MÁRIA BIELIKOVÁ
PAVOL NÁVRAT

Funkcionálne a logické programovanie

S T U
F I I T
SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
Fakulta informatiky a informačných technológií

Edícia učebných textov
informatiky a informačných technológií

Funkcionálne a logické programovanie

Lektori: prof. RNDr. Jozef Kelemen, DrSc., doc. Dr. Ing. Dušan Kolář

Grafická úprava: Mária Bieliková
Obálka: Peter Kaminský

PUBLIKÁCIU PODPORILO ZDRUŽENIE
GRATEX IT INŠTITÚT
v rámci fondu GraFIIT www.gratex.com

Vydala Slovenská technická univerzita v Bratislave v Nakladateľstve STU,
Bratislava, Vazovova 5.

Schválilo vedenie Fakulty informatiky a informačných technológií STU v Bratislave
dňa 10. 11. 2009, uznesenie číslo 2009.25.1, pre študijný program Informatika, 1. stupeň.

ISBN 978-80-227-3225-3

PREDSLOV

UČEBNICA VYSVETĽUJE PARADIGMY PROGRAMOVANIA

Dostávate do rúk učebnicu, ktorá poskytuje záujemcom o softvérové inžinierstvo, a to najmä o programovanie, ale aj o ďalšie etapy životného cyklu softvéru, rôzne pohľady na tvorbu softvéru. Programovanie je jadrom tohto procesu v tom zmysle, že jeho účelom je bezprostredne zostrojiť samotné programy, ktoré majú patriť do navrhovaného systému. Tu sa rozhoduje, nakoľko bude výsledný softvérový produkt efektívny, či bude správny, spoľahlivý a pod. Hoci tzv. tradičné postupy alebo paradigmy programovania sa v mnohom osvedčili, o čom svedčí napokon značná časť z toho množstva existujúceho softvéru fungujúceho dnes už skoro všade okolo nás, stále zostáva mnoho otvorených otázok. Týkajú sa napríklad príčin mnohých chýb v softvéri, ktoré sa v ňom vyskytujú a spôsobujú niekedy veľmi neblahé následky. Zameriavajú sa tiež na zefektívnenie samotného procesu programovania a jeho priblíženie sa k používateľovi.

UČEBNICA SA ZAMERIAVA NA FUNKCIONÁLNE A LOGICKÉ PROGRAMOVANIE

Jedna z ciest pri hľadaní odpovedí na naznačené otázky vedie k použitiu iných než tradičných (procedurálnej, a čoraz častejšie dnes už aj objektovo-orientovanej) paradigiem programovania. Medzi takéto alternatívne paradigmy možno zaradiť napríklad vizuálne programovanie alebo programovanie tabuľkových procesorov, ktoré sú veľmi efektívne vo svojich oblastiach aplikácií. Všeobecný charakter majú funkcionálne a logické programovanie. Práve týmto sa budeme v tejto učebnici venovať najviac.

POUŽÍVAME PROGRAMOVACIE JAZYKY LISP A PROLOG

Základné princípy vysvetľujeme v kontexte dvoch programovacích jazykov: lisp (list processor) v súvislosti s funkcionálnym programovaním a prolog (programming in logic) v súvislosti s logickým programovaním. Tieto však ostávajú v platnosti nezávisle od zvoleného programovacieho jazyka a možno ich jednoducho podľa potreby použiť.

UČEBNICA JE URČENÁ PRE ZAČIATOČNÍKOV AJ SKÚSENÝCH PROGRAMÁTOROV

Učebnicu môžete použiť či ste začiatočník alebo skúsený programátor. Môže slúžiť aj ako úvod do programovania v jazyku lisp a prolog. Zároveň poskytuje prehľad základných paradigiem programovania.

Predchádzajúce znalosti programovania nie sú nevyhnutné. Napriek tomu sme presvedčení, že aj v prípade, ak máte skúsenosti s funkcionálnym a logickým programovaním, nájdete v učebnici informácie, ktoré pomôžu lepšie pochopiť jednotlivé prístupy k programovaniu, porovnať ich a podporia rozhodnutia kedy a ako ich použiť. Ako vysokoškolská učebnica je určená najmä poslucháčom informatiky študujúcim predmet Funkcionálne a logické programovanie.

