3 1988

informatyka

Algorytmy kombinatoryczne

Smalltalk-80

Baza danych Holmes

WNICTWO NOT SIGMA

Cena 200 zł

informatyka

3 1988

Nr 3

Miesięcznik Rok XXIII

Marzec 1988

Organ Komitetu Naukowo-Technicznego NOT ds. Informatyki

Materiałów nie zamówionych redakcja nie zwraca

Redakcja: 01-517 Warszawa, ul. Mickiewicza 18 m. 17, tel. 39-14-34.

Zakł. Graf. „Tamka". Zam. 0918-1300/88. Obj. 4,0 ark. druk. Nakład 8650 egz. U-26.

ISSN 0542-9951. INDEKS 36124

Cena egzemplarza 200 zł
Prenumerata roczna 2400 zł

WYDAWNICTWO NACZELNA ORGANIZACJA TECHNICZNA
SIGMA
CZASOPISM I KSIĄŻEK TECHNICZNYCH

00-950 Warszawa
skrytka pocztowa 1004
ul. Biała 4

W NUMERZE

Strona

Algorytmy kombinatoryczne i ich efektywność (1).
Problem najkrótszego drzewa rozpinającego
Maciej M. Sysło 1

Język Smalltalk-80 (1)
Artur Krępski 4

O naturze uczenia się — problemy i kierunki badawcze (2)
Ryszard S. Michalski 8

Implementacja dedukcyjnej bazy danych Holmes (2)
Michał Kirpluk, Piotr Sobolewski 12

Język C. Coraz bliżej normy (1)
Jan Bielecki 16

Struktura systemu operacyjnego PC-DOS (6)
Anna Syfert, Andrzej Raznowiecki 18

Pamięci dyskowe Mazovii 1016 (1)
Andrzej Warda 21

Uwagi o powszechnym kształceniu informatycznym studentów wyższych uczelni
Maciej M. Sysło 23

ZE ŚWIATA 27

Komputerowe nauczanie Ady

TERMINOLOGIA 29

Terminologia języka Smalltalk-80

W NAJBLIŻSZYCH NUMERACH:

- Ralph i Madge Griswold piszą o językach wysokiego poziomu do przetwarzania napisów — COMIT, SNOBOL4 i Icon
- Jerzy Stawicki prezentuje narzędzia tworzenia systemów ekspertowych na mikrokomputerach
- Zbyszko Królikowski opisuje mikrokomputerowe systemy zarządzania bazami danych
- Krzysztof Zieliński i in. charakteryzują architekturę oprogramowania lokalnej sieci komputerowej UMMLAN-2
- Jarosław Deminet relacjonuje przebieg komputerowej wystawy BALTCOM '87

POLSKIE TOWARZYSTWO INFORMATYCZNE

Pierwsza wersja artykułu została przedstawiona podczas Szkoły Jesiennej PTI „Współczesne kierunki rozwoju informatyki" — Rydzyna, 9—13 października 1984 r.

MACIEJ M. SYSŁO
Instytut Informatyki
Uniwersytet Wrocławski

...jednym ze sposobów zwiększenia szybkości komputerów jest obarczenie ich mniejszą liczbą operacji...
Rolph Gomory (1980)

Algorytmy kombinatoryczne i ich efektywność (1)

Problem najkrótszego drzewa rozpinającego

Artykuł jest poświęcony omówieniu wybranych metod rozwiązywania trzech przykładowych problemów kombinatorycznych: problemu najkrótszego drzewa rozpinającego w sieci (w skrócie SST), problemu plecakowego i problemu kolorowania grafu. Najogólniej mówiąc, **problem kombinatoryczny** polega na znalezieniu szczególnego elementu (lub tylko na stwierdzeniu, czy taki element istnieje) w zbiorze, który jest dyskretny, najczęściej skończony i zwykle tworzy pewną strukturę, taką jak graf, sieć, albo jest rodziną permutacji lub partycji zbioru. Konkretne wartości parametrów problemu określają **zadanie** tego problemu.

W zadaniu SST jest dany graf i funkcja wagowa opisana na jego krawędziach. Należy znaleźć drzewo rozpinające grafu o sumarycznie najmniejszej wadze wśród wszystkich drzew rozpinających, których graf n-wierzchołkowy może mieć n^{n-2}. Zbiór możliwych rozwiązań zadania plecakowego tworzą wektory n-elementowe o współrzędnych 0 lub 1, a określić należy wektor spełniający jedno ograniczenie liniowe i maksymalizujący zysk wśród wszystkich wektorów dopuszczalnych. Kolorowanie wierzchołków grafu jest równoważne z określeniem funkcji, która sąsiednim wierzchołkom grafu przyporządkowuje różne liczby naturalne. Problem kolorowania grafu polega na znalezieniu pokolorowania najmniejszą liczbą kolorów.

We wszystkich trzech wypadkach, chociaż przestrzeń możliwych rozwiązań jest skończona, jej przeglądanie, gdy n rośnie, szybko przekracza możliwości jakiegokolwiek komputera, dostępnego teraz i w najbliższej przyszłości (por. tabela 1). Cała nadzieja leży zatem w efektywnych metodach rozwiązywania, które przeglądają jedynie niewielką część przestrzeni rozwiązań. W sensie mocy istniejących metod rozwiązywania, omawiane problemy należą do trzech różnych grup. Sytuację tę zilustrowano w następnych punktach, podając odpowiednie algorytmy. Problem SST może być rozwiązywany algorytmem wielomianowym, czyli takim, który wykonuje liczbę kroków ograniczoną przez wielomian zmiennej n. Nie są znane natomiast żadne algorytmy wielomianowe rozwiązywania pozostałych dwóch problemów. Problem plecakowy jest jednak w pewnym sensie łatwiejszy niż problem kolorowania, gdyż może być rozwiązywany algorytmem wielomianowym względem liczby zmiennych n oraz maksymalnej wartości parametrów. Problem kolorowania należy zaś do klasy najtrudniejszych problemów kombinatorycznych, dla których nie znaleziono dotychczas żadnych efektywnych algorytmów, przynajmniej dostatecznie dobrze przybliżających rozwiązania optymalne.

Tabela 1. Czasy działania algorytmów o różnej liczbie kroków, dla rosnących wartość rozmiaru problemu, na komputerze wykonującym 10^{10} operacji na sekundę (czasy podano w sekundach, jeśli nie zaznaczono inaczej)

Złożoność \ n	10	20	50	100	1000
n					10^{-7}
$n \log_2 n$					10^{-6}
n^2					10^{-4}
n^4	10^{-6}	$2 \cdot 10^{-5}$		10^{-3}	10^2
2^n	10^{-7}	10^{-4}	31^h	$4 \cdot 10^{12}$ lat	
$n!$	$3 \cdot 10^{-4}$	8 lat	10^{47} lat		
n^{n-2}	10^{-3}	$8 \cdot 10^5$ lat	10^{61} lat		
n^n	1	$3 \cdot 10^8$ lat	$3 \cdot 10^{67}$ lat		

Czytelnikom zainteresowanym złożonością obliczeniową algorytmów i problemów warto polecić książki [1, 4]. W tym artykule rezultaty złożonościowe nie występują jawnie, a jedynie z ukrycia sterują tokiem rozważań.

Chociaż wybrane problemy mogą być omawiane niezależnie jeden od drugiego, nadrzędnym celem jest zilustrowanie **zachłannego** podejścia do rozwiązywania zagadnień kombinatorycznych. Dokładne zdefiniowanie znaczenia pojęcia „zachłanności" jest możliwe jedynie w wąskiej klasie zagadnień. Na użytek tego artykułu można przyjąć, że zachłanna metoda rozwiązywania problemu konstruuje rozwiązanie element po elemencie w taki sposób, że kolejno dołączane elementy przyjmują najlepsze z możliwych wartości oraz każde rozwiązanie częściowe jest dopuszczalne, tzn. może być uzupełnione do kompletnego rozwiązania. Nie będzie tu natomiast omawiana zachłanność w sensie lokalnej optymalizacji, polegająca na przechodzeniu od jednego rozwiązania do innego rozwiązania, najlepszego w pewnym otoczeniu tego pierwszego. W wypadku wszystkich trzech problemów idea zachłanności dostarcza bardzo efektywnych algorytmów rozwiązywania, których dokładność maleje jednak z problemu na problem.

