INFORMATIKA

Srovnání vývojových diagramů a pseudokódu ve výuce algoritmizace

ONDŘEJ KOŘÍNEK

Pedagogická fakulta, Univerzita Hradec Králové

Algoritmizace, jeden z možných přístupů v programování, patří k nejpoužívanější možnosti, jak začínat výuku programování u začínajících studentů. Lze ji vyučovat několika možnými způsoby. V článku jsou zhodnoceny a porovnány dva způsoby výuky algoritmizace pomocí vývojových diagramů nebo pomocí pseudokódu. Článek se zabývá zavedením proměnné a základními algoritmickými konstrukcemi: sekvence, větvení a cykly.

Studentům na různých typech škol dělají největší potíže předměty, které se zabývají programováním. Výuka předmětu může probíhat více způsoby, protože existuje několik paradigmat programování. S paradigmaty úzce souvisejí i přístupy v programování. Přístup v programování je užší specifikace paradigmatu, kde není např. upřesněn programovací jazyk.

V současné době je asi nejrozšířenějším a nejpoužívanějším paradigmatem i přístupem objektově orientované programování. Přesto i jiné přístupy mají ve výuce svoje místo. Dalšími možnými přístupy ve výuce programování může být komunikativní přístup nebo algoritmizace.

Nejen objektově orientované programování vychází z reálného světa. V algoritmizaci se pracuje s pojmem algoritmus, což je postup, který nás přivede v konečném čase k cíli. S algoritmy se setkáváme v běžném životě např. při přecházení silnice nebo vaření jídla podle receptu. Takže i algoritmizace má v reálném životě zastoupení.

Pro výuku algoritmizace existují algoritmické jazyky, které se dělí na graficky orientované a textově orientované. Mezi graficky orientované programovací jazyky řadíme strukturogramy (Nassi–Schneidermanovy diagramy) nebo vývojové diagramy. Mezi textově orientované algoritmické jazyky řadíme pseudokód, rozhodovací tabulky, slovní popis algoritmu nebo zápis algoritmu v programovacím jazyku [1]. Mezi nejčastěji používané zápisy algoritmu patří vývojové diagramy a pseudokód.

Vývojové diagramy

Vývojové diagramy zobrazují algoritmus v grafické podobě. Jsou dány ISO normou. Každý vývojový diagram má jeden začátek a alespoň jeden konec. Tok výpočtu je znázorněn šipkami. Kromě základních symbolů obsahují i speciální symboly, které se moc nevyužívají. Mezi základní symboly patří symbol pro vstup a výstup dat, symbol zpracování, mezní značka a spojka.

Vývojové diagramy „sváděly programátory k používání nestrukturovaných konstrukcí a dalších programátorských konstrukcí, jež v současné době považujeme za nečisté“ [2].

Nečistost může být v tomto případě dána při složitějším problému velikou nepřehledností, což je v rozporu s moderními metodami o jednoduchém a přehledném kódu [3]. Tyto metody se nevyskytují pouze v programování, ale i při tvorbě statických webových stránek v jazyku HTML (XHTML) a kaskádových stylů.

Podle M. Viriuse [4] již používání vývojových diagramů není aktuální, vzhledem k jejich složitosti. Zpravidla se jim vytýká, že spíše než logickou strukturu programu zdůrazňují druh operací.

Výhody vývojových diagramů jsou v jejich přehlednosti a názornosti, ale algoritmus nesmí být moc složitý, jinak výše uvedené výhody neplatí. Mezi nevýhody vývojových diagramů patří jejich obtížná upravitelnost i neaktuálnost, pokud se zadání algoritmu změní.

Pseudokód

Pseudokód je možností zápisu algoritmu pomocí nezávislosti na programovacím jazyku. Studenti se nemusí zabývat syntaxí příslušného programovacího jazyka. Stačí, aby se naučili pár základních jednoduchých příkazů, které se vyskytují v různých programovacích jazycích s jinou syntaxí. Výhodou používání pseudokódu je, že pro začínající programátory se příkazy v pseudokódu mohou psát i v češtině nebo v jiném jazyce.

Výuce algoritmizace pomocí pseudokódu v jazyce Pascal se zabývá profesorka Milková na Univerzitě Hradec Králové v předmětu Algoritmy a datové struktury. Předmět prošel vývojem, když k němu byla postupně vydána čtyři skripta. Předmět se ustálil v roce 2010 vydáním skript [5]. V průběhu vývoje se ve skriptech zkoušelo začít výklad od datové struktury pole, kdy byly cykly a podmínka vysvětleny pouze zjednodušeně a až potom se vysvětlovala důkladněji práce s jednoduchou proměnnou.