Učebnica obsahuje množstvo schém riešenia rôznych situácií a problémov programovania. Je rozdelená na dve časti, ktoré sa vzťahujú na znalosti o funkcionálnom programovaní a o logickom programovaní. V úvode stručne predstavíme niektoré často používané paradigmy programovania. Ide o procedurálne, objektovo-oriento-

vané, funkcionálne, logické programovanie, programovanie ohraničení a vizuálne programovanie.

DIEL I SA VENUJE FUNKCIONÁLNEMU PROGRAMOVANIU

Diel I učebnice sa venuje funkcionálnemu programovaniu. Vysvetľujeme základné princípy funkcionálneho programovania, vlastnosti funkcionálneho programu na príkladoch programovacieho jazyka lisp.

Funkcionálne programovanie možno použiť aj v prípade, keď implementačným jazykom je napr. procedurálny jazyk. Preto sa v učebnici skôr sústreďujeme na metodológiu ako na jazyk a všetky jeho vlastnosti. Máte možnosť oboznámiť sa so základnými programovacími technikami (postupmi), ktoré sa používajú v súvislosti s tvorbou funkcionálnych programov.

DIEL II SA VENUJE LOGICKÉMU PROGRAMOVANIU

Diel II sa zaoberá logickým programovaním, konkrétne na príkladoch programovacieho jazyka prolog. Vysvetľujeme základné myšlienky odvodzovania riešenia z logického programu. Nechýbajú ani časti, kde sa môžete oboznámiť s rozšíreniami logického programovania, ktoré umožňujú efektívne riešenie problémov.

Štruktúry oboch dielov sa veľmi podobajú. Jednotlivé kapitoly končia zhrnutím podstatných informácií. V každej kapitole uvádzame cvičenia (s riešeniami vybratých cvičení v prílohe). Prvý a druhý diel predstavujú samostatné celky a možno ich študovať v ľubovoľnom poradí.

PRÍLOHY SÚ POMÔCKOU PRI PROGRAMOVANÍ

Učebnica obsahuje prílohu niektorých vstavaných funkcií jazyka Common lisp (ANSI štandard pre Common lisp) a predikátov jazyka prolog (ISO/IEC štandard pre prolog). Vybrali sme najmä tie, ktoré treba na pochopenie a precvičenie typov problémov vysvetľovaných v učebnici. Tieto jazykové konštrukcie predstavujú určité spoločné jadro (snáď len s výnimkou vstupno-výstupných funkcií) viacerých implementácií jazyka Common lisp a prolog. Príklady v učebnici sa takto stávajú nezávislé od konkrétnej implementácie príslušného jazyka. V prílohe uvádzame tiež riešenia vybraných cvičení.

NÁMETY DO UČEBNICE POSKYTLI ŠTUDENTI

Hlavným zdrojom učebnice boli prednášky v predmete Funkcionálne a logické programovanie na Fakulte elektrotechniky a informatiky Slovenskej technickej univerzity v Bratislave. Do našej viacročnej skúsenosti patrí aj poznatok, že práve pri štúdiu týchto dvoch paradigiem sú viaceré témy, ktoré sa zdajú byť mnohým študentom jasné, ale ich dokonalejšie pochopenie môže robiť problémy. Práve na tieto sme sa v učebnici najviac sústredili. Pri písaní učebnice sme vychádzali zo skúseností pri práci so študentami. Našim cieľom je priblížiť problematiku alternatívnych paradigiem programovania jednoduchým spôsobom s dôrazom na príklady a praktické použitie v inžinierskej praxi.

NOVÉ V TOMTO VYDANÍ

Učebnica Funkcionálne a logické programovanie vychádza zo skrípt s rovnomenným názvom (Bieliková, M., Návrat, P., 1997). Revidovali sme tie časti, ktoré spôsobovali študentom problémy. Zároveň sme učebnicu doplnili o niekoľko ďalších poznatkov, najmä v súvislosti s programovacími technikami vo funkcionálnom aj logickom programovaní.

Stanovenie typov pri tvorbe funkcionálneho programu je dôležité. Hoci systémy, ktorí vyhodnocujú výrazy zapísané v programovacom jazyku lisp patria k systémom s dynamickou kontrolou, nabádame čitateľa k systematickému prístupu k definícii funkcií dôslednejším stanovovaním typov v príkladoch.