Od Czytelnika tego artykułu jest oczekiwana znajomość jedynie podstawowych faktów z zakresu wstępu do informatyki. Książki [6, 9] umożliwiają pogłębienie wiedzy w dziedzinie kombinatoryki i teorii grafów. Ponadto w opracowaniu [7] przystępnie omówiono bogatszą rodzinę problemów kombinatorycznych oraz efektywne algorytmy ich rozwiązywania wraz z elementami złożoności obliczeniowej.

Teoretyczne opisy algorytmów i ich właściwości zawierają najczęściej informacje o pesymistycznym zachowaniu się algorytmów. W praktyce nie mniej ważne są wyniki eksperymentów komputerowych przeprowadzonych z konkretnymi realizacjami algorytmów. Czytelnikowi zainteresowanemu implementacjami algorytmów kombinatorycznych oraz wynikami obliczeń testowych warto polecić zbiory algorytmów w języku Algol 60 [5] i książkę [8] zawierającą programy w Pascalu.

PROBLEM NAJKRÓTSZEGO DRZEWA ROZPINAJĄCEGO

Wyznaczanie najkrótszego drzewa rozpinającego w sieci jest jednym z najpopularniejszych problemów kombinatorycznych, pojawiającym się zarówno w rozważaniach teoretycznych, jak i zastosowaniach praktycznych.

Niech $G_c = (V,E;c)$ będzie siecią, gdzie $G = (V,E)$ jest spójnym grafem symetrycznym, a c jest dowolną funkcją

rzeczywistą określoną na zbiorze krawędzi grafu G. Nie tracąc nic na ogólności, można przyjąć, że funkcja c jest określona na wszystkich nieuporządkowanych parach wierzchołków {u,x}. Jeśli bowiem $\{u,v\} \notin E$, to za c(u,v) przyjmuje się dostatecznie dużą liczbę, która będzie gwarantować, iż krawędź {u,v} nie znajdzie się w żadnym rozwiązaniu. Wystarczy na przykład, by wartość c(u,v) była n razy większa od największej wagi krawędzi. Funkcja c może być rozszerzona na dowolny podzbiór $F = E$ krawędzi grafu. Dla zbioru pustego przyjmuje się $c(0) = 0$, a w pozostałych przypadkach c(F) jest sumaryczną wagą krawędzi należących do F, czyli $c(F) = \sum_{f \in F} c(f)$. Warto przypomnieć że **drzewem rozpinającym** w grafie jest maksymalny podgraf acykliczny. Graf ma drzewo rozpinające wtedy i tylko wtedy, gdy jest spójny.

PROBLEM NAJKRÓTSZEGO DRZEWA ROZPINAJĄCEGO (SST)

Dane:

Spójna sieć symetryczna $G_c = (V,E;c)$.

Cel:

Znaleźć najkrótsze drzewo rozpinające T w sieci G_c.

Jeśli sieć G_c jest zdefiniowana na grafie pełnym, to G zawiera n^{n-2} drzew rozpinających (twierdzenie Cayleya). Zatem na przykład, przestrzeń rozwiązań zadania SST dla sieci pełnej o 10 wierzchołkach składa się z 10^8 elementów. Dzięki jednak szczególnej strukturze rodziny drzew rozpinających i ich poddrzew, problem SST może być rozwiązany w bardzo naturalny sposób. W końcowej części artykułu zostaną opisane ogólne właściwości problemów rozwiązywanych w podobnie prosty sposób.

Najpopularniejsza i jednocześnie jedna z najefektywniejszych metod rozwiązywania problemu SST jest realizacją bardzo prostej idei podanej przez J. B. Kruskala, która poleca budować rozwiązanie krawędź po krawędzi, wybierając kolejno najkrótszą z krawędzi, których dołączenie do rozwiązania częściowego nie powoduje powstania w nim cyklu. Inaczej mówiąc, w metodzie tej w każdym kroku dokonuje się najlepszego wyboru dbając jednocześnie o to, by każde rozwiązanie częściowe było dopuszczalne, tzn. dało się uzupełnić do rozwiązania kompletnego. Algorytm jest więc jak najbardziej zachłanny.

ZACHŁANNY ALGORYTM KRUSKALA

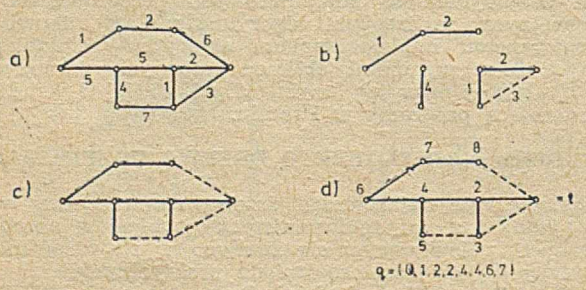
Dane:

Spójna sieć symetryczna $G_c = (V,E;c)$. Niech $n = |V|$.

Wynik:

Najkrótsze drzewo rozpinające T w sieci G_c.

```
begin
T ← 0; F ← E;
  while |T| < (n—1) and F ≠ 0 do begin
        e ← najkrótsza krawędź w E; F ← F—{e};
        if T ∪ {e} nie zawiera cyklu then T ← T ∪ {e}
  end
end
```



Rys. 1. Ilustracja działania algorytmu Kruskala

Działanie powyższego algorytmu zilustrowano na przykładzie sieci z rys. 1a. Na rysunku 1b przedstawiono rozwiązanie częściowe po sześciu krokach algorytmu, a rozwiązanie kompletne pokazano na rys. 1c. Krawędzie należące do rozwiązania są oznaczone linią ciągłą, a pozostałe — przerywaną. Rozwiązanie częściowe może składać się z więcej niż jednej części spójnej.

Algorytm zachłanny może być zmodyfikowany i ulepszony na wiele różnych sposobów. W powyższym opisie uwzględniono już, że jeśli $|T|$ osiąga n—1, to obliczenia mogą być przerwane, gdyż T jest już drzewem rozpinającym w sieci G_c. A więc żadna krawędź spoza T nie może być dołączona do T. Przy użyciu wyspecjalizowanych struktur danych oraz algorytmów posługiwania się nimi, algorytm zachłanny może być zrealizowany w czasie O(m logm), gdzie m jest liczbą krawędzi grafu (por. [1, 2, 8]).

Wersja algorytmu zachłannego, podana przez R. C. Prima i E. W. Dijkstrę, różni się od algorytmu zachłannego tym, że każde rozwiązanie częściowe jest spójne, a krawędzie są tylko częściowo porządkowane podczas działania algorytmu. Algorytm Prima-Dijkstry konstruuje najkrótsze drzewo rozpinające T od dowolnie wybranego wierzchołka s grafu G (s nazywa się **korzeniem** drzewa T) i w każdym z n—1 kroków dołącza do rozwiązania częściowego najkrótszą krawędź spośród krawędzi o jednym końcu w rozwiązaniu, a drugim spoza niego. W przedstawionym niżej opisie algorytmu, rozwiązanie T jest pamiętane w postaci **listy poprzedników**, tj. dla każdego wierzchołka v (z wyjątkiem s) jest określony numer wierzchołka bezpośrednio poprzedzającego wierzchołek v na drodze z s do v. Niech p_i będzie odległością (w sensie funkcji c) wierzchołka i od bieżącego rozwiązania częściowego, a q_i — wierzchołkiem tego rozwiązania, który realizuje tę odległość, tj. $c(q_i,i) = p_i$.

ALGORYTM PRIMA-DIJKSTRY

Dane:

Spójna sieć symetryczna $G_c = (V,E;c)$ i korzeń rozwiązania s.

Wyniki:

Najkrótsze drzewo rozpinające T w sieci G_c reprezentowane za pomocą listy poprzedników $(q_1,q_2,\ldots,q_n)$. Wagą drzewa T jest cT.