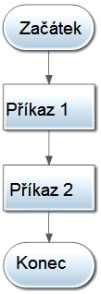
Základní algoritmické konstrukce

V obou výše uvedených možnostech výuky se používají pojmy: jednoduchá proměnná, symbol přiřazení, podmínka, cyklus, pole. Začínajícím studentům dělá největší problémy práce s jednoduchou proměnnou a symbol přiřazení. Symbol přiřazení se v různých programovacích jazycích značí odlišně, obvykle symbolem = nebo :=. Při práci se strukturovanou proměnnou již tolik problémy nemívají. Proměnná označuje objekt nebo místo v paměti. Její zavedení je možné pomocí paměti rozdělené na malé paměťové buňky:

Součet	A
B	Průměr

Sekvence

Sekvence patří mezi základní algoritmické konstrukce, kde jednotlivé příkazy jsou vykonávány postupně za sebou. Rozdíl v zápisu pomocí vývojových diagramů a pseudokódu není příliš patrný (obr. 1). V obou případech je kód jednoduchý a celkem srovnatelný, vývojový diagram je pracnější na vytvoření.

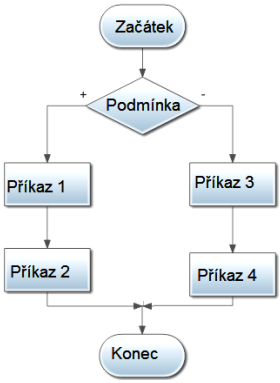



```
Začátek
   Příkaz 1;
   Příkaz 2;
Konec
```

Obr. 1

Větvení

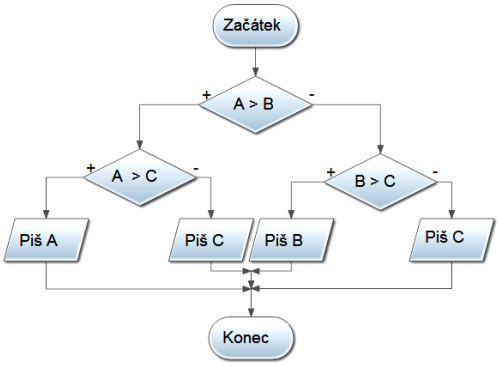
Větvení je další z algoritmických konstrukcí, která se často využívá. Větvení, podmíněný příkaz, může být úplné nebo neúplné. Při zápisu pomocí pseudokódu a vývojového diagramu rozdíl opět příliš patrný není, i když znázornění podmínky pomocí vývojového diagramu je o něco názornější než oproti pseudokódu (obr. 2), ale pouze pro jednodušší příklady.




```
Začátek
  jestliže (podmínka) pak
    začátek
      Příkaz 1;
      Příkaz 2;
    konec
  jinak
    začátek
      Příkaz 3;
      Příkaz 4;
    konec
Konec
```

Obr. 2

Při složitějších úlohách je již znázornění pomocí vývojového diagramu náročnější a může být méně přehledné. Při vnořování podmínek do sebe se již nevýhoda vývojového diagramu projeví, protože zápis pomocí symbolů je již dost nepřehledný na rozdíl od pseudokódu, jehož zápis je přehledný i snadno pochopitelný. Na obr. 3 je příklad na zjištění maxima ze tří čísel.




```
Začátek
  jestliže (A>B) pak
    jestliže (A>C) pak
      Piš A;
    jinak
      Piš C;
  jinak
    jestliže (B>C) pak
      Piš B;
    jinak
      Piš C;
    konec
Konec
```

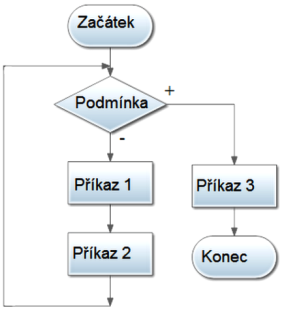
Obr. 3

Cyklus

Cyklus slouží k opakování určité činnosti příkazů. Existují cykly s pevným počtem opakování, s podmínkou na začátku a s podmínkou na konci. Poslední dva uvedené cykly jsou zástupné, tzn., že cyklus s podmínkou na začátku lze vyjádřit pomocí cyklu s podmínkou na konci a naopak.

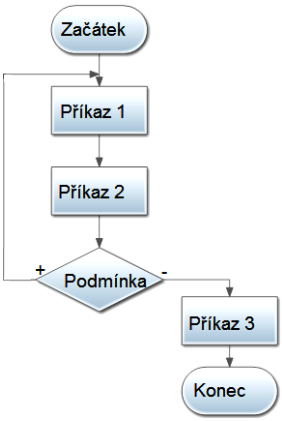
Cykly dělají začínajícím studentům potíže, protože často nevědí, jaký typ mají používat. Cyklus s podmínkou na začátku nemusí proběhnout ani jednou. Cyklus s podmínkou na konci proběhne vždy alespoň jednou.