V učebnici nájdete viac príkladov rekurzívnych definícií funkcií. Zaradili sme predikáty, ktoré zisťujú vlastnosti prvkov zoznamu a tiež rekurzívne spracovanie súboru s-výrazov. Ďalej sme doplnili a rozšírili časti, v ktorých sa venujeme programovým schémam spracovania zoznamov vo funkcionálnom aj logickom programovaní.

K programovacím technikám logického programovania sme pridali časť, ktorá sa zaoberá tzv. rozdielovými štruktúrami. Rozdielové štruktúry umožňujú zjednodušiť a zefektívniť logický program. Do textu sme ich zaradili najmä kvôli zdôrazneniu niektorých vlastností logických programov, ktoré v iných programovacích paradigmách nenájdeme.

DOPLNKOVÝ MATERIÁL DOSTUPNÝ NA INTERNETE

Doplnkový materiál k učebnici môžete nájsť v celosvetovej pavučine na adrese: http://www.dcs.elf.stuba.sk/~bielik/books/flp/. Obsahuje zdrojové texty programov v lispe a prologu, ktoré používame v učebnici. Vzhľadom na nestálu povahu informačných prameňov v Internete, v učebnici neuvádzame ďalšie odkazy súvisiace s funkcionálnym a logickým programovaním. Tieto nájdete v celosvetovej pavučine na vyššie uvedenej adrese. Veríme, že pomôžu všetkým záujemcom na náročnej ceste poznania.

POĎAKOVANIE

Skutočný život každej publikácie začína okamihom, keď sa dostane do rúk čitateľom. Čitateľ rozhoduje o ďalšom osude publikácie vyjadrením svojho postoja k vytvorenému dielu. My, ako autori, sme v predchádzajúcich troch rokoch prijímali s radosťou všetky hodnotenia a pripomienky čitateľov – najmä študentov. Ďakujeme všetkým, ktorí nám pomohli svojimi poznámkami a našli aj niekoľko chýb v programoch. Osobitne ďakujeme Mariánovi Vargovi a Tomášovi Gerhátovi, ktorí poslali podrobne zdokumentované riešenia príkladov, v ktorých našli nedostatky.

EŠTE PÁR SLOV...

Všetky príklady programov sme implementovali a otestovali v dvoch prostrediach jazyka Common lisp: Golden Common lisp 1.01 a LispWorks 4.1 a v dvoch prostrediach jazyka prolog: Arity prolog 5.0 a LPA Win-Prolog 3.3.

Aj napriek viacnásobnej kontrole nám mohli nejaké chyby v programoch uniknúť. Uvítame, ak sa v prípade, že nájdete chyby alebo máte iné pripomienky, ktoré by mohli zlepšiť text, s nami skontaktujete.[1]

Veríme, že učebnica si nájde miesto v knižnici každého programátora, ako začiatočníka tak aj pokročilého.

Autori

[1] http://www.elf.stuba.sk/~bielik/, http://www.elf.stuba.sk/~navrat/

OBSAH

ÚVOD

V úvodnej časti učebnice stručne zhrnieme základné vlastnosti jednotlivých v súčasnosti používaných paradigiem programovania, aby sme potom v ďalších častiach mohli podrobne vysvetliť dve z nich: funkcionálnu a logickú. Najprv sa však aspoň letmo dotkneme samotného pojmu paradigma, a v tejto súvislosti aj pojmov metóda a technika, ktoré tiež používame.

Jednotlivé druhy, spôsoby, či prístupy k programovaniu označujeme ako *paradigmy programovania*. Robíme tak v súlade s väčšinou odbornej literatúry a s praxou, kde sa takéto označenie zaužívalo. To však neznamená, že by v používaní rôznych metodologických pojmov v oblasti programovania panovala úplná zhoda. Nezaškodí preto zastaviť sa pri význame týchto pojmov. Patrí to do metodológie vedy, v tomto prípade do špeciálnej metodológie programovania.

Ak hovoríme o metóde tvorby programov, treba si v prvom rade uvedomiť, že tvorba programov je súčasťou procesu riešenia problému. Program je vlastne riešením problému vyjadreným v tvare, v princípe vykonateľnom na počítači. Metóda je jednoducho cesta, spôsob dosiahnutia požadovaného výsledku, t.j. nájdenia riešenia vyjadreného programom.