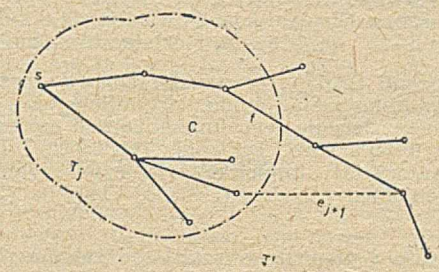
```
Krok 1 [Zapoczątkowanie obliczeń]
p_s ← 0; q_s ← 0; T ← 0; cT ← 0; r ← s;
for każdego v ∈ W = V—{s} do
  begin p_v ← c_sv; q_v ← s and;
Krok 2 [Iteracja]
for i = 1,2,...,n—1 do
begin
  wyznaczyć p_w = min {p_v : v ∈ W};
  T ← T ∪ {q_w, w}; cT ← cT + p_w; W ← W—w;
  for każdego u ∈ W do
    if p_u > c_wu then begin p_u ← c_wu; q_u ← w end
end
```

Działanie algorytmu Prima-Dijkstry zilustrowano na przykładzie sieci z rys. 1a. Najkrótsze drzewo rozpinające wygenerowane przez ten algorytm przedstawiono na rys. 1d, na którym numery wierzchołków oznaczają kolejność, w jakiej wierzchołki są dołączane do rozwiązania.

Dowód poprawności algorytmu Prima-Dijkstry jest rozumowaniem typowym dla dowodów poprawności działania algorytmów zachłannych. Przypuśćmy, że drzewo T wyznaczone przez ten algorytm dla sieci G_c nie jest najkrótszym drzewem rozpinającym; niech T' będzie najkrótszym takim drzewem w G_c. Nie tracąc nic na ogólności można przyjąć, że oba te drzewa mają korzenie w wierzchołku, z którego algorytm rozpoczął tworzyć drzewo T. Można też przyjąć, że krawędzie drzewa T są ponumerowane zgodnie z kolejnością, w jakiej były dołączane do T, a więc $T = \{e_1, e_2, \ldots, e_{n-1}\}$. Niech $T_i = \{e_1, e_2, \ldots, e_i\}$, czyli $T = T_{n-1}$. Ponieważ $T' \neq T$, więc istnieje j takie, że $T_j \subset T'$, i $T_{j+1} \not\subset T'$, czyli $e_{j+1} \notin T'$. Z właściwości drzew rozpinających wynika, że $T' \cup \{e_{j+1}\}$ zawiera dokładnie jeden cykl C.

Ponieważ część krawędzi cyklu C należy do T_j, a część nie należy, więc oprócz krawędzi e_{j+1}, cykl C zawiera jeszcze jedną krawędź, f, której jeden koniec należy, a drugi nie należy do T_j (rys. 2). Oczywiście $f \in T'$. Z właściwości algorytmu wynika, że $c(f) \geqslant c(e_{j+1})$, gdyż krawędź e_{j+1} została dołączona do T_j jako najkrótsza krawędź wychodząca z T_j. Można więc określić nowe drzewo rozpinające $T'' = T' \cup \{e_{j+1}\} - \{f\}$ w sieci G_c, dla którego $c(T'') \leqslant c(T')$. Zatem, jeśli $c(f) > c(e_{j+1})$, to dochodzi się do sprzeczności, gdyż T'' byłoby drzewem krótszym od najkrótszego drzewa T'. Jeśli natomiast $c(f) = c(e_{j+1})$, to $c(T'') = c(T')$ i $T_{j+1} \subset T''$. W tym drugim wypadku powtarzając powyższe rozumowanie, dochodzi się albo do drzewa krótszego od T', albo do drzewa $\tilde{T}$, które spełnia $\tilde{T} = T_{n-1} = T$, $c(\tilde{T}) = c(T')$, co także jest w sprzeczności z założeniem, że T nie jest najkrótszym drzewem.

Liczba elementarnych operacji wykonywanych przez algorytm Prima-Dijkstry jest równa $0(n^2)$. W kroku 1 wykonuje się bowiem 0(n) operacji, a w i-tej iteracji kroku 2 jest wykonywanych 0(n—i) operacji związanych z wyzna-



Rys. 2. Drzewo zawierające cykl w algorytmie Prima-Dijkstry

czaniem minimum oraz 0(n—i) operacji uaktualniających wartości p_n, q_n. Najszybsze znane algorytmy wyznaczania najkrótszych drzew rozpinających w sieciach przedstawiono w [3].

PORÓWNANIE EFEKTYWNOŚCI ALGORYTMÓW ROZWIĄZYWANIA PROBLEMU SST

Dla sieci rzedkich (takimi są na przykład sieci planarne), dla których m jest równe 0(n), algorytm Kruskala ma złożoność 0(n log n), natomiast algorytm Prima-Dijkstry działa w czasie $0(n^2)$. Dla sieci pełnych porównanie efektywności wypada na korzyść algorytmu Prima-Dijkstry. Te porównania, oparte jedynie na asymptotycznym zachowaniu się algorytmów, potwierdzają obliczenia testowe wykonane dla losowych sieci (tabela 2).

Tabela 2. Przykładowe czasy obliczeń algorytmami Kruskala i Prima-Dijkstry (komputer Amdahl 470 V/8, por. [8], S. 265)

Liczba krawędzi m	Czas działania (ms)	
	Algorytm Kruskala	Algorytm Prima-Dijkstry
Sieci gęste z 80 wierzchołkami		
790	63	52
1580	96	51
3160	186	50
Sieci rzadkie ze 100 wierzchołkami		
200	44	78
300	66	80

Niemal identyczne rozumowanie można przeprowadzić dla problemu wyznaczania najdłuższego drzewa rozpinającego w sieci. Metodą zachłanną można rozwiązywać wiele innych problemów kombinatorycznych, o czym będzie jeszcze mowa w ostatniej części artykułu.

LITERATURA

[1] Aho A. V., Hopcroft J. E., Ullman J. D.: Projektowanie i analiza algorytmów komputerowych. PWN, Warszawa, 1983

[2] Banachowski L., Kreczmar A.: Elementy analizy algorytmów. WNT, Warszawa, 1981

[3] Gabow H. N., Galil Z., Spencer T., Tarjan R. E.: Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. Combinatorica, Vol. 6, pp. 109—122, 1986

[4] Garey M. R., Johnson D. S.: Computers and Intractability. Freeman, San Francisco, 1979

[5] Kucharczyk J., Sysło M. M.: Algorytmy optymalizacji w języku Algol 60. PWN, Warszawa, 1975

[6] Lipski W.: Kombinatoryka dla programistów. WNT, Warszawa, 1982

[7] Sysło M. M.: Optymalizacja kombinatoryczna. Instytut Informatyki, Uniwersytet Wrocławski, 1981 (skrócona wersja — rozdział III w książce: Ekonomiczne zastosowania optymalizacji dyskretnej, T. Kasprzak (red.), PWE, Warszawa, 1984)

[8] Sysło M. M., Deo N., Kowalik J. S.: Discrete optimization algorithms with Pascal programs. Prentice-Hall, Englewood Cliffs (NJ), 1983

[9] Wilson R. J.: Wprowadzenie do teorii grafów. PWN, Warszawa, 1985.

ARTUR KRĘPSKI
Instytut Podstaw Informatyki
Polska Akademia Nauk
Warszawa

Język Smalltalk-80 (I)

Niniejszy artykuł jest pierwszy z cyklu trzech poświęconych systemowi programowania Smalltalk-80. Kolejno będą omówione: język Smalltalk-80, system Smalltalk/V oraz grafika w systemie Smalltalk/V. Ich celem jest zaprezentowanie zasadniczych cech systemu Smalltalk-80 i w ten sposób podjęcie próby jej spopularyzowania w Polsce. Smalltalk/V jest realizacją systemu Smalltalk-80 na mikrokomputerze IBM PC i wszystkie uściślenia (języka i systemu) są wzięte z tej konkretnej realizacji. Terminologię Smalltalka wyjaśniono w oddzielnym artykule (str. 29).

Druk artykułów nt. systemu Smalltalk/V przewidujemy w przyszłym roku.

Autor niniejszego cyklu zainteresował się systemem Smalltalk-80 „tylko jako fascynującym systemem programowania, ale wyraża głębokie przekonanie, że system ten powinien być co najmniej inspiracją (a właściwie, wysokiej jakości narzędziem) dla szerokiego grona informatyków — w dziedzinach takich, jak systemy konwersacyjne, projektowanie wspomagane komputerem, systemy dydaktyczne wspomagane komputerowo, sztuczna inteligencja, grafika, symulacja.