Cyklus s pevným počtem opakování se v zápisu pomocí vývojových diagramů liší od ostatních, protože pro tento typ cyklus je k dispozici symbol přípravy, který někteří vyučující nepoužívají. Obrázky cyklů s neznámým počtem opakování jsou uvedeny na obr. 4 a 5.




```
Začátek
   dokud (podmínka) opakuj
      začátek
         Příkaz 1;
         Příkaz 2;
      konec
      Příkaz 3;
      konec
Konec
```

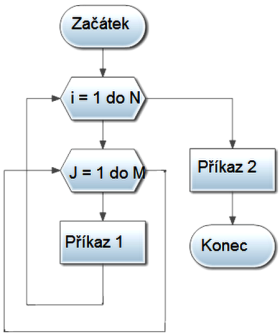
Obr. 4




```
Začátek
   opakuj
      začátek
         Příkaz 1;
         Příkaz 2;
      konec
   dokud (podmínka)
      konec
Konec
```

Obr. 5

Při zápisu cyklu pomocí vývojového diagramu může vést k problémům při vnořování překrývání spojnic. Ukázka špatného vnořování u cyklu s pevným počtem opakování je na obr. 6.




```
Začátek
   pro i od 1 do N opakuj
      pro j od 1 do M opakuj
         Příkaz 1;
   Příkaz 2;
Konec
```

Obr. 6

Cykly zapsané pomocí vývojových diagramů jsou opět náročnější na vytvoření než pomocí pseudokódu. Pomocí pseudokódu je jejich zápis pro studenta, který se učí samostatně pochopitelnější, protože pomocí vývojového diagramu nemusí být jejich smysl na první pohled, hlavně u cyklu s pevným počtem opakování, pochopitelný.

Článek shrnul možnosti zápisu základních konstrukcí algoritmů pomocí vývojových diagramů a pomocí pseudokódu. Při jednodušších příkladech je použití vývojových diagramů názornější než pomocí pseudokódu. Jednodušších příkladů se vyskytuje ale daleko méně, než složitějších. Vývojové diagramy jsou pracnější na zápis a při složitějších úlohách jsou méně názorné, protože řada spojnic může vést k nepřehlednosti, což bylo v článku ukázáno. Křížení spojnic, kterému se při složitých úlohách nevyhneme, vede k rozporu s ISO normou. Pseudokód je bližší k programovacímu jazyku, je přehlednější a lépe se vytvoří, než vývojové diagramy. Úskalím u pseudokódu je špatné odsazování a z toho pramenící horší přehlednost. Stejně jako je nutný při programování přehledný kód, tak u pseudokódu začínající programátoři získají při správném výkladu a jasných požadavcích na dobré odsazování správné návyky, které pak snadno využijí při přechodu na konkrétní programovací jazyk, což vývojové diagramy neumožňují. K lepší vizualizaci pseudokódu existuje pro skripta [5] program, který slouží k dobré vizualizaci a odstraňuje drobné nedostatky na začátku výkladu.

Literatura

[1] *Klimeš, C., Skalka J., Lovászová G., Švec P.*: Informatika pro maturanty a zájemce o studium na vysokých školách. Enigma, Nitra, 2008.

[2] *Pecinovský, R.*: Methodology Architecture First. [online] 2013. Dostupné z: http://vyuka.pecinovsky.cz/prispevky/2013_DIG_Metodika_Architecture_First.pdf [cit. 2013-12-23].

[3] *Fiala, M.*: Vytvořte editor kopenogramů. Diplomová práce, VŠE, Praha, 2012. Dostupné z: https://www.vse.cz/vskp/34803_vytvorte_editor_kopenogramu.

[4] *Virius, M.*: Základy algoritmizace. ČVUT, Praha, 1998.

[5] *Milková, E.*: Algoritmy: základní konstrukce v příkladech a jejich vizualizace. Vyd. 1., Gaudeamus, Hradec Králové, 2010.

Modelování a vizualizace fyzikálních polí v QuickFieldu

JAN RŮŽIČKA

Ústí nad Labem

Druhy fyzikálních polí (Malé repetitorium)

Přírodní děje často popisujeme pomocí polí. Jedná se zejména o skalární a vektorová pole.

Skalární pole je zobrazení, které události (v čase a prostoru) přiřazuje jedinou reálnou veličinu. Jako příklad lze uvést: hustotu, tlak nebo teplotu prostředí. V elektromagnetickém (dále EM) poli je tímto polem např. elektrostatický potenciál, nebo hustota náboje. Obecně je skalární pole funkcí tří prostorových proměnných a času. Vizualizujeme ho pomocí *isočar* nebo *ekvičar*, tedy míst, kde příslušná veličina má konstantní hodnotu.

Vektorové pole je zobrazení, které události (v čase a prostoru) přiřazuje trojici veličin – složek. Příkladem mohou být rychlost proudu kapaliny, napětí a deformace tělesa aj. V elektromagnetickém poli to je např. magnetická indukce nebo intenzita elektrického pole. S vektorovým polem