Takéto chápanie pojmu metóda je dosť všeobecné. Vedné disciplíny si formulujú zvyčajne množstvo vlastných zvláštnych metód, ktoré niekedy opisujú postup riešenia veľmi úzkych tried problémov. Aj v programovaní nehovoríme iba o metódach programovania, ale aj o spôsoboch programovania, o metodikách programovania a o programovacích postupoch. Sú to špeciálne metódy, ktoré opisujú čiastkové cesty riešenia, napr. iba v etape návrhu alebo implementácie. Jednotlivé metodiky, postupy a pod. Môžu byť navzájom alternatívou, napr. spôsob implementácie údajového typu zoznam môže byť buď pomocou vektora alebo pomocou zreťazenej voľnej pamäti. Môžu sa však aj dopĺňať, napr. po použití metodiky štruktúrovanej systémovej analýzy je vhodné použiť metodiku štruktúrovaného systémového návrhu. Práve tento druhý prípad, ktorý dáva do súvisu viaceré metodiky opierajúce sa o spoločný východiskový princíp, sa niekedy blíži k tomu, čo sa chápe pod pojmom paradigma.

PARADIGMA SÚ NÁZORY, KTORÉ SA V DANEJ OBLASTI UZNÁVAJÚ

Paradigma je vo všeobecnosti súhrn spôsobov formulácie problémov, metodologických prostriedkov ich riešenia, štandardných metodík rozpracovania a pod. Sú to názory, teórie, metódy, metodiky a pod., ktoré sa v danej oblasti uznávajú. V oblasti programovania možno považovať za paradigmu napr. procedurálne programovanie, keďže predstavuje súbor názorov, teórií, metód, metodík ako programovať pomocou príkazov a explicitne pomenovaných pamäťových miest, bez ohľadu na voľbu tohoktorého programovacieho jazyka (C, pascal, cobol), metódy programovania (napr.

štruktúrované, modulárne) a pod. Podobne sa za paradigmy označujú aj funkcionálne, logické a objektovo-orientované programovania.

Pokus o ich systematické vymenovanie možno nájsť napr. v *Úplnom klasifikačnom systéme*, ktorý sa vypracoval v rámci ACM[1] pre potreby časopisu Computing Reviews. V strome pojmov, opisujúcich softvér, je jeden podstrom označený "programovacie techniky". Programovacie techniky podľa tejto klasifikácie zahŕňajú:

- aplikatívne (funkcionálne) programovanie,
- automatické programovanie,
- súbežné programovanie, kam patrí
  - distribuované programovanie,
  - paralelné programovanie,
- sekvenčné programovanie,
- objektovo-orientované programovanie,
- logické programovanie,
- vizuálne programovanie.

Ako je vidieť, podobný obsah ako paradigma programovania má podľa ACM aj pojem programovacia technika (pojem paradigma sa v tejto klasifikácii nevyskytuje). Pravda, niekedy sa pojem programovacie techniky používa na súhrnné označenie metód, metodík a pod. v rámci jednej paradigmy programovania.

Tak napr. v predmete Programovacie techniky sa síce oboznamujeme s metódami, metodikami a pod. riešenia úloh pomocou základných algoritmov a štruktúr údajov na dostatočne abstraktnej úrovni, ktorá je spoločná pre viaceré paradigmy, avšak pri ich implementácii používame techniky typické najmä pre procedurálne (imperatívne) programovanie.

V PROCEDURÁLNOM PROGRAMOVANÍ PRÍKAZY PREDPISUJÚ VYKONANIE OPERÁCIÍ

V procedurálnom alebo imperatívnom programovaní je program v podstate postupnosť príkazov. Príkazy predpisujú vykonanie operácií. Ak neurčí riadiaca štruktúra inak, vykonajú sa príkazy v tej postupnosti, v akej sú zapísané. Programovací jazyk obsahuje jazykové konštrukcie pre vetvenie (napr. príkaz if, case, switch) a pre cyklus (napr. príkaz while-do, repeat-until). Programovací jazyk obsahuje pre zvolenú množinu operácií príkazy, ktorými ich možno volať. Na vytvorenie ďalších operácií poskytuje programovací jazyk možnosť naprogramovať ich pomocou procedúr a volať volaním procedúry vrátane možnosti odovzdania parametrov. Údaje sa ukladajú do pamäťových miest, ktoré sa v programe pomenujú pomocou premenných. Príkazy postupne menia obsah pamäťových miest tak, aby sa nakoniec dosiahol želaný výsledok.