System Smalltalk-80 jest interakcyjnym, zintegrowanym środowiskiem programowania, jednostanowiskowym z możliwością wykonywania wielu zadań równolegle, wspomagającym programowanie obiektowe (ang. object-oriented programming) w języku Smalltalk-80 z wykorzystaniem grafiki komputerowej. Unifikując sprzęt i oprogramowanie pozwala on przekształcić standardowy zestaw komputerowy w elastyczny, osobisty warsztat programowania, który umożliwia jednoczesne projektowanie i implementowanie programów aplikacyjnych.

Niniejszy artykuł, ze względów redakcyjnych, podzielono na dwie części; w pierwszej przedstawiono intuicyjnie elementarne pojęcia języka oraz dokonano przeglądu podstawowych wyrażeń (stałe, zmienne, komunikaty), w drugiej — dokończono przegląd konstrukcji języka (bloki, metody) oraz zaprezentowano przykład klasy. Całość artykułu poprzedza krótki wstęp — geneza i historia systemu Smalltalk-80, a kończy — podsumowanie i gramatyka języka.

Dr ARTUR ANDRZEJ KRĘPSKI ukończył w 1974 roku studia na Wydziale Matematyki-Mechaniki Uniwersytetu Warszawskiego. W 1979 r. uzyskał stopień doktora nauk technicznych (specjalność — informatyka). W latach 1984—1985 był stypendystą Fundacji im. A. von Humboldta (RFN). Jest adiunktem w Instytucie Podstaw Informatyki PAN, pracuje na stanowisku kierownika Zespołu Inżynierii Oprogramowania. Zainteresowania naukowe dotyczą inżynierii oprogramowania, szczególnie języków programowania i ich kompilatorów, metodologii i systemów programowania.

System Smalltalk-80 jest wynikiem intensywnych prac prowadzonych bez większego rozgłosu od kilkunastu lat w ośrodku firmy Xerox (Palo Alto Research Center), w USA. Język opracowano iterując pięć razy cykl projekt-implementacja-próbna eksploatacja. Powstały więc kolejno systemy Samlltalk-72, -74, -76, -78 i ostatecznie Smalltalk-80.

Decydującymi etapami w popularyzowaniu systemu były: decyzja opublikowania cyklu artykułów w czasopiśmie BYTE [1] oraz zlecenie jego implementacji czterem firmom: Apple, Hewlett-Packard, Tektronix (MC68000) i DEC (VAX-11/780). Autorskie implementacje były zrealizowane na komputerach firmy Xerox (Dorado, Dolphin (1100 SIP) i Alto).

Nie wszystkie planowane realizacje zakończyły się sukcesem; najbardziej znana jest implementacja na komputer Tektronix 4404 oferowana jako standardowy system (Artificial Intelligence System) na mikroprocesorze MC68010 z zegarem 10 MHz. Wymaga ona 1 MB pamięci operacyjnej (rozszerzalnej do 2 MB), dysku stałego 20 MB, stacji dyskietek 5-calowych, monitora monochromatycznego o rozdzielczości 640×480 punktów, klawiatury i myszy. Cena razem ze sprzętem — ok. 15 tys. dolarów (1985 r.) — jest wysoka, choć znacznie niższa od ceny wersji autorskiej na Dolphin (60 tys. dolarów).

W latach 1983—1984 autorzy systemu opublikowali serię książek [2, 3, 4]. Niestety, pomimo ich obietnic, tak przygotowana dokumentacja, choć obszerna, jest niepełna (opisuje 30—50% systemu); zawiera też błędy i sprzeczne informacje.

Jednocześnie autorzy zaczęli rozpowszechniać (za wysokimi opłatami) licencjonowaną taśmę dystrybucyjną z systemem przygotowanym do implementacji przez abstrakcyjną maszynę stosową. W wyniku współpracy ze wspomnianymi czterema firmami wystarczająco dobrze przetestowano dokumentację oraz oprogramowanie systemu, usuwając podstawowe błędy.

Sprzedaż systemu zbudowanego na podstawie autorskiej implementacji wymaga drogiej licencji z ośrodka Xerox w Palo Alto, zaś system jest objęty embargiem do krajów Europy Wschodniej.

Zbyt wysokie ceny systemu Smalltalk-80 na sprzęcie profesjonalnym wywołały próby jego realizacji na mikrokomputerach (IBM PC/XT, IBM PC/AT, Apple II). Firma Digitalk opracowała dla IBM PC/XT oraz AT najpierw system Methods, a potem Smalltalk/V [7]. Pierwszy jest bardzo ograniczony w porównaniu z oryginałem (bez grafiki — z monitorem alfanumerycznym i symulacją myszy na klawiaturze, okrojono też znacznie zestaw klas systemowych); cena tego systemu szybko spada — z 250 dolarów (1985 r.) do 80 dolarów obecnie. Drugi (cena ok. 100 dolarów) — zawiera dużą część klas graficznych, co pozwala dość dobrze zrealizować interakcyjną współpracę z użytkownikiem; dostępna jest także ok. połowa oryginalnych klas systemowych. System ten uzyskał bardzo pochlebne opinie użytkowników. Wymaga on komputera IBM PC/AT lub XT z pamięcią operacyjną co najmniej 512 KB, adaptera i monitora graficznego dużej rozdzielczości (np. Hercules), klawiatury, myszy oraz stałego dysku 20 MB.

Dla IBM PC/AT dostępna jest również pełna realizacja systemu Smalltalk-80 (firmy Softsmarts, cena ok. 1000 dolarów).

Obecnie prace nad systemem Smalltalk-80 zmierzają, z jednej strony, do ulepszenia samego języka (opracowanie efektywniejszych algorytmów jego implementacji, wprowa-

dzenie pojęcia typu), a z drugiej — do realizacji sprzętowej systemu. Opracowano dwa mikroprocesory o architekturze wspomagającej efektywną realizację systemu Smalltalk-80 na poziomie kodu pośredniego maszyny stosowej: SOAR (ang. Smalltalk On A RISC) na Uniwersytecie w Berkeley (USA) w latach 1983—1985, oraz SWORD 32 na Uniwersytecie Tokijskim (Japonia) w 1984 r. Pierwszy osiąga szybkość interpretacji ponad 1 mln instrukcji języka pośredniego na sekundę, drugi — ok. 1,4 mln.

W Europie nie są znane żadne ośrodki prowadzące prace nad systemem Smalltalk-80, choć wiele uniwersytetów zainstalowało pełną wersję systemu.

W Polsce, w latach 1983—1984, prowadzono pewne prace wstępne (m.in. próba realizacji systemu na komputerze Mera 400) w zespole dr. inż. Z. Kulpy w Instytucie Biocybernetyki i Inżynierii Biomedycznej. Nie osiągnięto zamierzonego celu (m.in. z powodu trudności sprzętowych) i prace przerwano.

Od 1985 r., w Zespole Inżynierii Oprogramowania Instytutu Podstaw Informatyki PAN, w ramach badań nad systemami programowania, są prowadzone prace dotyczące systemu Smalltalk-80 — m.in. w 1986 r. odbył się cykl seminariów badawczych na ten temat (ich podsumowanie w postaci raportu [5] jest podstawą niniejszego artykułu).

PROGRAMOWANIE OBIEKTOWE

Pojęcie programowania obiektowego nie jest do tej pory ściśle zdefiniowane i właściwie popularność zyskuje dopiero teraz, wraz z rozpowszechnianiem się języka Smalltalk-80.

Podstawą tej metody programowania jest pojęcie klasy, której reprezentantami (wcieleniami) są obiekty; w tym zakresie jest to zapożyczenie z dobrze znanego języka Simula 67 [6].

Obiekt

Obiekt można uważać za zestaw danych plus zestaw operacji; dane obiektu są dostępne poza nim jedynie pośrednio — przez jego operacje. Obiekt spełnia więc podstawowe warunki abstrakcyjnego typu danych.

Komunikacja między obiektami odbywa się wyłącznie w ten sposób, że do wskazanego obiektu wysyła się żądanie wykonania jednej z jego operacji (być może, zadając pewne argumenty). Żądanie to zapisuje się jako instrukcję **wysłania komunikatu** wskazując obiekt — odbiorcę komunikatu oraz komunikat — oznaczenie operacji (selektor) z zadanymi argumentami. Obiekt, w którym dotąd działano, nazywa się nadawcą komunikatu.

