy a aritmetické výrazy, s. 142 – 147

Soubory z internetu

4. **Kozel, T.:** *EduPage – Přednášky - PRO1o - přednáška č. 11 (Stromy)*. Česká republika, 2006 (leden, 25., 2006). Přístup z internetu: URL: http://lide.uhk.cz/home/fim/ucitel/kozelto1/www

5. **Mach, P.:** *Python – Slovníček a zkratky*. Česká republika, 2005 (citace leden, 25., 2006). Přístup z internetu: URL: http://wraith.iglu.cz/python/slovnicek.php

Zadání práce

Příjmení a jméno autora: Voborník Petr

Specializace studia: IM
Příjmení a jméno vedoucího práce: Milková Eva

Přesný název práce:
Programovací jazyk pro podporu výuky algoritmů

Cíl práce:
Vytvoření prostředí umožňujícího převod algoritmů, formálně vyjádřených pomocí pseudokódu, do programovacího jazyka, a zajišťujícího spuštění a trasování těchto algoritmů.

Osnova práce:
- Vytvoření programu v programovacím jazyku Borland Delphi
- Vytvoření ukázkových příkladů
- Textové zpracování poznatků

Projednáno dne: 19.1.2005

Podpis studenta: …… Podpis vedoucího práce: ……