Zo skutočnosti, že sa pracuje s premennými, ktoré priamo odkazujú na pamäťové miesta, nastáva pre programátora nutnosť organizovať spôsob ich použitia pri uchovávaní údajov v pamäti. Programátor má teda na zreteli viac vecí:

- opísať, čo sa má počítať,

[1] ACM – The Association for Computing Machinery – je medzinárodná organizácia, ktorá združuje profesionálov a záujemcov o informatiku (http://www.acm.org).

- navrhnúť celý výpočet ako postupnosť jednotlivých krokov,
- organizovať použitie pamäti počas výpočtu.

V ideálnom prípade by sa programátor mal starať iba o prvú z uvedených vecí. Hľadanie alternatívnych paradigiem programovania je motivované práve snahou zredukovať ich rozsah alebo aspoň oddeliť ich od seba. Oddelenie by umožnilo starať sa o ne rozličným ľuďom, prípadne niektoré veci zautomatizovať.

V OBJEKTOVO-ORIENTOVANOM PROGRAMOVANÍ SA PROGRAM CHÁPE AKO MNOŽINA OBJEKTOV

V objektovo-orientovanom programovaní je program v podstate množina objektov. Objekt má tieto charakteristiky:

- jedinečná totožnosť (meno),
- stav (reprezentácia atribútov pomocou údajových štruktúr vnútri objektu),
- správanie sa (množina povolených operácií).

Vykonanie operácií sa v programe opíše vzájomnou komunikáciou objektov. Objekt komunikuje s iným tak, že si s ním vymieňa správy podľa nejakého protokolu.

Objekt je teda štruktúra, ktorá zahŕňa údaje aj procedúry. Objekt si uchováva stav vo vlastných premenných a reaguje na podnety (správy) iných objektov vlastnými akciami. Spôsob reakcie objektu na daný druh správy definuje procedúra – niekedy sa nazýva metóda.

Správy sú spôsobom vyjadrenia vykonania operácií (akcií). Ide vlastne o volanie procedúry v objekte – príjemcovi.

Implementácia procedúr sa na rozdiel od ich špecifikácie považuje za skrytú v danom objekte. Volajúce objekty nevidia na definície procedúr. Skryté sú aj údaje objektu, t.j. premenné objektu sú prístupné na čítanie a zmenu iba procedúram toho objektu. *Objekt* predstavuje abstrakciu, ktorá skrýva detaily implementácie svojho stavu a správania sa. Hovorí sa aj o tzv. zapúzdrení.

Pojem *trieda* opisuje štruktúru a správanie sa množiny podobných objektov. Trieda slúži ako vzorec na konštruovanie objektov. Trieda ako množina objektov rovnakého typu predstavuje údajovú abstrakciu, čiže abstraktný údajový typ.

Medzi triedami sa definujú dedičné vzťahy (vzťah množina – podmnožina). Trieda má svoju nadtriedu (jednoduché dedenie), prípadne nadtriedy (viacnásobné dedenie), od ktorej (ktorých) dedí vlastnosti. Metóda, ktorá má spracovať správu pre daný objekt, sa hľadá najprv medzi metódami príjemcu, ktoré sú definované v deklarácii triedy, do ktorej patrí. Metóda sa tam však nemusí nachádzať. Nie je to chyba – stačí, ak sa nachádza v nadtriede tejto triedy. Ak sa nenachádza ani tam, hľadá sa stále vyššie v hierarchii tried.

Ak má niektorá trieda definovanú tú istú metódu ako niektorá z jej nadtried v hierarchii, dochádza k tzv. prekrývaniu. Takýmto spôsobom daná trieda predefinovala príslušnú metódu pre svoje objekty.

Objektovo-orientované programovanie je založené na uvedených vlastnostiach. Ich podpora je kritériom pri voľbe programovacieho jazyka. Programovací jazyk pre objektovo-orientované programovanie je taký, ktorý má tieto vlastnosti:

- podporuje objekty v zmysle údajových abstrakcií, s rozhraním navonok

v podobe pomenovaných operácií a so skrytým vnútorným stavom,

- objekty prináležia nejakému typu (alebo triede) a typy (triedy) môžu dediť atribúty od nadtypu (nadtriedy).