Wykonanie instrukcji wysłania komunikatu polega na odnalezieniu pasującej do niego operacji (tj. operacji oznaczonej selektorem zadanym w komunikacie) i jej wykonaniu; obliczona wartość jest komunikowana nadawcy i stanowi wynik wysłania komunikatu.

Stąd właśnie bierze się nazwa **programowanie obiektowe;** obiekt nie jest przedmiotem obliczeń (manipulacji) dla jakiejś siły zewnętrznej względem niego, lecz sam wyznacza przebieg przetwarzania stając się podmiotem (jako odbiorca komunikatu) interpretującym komunikaty. Ten sam komunikat wysłany do różnych obiektów (odbiorców) może mieć różną semantykę, np. żądanie dodania liczby rzeczywistej wysłane do odbiorcy — liczby rzeczywistej, oznaczać będzie dodanie liczb rzeczywistych, natomiast wysłane do tablicy może oznaczać dodanie argumentu do każdego elementu tablicy. W ten naturalny sposób osiągamy przeciążenie (ang. overloading) wszystkich operatorów języka (selektorów metod).

Klasa

Praktyka wskazuje, że wiele obiektów różni się od siebie jedynie wielkością lub(i) wartością danych, natomiast operacje na nich wykonywane są takie same. Dlatego wygodnie jest mieć pewien wzorzec (opis) tak podobnych obiektów. Opis ten nazywamy **klasą** obiektów (podobnych), każdy z nich — **reprezentantem** klasy, (pod)programy poszczególnych operacji — **metodami**, zaś dane opisujemy jako wartości zmiennych reprezentatywnych (być może, indeksowanych).

Sama metoda potrzebuje zwykle zmiennych pomocniczych, które mają znaczenie tylko w jej podprogramie — nazywamy je zmiennymi roboczymi metody.

Czasami pewne dane (np. stałe) są dostępne wszystkim reprezentantom danej klasy — opisujemy je za pomocą zmiennych klasowych.

Wszystkie metody w danej klasie tworzą protokół komunikatów, które reprezentanci tej klasy umieją interpretować; można więc nazwać go sprzężeniem — specyfikacją zachowania się — reprezentantów tej klasy.

Programowanie obiektowe polega na tworzeniu klas, których metody opisują sposób interpretowania komunikatów wysyłanych do reprezentantów tych klas.

Hierarchia klas

Same klasy też mogą być do siebie podobne, w tym sensie, że część ich metod jest taka sama, a część jest różna. W języku Smalltalk-80 dopuszczono możliwość budowania nowych klas, wychodząc od klasy już zbudowanej, w ten sposób, że możliwe jest dodawanie do niej nowych metod lub modyfikowanie już dostępnych. Mówi się wówczas, że nowe klasy są podklasami danej klasy (ją samą nazywamy ich nadklasą), metody nie zmienione nazywamy dziedziczonymi, a zmodyfikowane — przedefiniowanymi (zmienne reprezentatywne nadklasy są dziedziczone w całości, tak że w podklasach można jedynie dodawać nowe). Powstaje w ten sposób drzewiasta **hierarchia klas**, którą można interpretować dwojako: patrząc z góry w dół — podklasy danej klasy stanowią jej specjalizację do konkretnych celów, zaś z dołu do góry — nadklasa jest uogólnieniem (abstrakcją) swoich podklas.

Aby obraz był pełny należy jeszcze dodać, że w języku Smalltalk-80 pojęcie obiektu uczyniono totalnym — każdy element języka jest obiektem, w szczególności również sama klasa. Klasę, której reprezentant jest klasą, nazywamy jej metaklasą. Metody metaklasy (np. powoływanie i inicjowanie reprezentantów klasy) nazywamy metodami klasowymi — w odróżnieniu od dotychczas opisywanych, które nazywamy metodami reprezentatywnymi.

Ostatecznie można więc powiedzieć, że **programowanie obiektowe** polega na tworzeniu klas powiązanych w hierarchię za pomocą mechanizmu dziedziczenia i specjalizowania; elementarnymi jednostkami tego procesu są metody, które przez swoje instrukcje opisują proces obliczeniowy, polegający zasadniczo na wysyłaniu komunikatów do obiektów.

SMALLTALK JAKO JĘZYK KODOWANIA METOD

W tym punkcie dokonano przeglądu podstawowych wyrażeń języka podzielonych na stałe, zmienne i komunikaty.

Stałe

Stałe służą oznaczaniu reprezentantów pewnych wyróżnionych, najczęściej spotykanych, klas; są to liczby (reprezentanci klas Float, Fraction, Integer), znaki (Character), napisy (String), symbole (Symbol) oraz tablice (Array). Składnia stałych jednoznacznie określa klasę, której reprezentanta one oznaczają.

Dodatkowo trzy stałe są oznaczone identyfikatorami zastrzeżonymi; oto one:

nil — oznacza (jedynego) reprezentanta klasy Undefined Object; służy do inicjowania zmiennych w wypadku, gdy w programie nie są im nadane wartości;

false — oznacza reprezentanta klasy Boolean reprezentującego wartość logiczną „fałsz";

true — oznacza reprezentanta klasy Boolean reprezentującego wartość logiczną „prawda".

Obiekty oznaczone stałymi są przechowywane w systemie lub tworzy się je na etapie translacji (często są one kodowane nieco inaczej — oszczędniej — niż reprezentanci pozostałych klas).

Wszystkie **liczby** są obiektami klasy Number i zapisuje się je podobnie jak w tradycyjnych językach programowania (z kropką dziesiętną i czynnikiem skalującym poprzedzonym literą **e**). Sama postać zapisu liczby precyzuje jej podklasę — Float, Fraction lub Integer.

Jeżeli liczba zawiera kropkę dziesiętną, to należy ona do klasy Float (liczby zmiennopozycyjne). Jeśli nie ma kropki dziesiętnej i ma wykładnik ujemny, to należy do klasy Fraction (liczby wymierne reprezentowane dokładnie jako pary liczb całkowitych). We wszystkich pozostałych wypadkach liczba należy do klasy Integer (liczby całkowite).

W zapisie liczby może wystąpić litera **r** (ang. radix) — wówczas dziesiętne cyfry poprzedzające literę **r** oznaczają podstawę systemu liczbowego; do oznaczania cyfr liczby, gdy są one większe od 9, można użyć dużych liter — **A, B,** itd. Wykładnik, do którego należy podnieść podstawę systemu liczbowego, aby wyliczyć czynnik skalujący, jest zawsze w notacji dziesiętnej.

Przykłady liczb: —100, 1, 0, 16rFF (liczby całkowite), 5e—1, —1e—5 (liczby wymierne), 3.1416, 8r—1.077e10 (liczby zmiennopozycyjne).

W **stałych znakowych**, znak (reprezentant klasy Character) jest poprzedzony znakiem $, np. $a, $A, $s.

Napis jest ciągiem znaków ujętych w znaki apostrofu (obrazem apostrofu w napisie jest ciąg dwóch apostrofów) np. 'dwa apostrofy '', to obraz apostrofu w napisie', 'to jest napis bez apostrofu'.

Symbol reprezentuje ciąg znaków (poprzedzony znakiem #), przy czym wszystkie takie same ciągi znaków są reprezentowane przez jeden i ten sam symbol np. #at:put:, #+, #==.

Stała tablicowa jest ciągiem stałych (wśród których może być znowu tablica) ujętych w okrągłe nawisy i poprzedzonych znakiem #; składowe tablicy będące tablicami i symbolami nie są poprzedzane znakiem #, np. #('raz' 'dwa' 'to dwa napisy'), #(raz dwa 'to dwa symbole'), #(1 $2 '3' cztery (5) 'to liczba, stała znakowa, napis, symbol, tablica'). Poszczególne elementy tablicy są numerowane indeksami całkowitoliczbowymi począwszy od 1; odwołania do nich mają wyłącznie postać komunikatów kluczowych, np. at: 1, at 3: put: $$.