Programovacie jazyky smalltalk, eiffel a pod. vznikli so zámerom podporiť objektovo-orientované programovanie. Okrem toho je skupina jazykov, ktoré sú rozšíreniami jazykov, pôvodne podporujúcich inú paradigmu programovania: C++, ObjectPascal, CLOS (Common Lisp Object System). Pri vymenúvaní jazykov pre objektovo-orientované programovanie by sme nemali zabudnúť na jazyk java, ktorého použitie sa stále rozširuje najmä v súvislosti s tvorbou aplikácií pre použitie v Internete.

DEKLARATÍVNE PROGRAMOVANIE SA SÚSTREĎUJE NA TO, ČO SA MÁ RIEŠIŤ

Jeden smer odklonu od procedurálneho programovania, motivovaného dôvodmi uvedenými vyššie, vedie k posilneniu deklaratívnej stránky programovania (špecifikovať najmä, ČO sa má riešiť, a čo najmenej to, AKO sa to má riešiť). Zúži sa tak priestor medzi špecifikáciou a programom. Vykonateľná špecifikácia dobre poslúži napr. pri vytváraní softvérového prototypu. Špecifikáciu možno zostrojiť zvyčajne rýchlejšie než program. Ak nie je dostatočne efektívna, otvára sa cesta jej vylepšujúcich transformácií (tzv. optimalizácií), ktoré sa môžu aj zautomatizovať.

APLIKATÍVNY PROGRAM OPISUJE VÝPOČET VÝRAZOM

Posun smerom k deklaratívnemu prístupu k programovaniu možno sledovať pri aplikatívnom programovaní. Pri aplikatívnom programovaní sa želaný výpočet opíše výrazom. Jeho vyhodnotením sa získa požadovaný výsledok. Aplikatívne programovanie svojou vysokou úrovňou abstrahovania od toho, ako sa výpočet vykonáva chápe programovanie ako deklarovanie toho, čo sa počíta, čiže o aký výpočet ide.

Na ilustráciu uvažujme tento výraz

```
(x + y) * (x - y)
```

Výraz opisuje aplikáciu niekoľkých funkcií (sčítanie, násobenie a odčítanie). Vo výraze sa neurčujú žiadne podrobnosti výpočtu ako napr. spôsob a miesto uloženia medzivýsledkov. Ďalej možno pozorovať viaceré alternatívy čo sa týka poradia vyhodnocovania jednotlivých podvýrazov a možnosť ich paralelného vyhodnotenia.

Ďalším dôležitým faktom je, že `x` a `y` vystupujú vo výraze ako číselné hodnoty a nie ako pamäťové bunky.

Na druhej strane v priraďovacom príkaze, napr. `x = x + 1` prvý výskyt `x` označuje pamäťovú bunku a druhý výskyt x označuje číslo (obsah tejto pamäťovej bunky). Vo výrazoch sa teda kladie dôraz na hodnoty samotné a nie na organizáciu ich uloženia.

Ak sa pri aplikatívnom programovaní ako základný výrazový prostriedok použijú funkcie, vrátane funkcií vyšších rádov (t.j. takých, ktoré operujú nad inými funkciami), hovoríme o *funkcionálnom programovaní*. Ak sú základným výrazovým prostriedkom relácie, opísané pomocou logických predikátov, hovoríme o *logickom programovaní*.

VO FUNKCIONÁLNOM PROGRAMOVANÍ SA VÝPOČET OPISUJE VÝRAZOM

Vo funkcionálnom programovaní sa program chápe ako množina funkcií. Na rozdiel od procedurálneho programovania, ktoré vychádza z modelu výpočtov založeného na von Neumannovej architektúre počítača, opiera sa funkcionálne programovanie o tzv. lambda počet ako jednoduchý model výpočtov. Základné pojmy funkcionál-

neho programovania, ako funkcia, výraz, zloženie výrazov, rekurzívna definícia funkcie sa podrobne vysvetľujú v prvej časti tejto učebnice. Takisto sa tu rozoberajú základné funkcionálne programovacie techniky, ako jednotlivé vzory rekurzívnych definícií funkcie, programovanie filtrov, generátorov, programovanie pomocou funkcií vyšších rádov (funkcionálov).

V LOGICKOM PROGRAMOVANÍ JE VÝPOČET DOKAZOVANÍM DOPYTU

Pri logickom programovaní sa ako programovací jazyk využíva predikátová logika. Základom je interpretácia implikácií ako deklarácií procedúr. Ide o tzv. procedurálnu interpretáciu predikátového počtu prvého rádu. Vytvoriť logický program znamená sformulovať sústavu axióm opisujúcich triedu riešených úloh. Špeciálnu úlohu, ktorú treba vyriešiť, treba sformulovať ako cieľový príkaz. Je to formula predikátového počtu, ktorá sa zapíše v špeciálnom tvare. Výpočet je potom dôkaz, že cieľový príkaz (dopyt) je logický dôsledok množiny axióm, tvoriacej program.

Základné pojmy logického programovania ako klauzula, predikát, term, odvodenie odpovede na zadaný dopyt spolu s programovacími technikami logického programovania sa podrobne vysvetľujú v druhej časti tejto učebnice.

PROGRAMOVANIE OHRANIČENIAMI MÁ DEKLARATÍVNY CHARAKTER

Inou paradigmou deklaratívneho programovania je programovanie ohraničeniami (angl. constraint programming) [23]. V programovaní ohraničeniami opisujeme relácie, pomocou ktorých sa vyjadrujú znalosti o riešenej úlohe. V tomto sa programovanie ohraničeniami podobá logickému programovaniu. Podobný je aj rozdiel oproti procedurálnemu programovaniu. Uvažujme napríklad príkaz (v nejakom procedurálnom jazyku, povedzme v C alebo vo fortrane)

```
celsius = (fahrenheit - 32) * 5 / 9
```

Predpíšeme ním výpočet teploty v stupňoch Celsia, ak poznáme teplotu udanú v stupňoch Fahrenheita. Ak však chceme vypočítať opačný prevod, potrebujeme napísať príkaz

```
fahrenheit = 32 + 9 / 5 * celsius
```

a oba príkazy vnoriť do príkazu vetvenia, ktorý zabezpečí výber jedného z nich podľa potreby. V programovaní ohraničeniami sa považuje výraz

```
celsius = (fahrenheit - 32) * 5 / 9
```

za program, ktorý definuje reláciu medzi stupňami Fahrenheita (`fahrenheit`) a stupňami Celsia (`celsius`). Ak je jedna z týchto veličín daná, druhá sa dá podľa tohto výrazu vypočítať. Netreba dva rôzne príkazy, ako to bolo v prípade procedurálneho programovania. Jeden program (pozostávajúci z jediného výrazu) opisuje riešenie viacerých druhov úloh. V našom príklade okrem dvoch už spomenutých možno týmto programom vyriešiť ešte aj úlohu nájdenia teploty, ktorej číselné vyjadrenia v stupňoch Celsia a Fahrenheita sa rovnajú.

Program v jazyku ohraničení pozostáva z množiny relácií medzi množinou objektov. V našom príklade boli `celsius` a `fahrenheit` objekty. Ohraničenie

```
celsius = (fahrenheit - 32) * 5 / 9
```

je reláciou medzi týmito dvoma objektami. Ak je hodnota jedného z týchto objektov daná, výpočet nájde hodnotu druhého.

V PROGRAMOVANÍ

Výpočet je v podstate spĺňaním ohraničení (angl. constraint satisfaction).

OHRANIČENIAMI JE VÝPOČET SPĹŇANÍM OHRANIČENÍ

Pri výpočte sa použijú pravidlá algebry. Tu je rozdiel oproti logickému programovaniu, kde výpočet znamená pokus o odvodenie dopytu z daných axióm.

Rozdiel oproti procedurálnemu programovaniu je už v ponímaní samotného výrazu

```
celsius = (fahrenheit - 32) * 5 / 9.
```

V procedurálnom programovaní je to priraďovací príkaz. V programovaní ohraničeniami je to výraz, vyjadrujúci vzťah rovnosti. Vzťah rovnosti sa chápe ako invariant, ktorý má byť splnený stále počas vykonávania programu. Pri priraďovacom príkaze sa ľavá a pravá strana zaručene rovnajú iba v okamihu ihneď po jeho vykonaní. Navyše, výskyt premennej na ľavej strane príkazu má iný význam ako výskyt tej istej premennej na pravej strane. To umožňuje napísať v procedurálnom programe napríklad

```
i = i + 1
```

v protiklade s intuíciou, naznačenou symbolom =. V programovaní s ohraničeniami by sa naopak takáto relácia vyhodnotila na nepravdu, pretože pre žiadnu konečnú hodnotu sa táto relácia nedá splniť. Použitím pravidiel algebry by sa tento výsledok našiel aj bez udania hodnoty `i`. Od oboch strán rovnosti možno predsa odpočítať `i`, čím dostaneme rovnosť `0 = 1`, ktorá sa vyhodnotí na nepravdu.