Zmienne

Zmienną można traktować jak pojemnik, którego zawartość w dowolnej chwili wskazuje jakiś obiekt. Zmienne oznacza się identyfikatorami.

Zmianie wartości zmiennej (tj. wymianie obiektu, na który ona wskazuje) służy **przypisanie**; w wyniku wykonania przypisania postaci

zmienna1 := zmienna2 := ... := wyrażenie (np. i := 1)

na obiekt będący wartością wyrażenia będą wskazywały zmienne zmienna1, zmienna2, itd. (a wartością całego przypisania — bo ono samo także jest wyrażeniem — będzie również ten obiekt).

Można wyróżnić trzy następujące grupy zmiennych:

— zmienne reprezentatywne,

— zmienne lokalne,

— zmienne wspólne.

Zmienne reprezentatywne i lokalne oznacza się identyfikatorami zaczynającymi się małą literą, natomiast zmienne wspólne — dużą. Zwyczajowo, kolejne wyrazy składające się na jeden identyfikator zaczynają się dużą literą.

Zmienne reprezentatywne określają stan obiektu; można je uważać za zmienne prywatne obiektu (traktowanego jako abstrakcyjny typ danych), bo są dostępne bezpośrednio jedynie w metodach klasy, której ten obiekt jest reprezentantem. Zmienne te istnieją tak długo, jak długo istnieje obiekt.

Zmienne reprezentatywne oznacza się albo identyfikatorem, albo indeksem całkowitoliczbowym — nazywamy je wtedy **zmiennymi indeksowymi** (indeks ma zakres od 1 do wartości ustalonej w momencie tworzenia reprezentanta danej klasy).

W wypadku zmiennych nieindeksowanych, każdy identyfikator oznacza pojedynczą zmienną, ich liczba jest taka sama dla wszystkich reprezentantów danej klasy i ustalona w momencie definiowania klasy.

Odwołania do zmiennych indeksowanych mają postać wyłącznie komunikatów (kluczowych — p. punkt następny), np. at: indeks, at: indeks put: wartość itp., poszczególni reprezentanci tej samej klasy mogą mieć różną liczbę zmiennych indeksowanych, a ich występowanie sygnalizuje się w momencie definiowania klasy.

Zmienne lokalne są powoływane na czas aktywności metody i giną bezpowrotnie po jej wykonaniu. Na zmienne te składają się parametry i zmienne robocze metody. Parametry są inicjowane (według kolejności wystąpienia) wartościami odpowiadających im argumentów komunikatu uaktywniającego tę metodę i nie można im przypisywać nowej wartości. Zmienne robocze są inicjowane stałą **nil.**

Parametry bloku (p. część II) mają znaczenie podobne do parametrów metody i na czas wykonywania tego bloku stanowią rozszerzenie zbioru zmiennych lokalnych zawierającej go metody.

Zmienne wspólne są dostępne w metodach wielu obiektów; muszą być jawnie powoływane i usuwane. Zmienne wspólne grupuje się w tzw. pule — są to słowniki przechowujące pary, które wiążą identyfikator zmiennej z jej wartością. Poszczególne pule udostępnia się klasom (i ich reprezentantom) w procesie definiowania klas.

Dwa szczególne rodzaje zmiennych wspólnych przechowuje się w systemie — zmienne klasowe i zmienne globalne.

Zmienne klasowe (zbierane w pule automatycznie przez system w procesie definiowania klasy) są dostępne swojej klasie, jej podklasom i ich wszystkim reprezentantom.

Zmienne globalne, zebrane są w pulę o zastrzeżonej nazwie **Smalltalk,** są dostępne wszystkim obiektom w systemie. Część tych zmiennych jest zdefiniowana przez realizację, część jest automatycznie wprowadzana przez system (nazwy wszystkich klas i pul), inne może wprowadzać sam użytkownik.

Poza wymienionymi trzema rodzajami zmiennych, wprowadzono **zmienne o identyfikatorach zastrzeżonych** i ściśle zdefiniowanych wartościach (zmiennym tym nie można przypisywać wartości). Są to zmienne **self** oraz **super** (mogą wystąpić jeszcze inne w konkretnych realizacjach). Obydwie wskazują na odbiorcę komunikatu, który uaktywnił metodę, ale ponadto mają dodatkowe znaczenie, jeśli są odbiorcami komunikatów (p. część II). Takie symboliczne oznaczenie odbiorcy komunikatu pozwala go traktować jako dodatkowy — wyróżniony argument metody; umożliwia się w ten sposób korzystanie z niego w wyrażeniach metody (np. przy zapisie algorytmów rekurencyjnych).

Komunikaty

Wyrażenia są zapisem sposobu wyznaczania pojedynczego obiektu — wartości wyrażenia. Ciąg wyrażeń składa się z wyrażeń rozdzielanych znakiem kropki. Ciąg wyrażeń wartościuje się wartościując kolejno jedno po drugim wyrażenia składowe, a wartość ostatniego wyrażenia jest zarazem wartością całego ciągu.

Składniowo, wyrażenie jest zbudowane z wyrażeń prostych (stałych, zmiennych, bloków i wyrażeń ujętych w nawiasy okrągłe) połączonych selektorami; dodatkowo takie wyrażenia mogą występować w przypisaniach. Selektory (odpowiedniki operatorów w zwykłych językach programowania) są albo identyfikatorami **(selektory unarne),** albo jedno- i dwuznakowymi symbolami **(selektory binarne),** albo wreszcie konkatenacją kluczy — identyfikatorów zakończonych znakiem dwukropka **(selektory kluczowe).** Przykłady selektorów:

- unarne — sin, sqrt; negated;
- binarne — <=, +, /=;
- kluczowe — at:, at:put:

Semantycznie podstawowym rodzajem wyrażenia jest **wysłanie komunikatu,** tj. zastosowanie komunikatu zapisanego jako selektor z argumentami (wyrażeniami) do odbiorcy komunikatu (wyrażenia). Jest ono interpretowane jako żądanie wykonania metody wyznaczonej przez selektor (p. część II) w obiekcie będącym odbiorcą komunikatu; obiekty będące argumentami komunikatu są przypisane (według kolejności występowania) parametrom wyznaczonej metody.

W zapisie wysłania komunikatu występuje najpierw wyrażenie oznaczające odbiorcę komunikatu, a po nim komu-

nikat. Jeśli selektor komunikatu jest unarny — to nie występuje argument **(komunikat unarny)**, jeśli binarny — to po nim wystąpi dokładnie jeden argument **(komunikat binarny)**, jeśli kluczowy — to po każdym składowym kluczu wystąpi odpowiadający mu argument **(komunikat kluczowy)**.

Przykłady

1. Nazewnik funkcji **sin(x)** zapisuje się jako wysłanie komunikatu unarnego **sin** do obiektu **x** tj. x sin.

2. Wyrażenie k + 1 należy rozumieć jako komunikat binarny o selektorze + oraz argumencie 1 wysłany do obiektu będącego wartością zmiennej o nazwie k.

3. Pobranie wartości zmiennej indeksowej **list** o wartości indeksu **index** zapisujemy jako komunikat kluczowy o selektorze **at:** oraz argumencie **index**, wysłany do obiektu **list** tzn.: list at: index;
Przypisanie zmiennej indeksowej **list** nowej wartości **x** zapisujemy za pomocą komunikatu kluczowego o selektorze **at:put:** w następujący sposób:
list at: index put: x

4. Pierwiastek kwadratowy z wyrażenia **sin(x)** zapisujemy jako x sin sqrt

5. Wyrażenie języka Pascal
list[k+1] := —a + b
zapisujemy jako wysłanie do obiektu **list** komunikatu o selektorze **at:put:** i nieco złożonych argumentach:
list at: k+1 put: a negated + b

Częstym wypadkiem w praktyce jest wysłanie sekwencji komunikatów do jednego i tego samego odbiorcy. Można to krótko zapisać jako **kaskadę komunikatów** — po wysłaniu pierwszego komunikatu w zwykły sposób, wysłanie każdego następnego z kaskady zapisujemy stawiając znak średnika i zadając tylko komunikat (selektor i argumenty), bez powtarzania odbiorcy określonego przy wysłaniu pierwszego komunikatu. Przykładowo, kaskada komunikatów:

printer newline; print: title; newlines: 2;
print: author; newlines: 20;
print: placeDate; newline

jest równoważna następującemu ciągowi wysłań komunikatów: printer newline. printer print: title. printer newlines:2. itd.