Pre programovanie ohraničeniami vzniklo viacero programovacích jazykov. Vzhľadom na to, že na splnenie ohraničení treba poznať pravidlá platiace pre príslušnú oblasť (napr. pre celé čísla), umožňujú jazyky zvyčajne iba riešenie úloh zo špeciálnej oblasti.

Inými systémami, ktoré sú založené na programovaní ohraničeniami, sú tabuľkové procesory (angl. spreadsheet).

Variantom paradigmy programovania ohraničeniami je logické programovanie ohraničeniami, ktoré kombinuje logické programovanie s možnosťou stanovenia ohraničení. Príkladom je programovací jazyk Prolog III.

VO VIZUÁLNOM PROGRAMOVANÍ SA PROGRAM VYJADRUJE GRAFICKÝMI SYMBOLMI

Vizuálny programovací jazyk rozširuje paletu výrazových prostriedkov na vyjadrenie programu z textu na rôzne triedy grafických symbolov. Motívy prechodu od jednorozmerného (textového) jazyka k dvoj-, prípadne trojrozmerným (vizuálnym) jazykom sú viaceré:

- grafické vyjadrenie zvyšuje prehľadnosť programovej štruktúry,
- grafické vyjadrenie môže znázorniť myšlienky a pojmy z oblasti úlohy a podporiť tak pochopenie programu, usudzovanie o ňom a jeho vzťahu k oblasti úlohy.

Vizuálny programovací jazyk sa sústreďuje zvyčajne na podporu znázornenia

- toku riadenia,
- toku údajov,
- vizuálneho prepisovania (pravidiel prepísania daného obrazca do iného).

Vizuálne programovanie sa používa v kombinácii s niektorou z vyššie uvedených paradigiem.

PRED TÝM NEŽ

Súčasný stav charakterizuje koexistencia viacerých paradigiem. V zmysle definície

NAOZAJ ZAČNEME ...

samotného pojmu paradigma (prevládajúce ustálené názory) tu však ide o určitý protiklad. Pravda, je možné, že prebieha proces zmeny paradigmy a uvedený fakt je výrazom hľadania novej paradigmy. Pre túto hypotézu by svedčilo neustále sa rozširujúce prijatie objektovo-orientovaného programovania, čo môže byť predzvesťou jeho uznania za všeobecnú paradigmu v budúcnosti.

Je však možné, že použitie pojmu paradigma v prípade programovania nie je úplne vhodné, a preto sa nedajú prevziať všetky jej všeobecné vlastnosti a treba aj trvalejšie pripustiť koexistenciu viacerých paradigiem. V prospech tejto hypotézy hovoria vlastnosti jednotlivých paradigiem, ktoré sa často svojimi výhodami dopĺňajú. Dôsledkom toho boli už v osemdesiatych rokoch snahy o kombinovanie jednotlivých paradigiem do jedného programovacieho jazyka. Príkladom sú jazyky kombinujúce funkcionálnu a objektovo-orientovanú paradigmu (CLOS – rozšírenie jazyka lisp o objekty), funkcionálnu a logickú paradigmu (L&O – rozšírenie jazyka prolog o objekty), procedurálnu a objektovo-orientovanú paradigmu (C++) a pod.

Zovšeobecnením týchto snáh je hľadanie mnohoparadigmového jazyka. Ďalší smer zovšeobecnenia je zahrnúť vyjadrenie nie len programov, ale vyjadrovať všeobecnejšie znalosti. Tu sa nadväzuje na vývoj v umelej inteligencii, kde sa hľadajú jazyky na reprezentáciu znalostí. Treba však uviesť, že súčasný stav v tvorbe softvéru charakterizuje množstvo problémov a ani kombinácia uvedených paradigiem, ani hociktorá z nich ich úplne nerieši. Trend približovania počítača k používateľovi sa uplatňuje aj v prípade programátora. Hľadajú sa vhodné špecifikačné jazyky, prípadne programovacie jazyky poskytujúce ešte vyššiu úroveň abstrakcie než doteraz. Uvedené paradigmy možno z tohto hľadiska hodnotiť tak, že deklaratívna paradigma je spôsobom poskytnutia vyššej abstrakcie oproti procedurálnej paradigme. A práve touto paradigmou sa budeme zaoberať v ďalších kapitolách.