Odbiorca i argumenty komunikatu są wartościami wyrażeń prostych albo obiektami wyznaczonymi znowu przez wysłanie komunikatu. Musi więc być ściśle określona **kolejność wartościowania wyrażeń** składowych; jest ona następująca:

— wyrażenia proste,

— wyrażenia wysłania komunikatów unarnych wartościowane z lewa na prawo;

— wyrażenia wysłania komunikatów binarnych wartościowane z lewa na prawo,

— wyrażenia wysłania komunikatów kluczowych (argumenty są wartościowane z lewa na prawo),

— przypisania (wykonywane na zmiennych z prawa na lewo).

Przykłady

1. Wyrażenie

2+3*4

nie ma wartości 14 (może w pierwszej chwili nieoczekiwanie), ale zgodnie z zasadą wartościowania komunikatów binarnych z lewa na prawo jest ono równoważne wyrażeniu (2+3)*4 tzn. ma wartość 20.

2. Wyrażenie:

x*x + y*y sqrt

jest wartościowane tak jak wyrażenie:

((x*x) + y)*(y sqrt)

Aby uzyskać oczekiwaną kolejność wartościowania należy użyć nawiasów:

((x*x) + (y*y) sqrt

3. Jeżeli wynik wysłania komunikatu kluczowego ma być odbiorcą lub argumentem innego komunikatu kluczowego, to należy stosować nawiasy. Przykładowo, pominięcie nawiasów w wyrażeniu:

list at: 1 put: (list at: 2)

spowoduje wysłanie komunikatu o selektorze:

at:put:at:

(prawdopodobnie błąd) zamiast oczekiwanego przypisania: list[1] := list[2]

4. Wyrażenie:

Vector new x: aVector x * 2 y: aVector y * 2

jest równoważne następującemu (x:y: jest selektorem metody reprezentatywnej w klasie Vector :

(Vector new) x: ((aVector x) * 2) y: ((aVector y) * 2)

Komunikat **new**, skierowany do dowolnej klasy, tworzy jej nowego reprezentanta; zwykle definiuje się dodatkowe metody klasowe, które nie tylko tworzą nowego reprezentanta, ale od razu inicjują jego zmienne reprezentatywne.

Przykładowo, w klasie Vector mogłaby to być metoda o selektorze:

newX:newY:

Wówczas powyższe wyrażenie komunikatowe można zapisać czytelniej jako:

Vector newX: aVector x * 2 newY: aVector y * 2.

dokończenie s. 17

RYSZARD S. MICHALSKI
University of Illinois
at Urbana-Champaign
USA

O naturze uczenia się — problemy i kierunki badawcze (2)

W drugiej, końcowej części artykułu omówiono paradygmaty, strategie i orientacje uczenia się.

PARADYGMATY BADAWCZE

Od chwili rozpoczęcia badań nad maszynowym uczeniem się, w latach pięćdziesiątych, w różnych okresach koncentrowano się na różnych podejściach i stawiano sobie różne cele badawcze. Można wyróżnić trzy główne paradygmaty badań w tej dziedzinie:

- modelowanie sieci neuronów i techniki teoriodezyjne,
- symboliczne przyswajanie pojęć,
- uczenie się w wąskiej dziedzinie z intensywnym wykorzystaniem wiedzy.

Podejścia te różnią się przede wszystkim pod względem wiedzy posiadanej **a priori** przez system uczący się i pod względem sposobu, w jaki wiedza jest **reprezentowana** i **modyfikowana** w systemie.

W podejściu polegającym na modelowaniu sieci neuronów dąży się do zbudowania uniwersalnych systemów uczących się, zaczynających pracę od niewielkiej wiedzy początkowej. Systemy takie nazywa się **sieciami neuronowymi** lub **systemami samoorganizującymi się**. System tego rodzaju składa się z sieci wzajemnie połączonych elementów, zazwyczaj podobnych w swym działaniu do neuronów, realizujących jakąś prostą funkcję logiczną, na przykład funkcję progową. Taki system uczy się drogą przyrostowego modyfikowania **trwałości połączeń** między elementami, zwykle przez zmianę wag związanych z tymi połączeniami. Początkowa wiedza systemu pochodzi z wyboru elementów wejściowych, reprezentujących określone atrybuty badanych obiektów oraz ze struktury i początkowej trwałości połączeń w sieci. Może to być struktura losowa, struktura przygotowana przez projektanta lub jakieś rozwiązanie pośrednie. Do systemów tego rodzaju zalicza się **Perceptron** [23], **Pandemonium** [24] i każdą maszynę uczącą się, wykorzystującą **funkcje dyskryminacyjne** [20]. Przykładami nowszych systemów wywodzących się z tego paradygmatu są różne adaptacyjne systemy sterowania [24]. Badania w tej dziedzinie doprowadziły **do powstania** podejścia opartego na zastosowaniu **teori decyzji** w rozpoznawaniu obrazów. **Wiążą się** z tym badania nad **ewolucyjnym uczeniem się** [5, 6] i **algorytmy genetyczne** [8, 18, rozdział 20]. Jak wspomniano, obserwuje się ponowny wzrost zainteresowania tym paradygmatem uczenia się, w związku z ostatnimi próbami budowy nowych maszyn neuronowych [7].

Charakterystycznymi cechami systemów budowanych według tego paradygmatu są: niski poziom wiedzy wbudowanej **a priori** oraz wykorzystanie parametrów zmieniających się w sposób ciągły, w trakcie uczenia się. Inną cechą tych systemów jest numeryczny charakter metod i algorytmów uczenia się. Kontrastuje to z dwoma następnymi paradygmatami, w których główny nacisk kładzie się na tworzenie i manipulowanie skomplikowanymi strukturami symbolicznymi.

W metodzie **symbolicznego przyswajania pojęć** (ang. symbolic concept acquisition, SCA) system uczy się, konstruując symboliczną reprezentację danego zbioru pojęć drogą analizy przykładów i kontrprzykładów tych pojęć. Reprezentacja może mieć postać wyrażenia logicznego, drzewa decyzyjnego, reguł produkcji lub sieci semantycznej. Niektóre spośród systemów skonstruowanych według tego paradygmatu znalazły praktyczne zastosowanie w wielu dziedzinach. Przykładami mogą być programy: ARCH [26], AQVAL [11] i ID3[22]. W tym paradygmacie, atrybuty oraz predykaty odnoszące się do danego pojęcia są wprowadzane do systemu przez nauczyciela (instruktora).

W paradygmacie **uczenia się w wąskiej dziedzinie z intensywnym wykorzystaniem wiedzy** (ang. knowledge-intensiwe domain-specific learning, KDL) system zawiera liczne (zdefiniowane z góry) pojęcia, struktury wiedzy, ograniczenia dziedzinowe, reguły heurystyczne i wbudowane na stałe transformacje odnoszące się do konkretnej dziedziny. Nie wszystkie niezbędne atrybuty i pojęcia są znane na wstępie; oczekuje się, że system wygeneruje nowe w procesie uczenia się (autor nazywa proces tego rodzaju **indukcja konstruktywna**). Tak więc, główną różnicą między paradygmatem uczenia się w wąskiej dziedzinie z intensywnym wykorzystaniem wiedzy a paradygmatem symbolicznego przyswajania pojęć jest ilość i rodzaj wiedzy podstawowej wprowadzonej do systemu. Systemy uczące się oparte na tym pierwszym podejściu są zwykle budowane dla konkretnej dziedziny i nie mogą być bezpośrednio wykorzystywane w innej dziedzinie. Badania związane z tym paradygmatem dotyczą nie tylko **uczenia się na przykładach**, lecz również **uczenia się przez analogię** i **uczenia się przez obserwację** i **odkrywanie** (p. następny punkt). Przykładami systemów opartych na tym podejściu są: Meta-DENDRAL [3] i AM [10].

Wiele budowanych ostatnio systemów wykorzystuje obydwa powyższe podejścia. Ciekawą kombinację paradygmatów SCA i KDL zastosowano w systemach opartych na idei **wymiennego modułu wiedzy**. System taki wiąże ogólne mechanizmy uczenia się z wbudowanymi mechanizmami definiowania i używania wiedzy specjalistycznej. Jeśli ma on być zastosowany do rozwiązania określonego problemu, to niezbędna wiedza musi być najpierw dostarczona przez nauczyciela. Dzięki oddzieleniu ogólnych umiejętności wnioskowania od wiedzy specjalistycznej, takiego systemu można używać w wielu różnych dziedzinach, zachowując przy tym zalety systemów specjalistycznych. Na tej zasadzie jest oparty m.in. system INDUCE, który uczy się strukturalnego opisu obiektów na podstawie przykładów [12]. Innym przykładem takiego podejścia jest program Winstona uczący się przez analogię [27]. Również system LEX udoskonalający heurystyki [19] oraz program Eurisko odkrywający nowe heurystyki [10] mogą być zaliczone do tej kategorii. W książce [18, rozdział 14] opisano podejście tego rodzaju oparte na analogi wynikowej (ang. derivational analogy).

Historię badań związanych z tymi trzema paradygmatami przedstawiono w [18, rozdział 1]. Przykłady aktualnie prowadzonych badań nad systemami samoorganizującymi sie można znaleźć w publikacji [4]. Langley i Carbonell dokonali przeglądu podejść do maszynowego uczenia się [9]. Książka [18] koncentruje się przede wszystkim na symbolicznym przyswajaniu wiedzy i uczeniu się w wąskiej dziedzinie, z intensywnym wykorzystaniem wiedzy.

STRATEGIE UCZENIA SIĘ

Uczeń transformuje informację dostarczoną mu przez nauczyciela (lub środowisko) na pewną nową postać, którą zapamiętuje. Sposób tego przekształcania określa typ użytej strategii uczenia się. Wyróżnia się następujące podstawowe strategie: uczenie się na pamięć, uczenie się przez instruktaż, uczenie się przez dedukcję, uczenie się przez analogię

i uczenie się przez indukcję. W tej ostatniej strategii można wyodrębnić uczenie się na przykładach oraz uczenie się przez obserwację i odkrywanie. Powyższe strategie zostały wymienione w kolejności rosnącej złożoności procesu przekształcającego początkowe informacje w docelową wiedzę. Kolejność ta odpowiada wzrostowi wysiłku ze strony uczenia i zmniejszaniu wysiłku ze strony nauczyciela. Wyróżnienie wymienionych strategii jest istotne z punktu widzenia dydaktyki, jak również przyszłego rozwoju systemów uczących się. Człowiek w procesie uczenia się stosuje zwykle łącznie kilka strategii. Pomimo że większość obecnych systemów uczących się jest oparta na jednej strategii, należy oczekiwać, że w przyszłości więcej uwagi poświęci się systemom wielostrategicznym. Poniżej podano krótką charakterystykę strategii uczenia się. Opis szczegółowy można znaleźć w [13].

W **uczeniu się na pamięć** w zasadzie nie następuje żadne przekształcenie informacji. Informacja jest zapamiętywana w postaci podawanej przez nauczyciela. Głównym zadaniem systemu jest odpowiednie poindeksowanie informacji w celu usprawnienia procesu wyszukiwania. W **uczeniu się przez instruktaż** informacja od nauczyciela podlega **selekcji** i **przeformułowaniu** (głównie na poziomie syntaktycznym). W **uczeniu się przez dedukcję** uczeń przeprowadza dedukcję na podstawie wiedzy i zapamiętuje użyteczne wnioski (strategia ta została wyodrębniona jako osobna kategoria dopiero ostatnio [13, 17]).

Jeśli w procesie uczenia się najpierw uogólnia się informacje, a następnie ocenia wyniki (czyli przeprowadza się wnioskowanie indukcyjne), to mamy do czynienia z **uczeniem się przez indukcję.** Uczenie się przez analogię łączy w sobie uczenie się przez dedukcję i indukcję. Porównuje się w nim opisy z różnych dziedzin w celu wyodrębnienia wspólnej podstruktury, która jest następnie odwzorowywana przez analogię. Poszukiwanie wspólnej struktury jest oparte na wnioskowaniu indukcyjnym, natomiast odwzorowywanie przez analogię jest formą dedukcji. Uczenie się przez przypominanie (ang. learning by reminding) można również uważać za pewną formę uczenia się przez analogię.

Uczenie się przez analogię opisano w trzech rozdziałach książki [18]. Burstein [18, rozdział 13] przedstawił model uczenia się z wykorzystaniem rozumowania przez analogię, opisując go w kontekście przyswajania semantyki instrukcji przypisania języka Basic. Zgodnie z tym modelem, użycie analogii do uczenia się pojęć w nowej dziedzinie zależy ściśle od przyczynowych abstrakcji utworzonych uprzednio w jakiejś dziedzinie pokrewnej. Analogie te są stopniowo rozbudowywane w celu dostosowywania ich do coraz większej liczby sytuacji. Z kolei, Carbonell [18, rozdział 14] przedstawił teorię **analogi wynikowej** i jej wpływ na wnioskowanie i przyswajanie ekspertyz. Wnioskowanie prowadzące do rozwiązania pewnej klasy problemów może być powtarzane i modyfikowane do rozwiązania nowych, bardziej skomplikowanych problemów. Metodę tę można wykorzystać do automatycznego przyswajania wiedzy i umiejętności w systemach ekspertowych. Dershowitz [18, rozdział 15] omówił wykorzystanie analogii w automatycznym programowaniu, pokazując jak można posłużyć się analogiami między specyfikacjami programów na etapie uruchamiania lub modyfikowania istniejącego programu w celu rozszerzenia jego możliwości. Analogie mogą być również wykorzystane do wyodrębnienia schematu abstrakcyjnego programu ze zbioru programów oraz do ukonkretniania schematu w celu otrzymania właściwego programu.

Uczenie się przez indukcję można podzielić na **uczenie się na przykładach** i **uczenie się przez obserwację i odkrywanie.** W uczeniu się na przykładach (zwanym również **przyswajaniem pojęć**) celem jest ustalenie uogólnienia tłumaczącego wszystkie przykłady pozytywne i wykluczającego wszystkie przykłady negatywne danego pojęcia. Źródłem przykładów może być nauczyciel, znający dane pojęcie lub środowisko, w którym uczeń wykonuje doświadczenia i z którego otrzymuje odpowiedzi. W tym drugim wypadku mamy do czynienia z **uczeniem się przez eksperymentowanie,** które obejmuje **uczenie się przez działanie** i **uczenie się przez rozwiązywanie problemu.** Uczenie się na zasadzie **pobudzenie-odpowiedź** (ang. stimulus-response learning) może być również sklasyfikowane jako uczenie się na przykładach.

Najnowsze badania wyodrębniły dwa ciekawe rodzaje uczenia się na przykładach: uogólnianie na zasadzie **egzemplarz-klasa** (ang. instance-to-class generalization) oraz uogólnianie na zasadzie **część-całość** (ang. part-to-whole generalization). W uogólnieniu na zasadzie egzemplarz-klasa system analizuje niezależne przykłady obiektów należących do pewnej klasy celem wyindukowania ogólnego opisu klasy. Większość dotychczasowych prac związanych z uczeniem się na przykładach dotyczy uogólniania na zasadzie egzemplarz-klasa. Obiektami mogą tu być struktury złożone z bloków, figury geometryczne, opisy chorób, opowiadania, rozwiązania problemów, operatory itp. Różne aspekty zadań tego rodzaju przedyskutowano w [18].

Winston [18, rozdział 3] podjął próbę interpretacji takich zagadnień, jak uczenie się z wykorzystaniem precedensów i ćwiczeń, użycie przybliżeń (ang. near-misses) w uczeniu się, uogólnienie reguł „if-then" i zastosowanie warunków „unless" zabezpieczających przed niewłaściwym użyciem reguły. Rola warunków „if-then" polega na zablokowaniu reguły „unless" pomimo spełnienia warunków zawartych w regule. Sposób ten ułatwia stopniowe korygowanie reguł. Utgoff [18, rozdział 5] zbadał rolę wskaźnika skłonności lub preferencji w określaniu prawdopodobnych hipotez uczenia się przez indukcję. Przedstawił pełną metodologię oraz program STABB, modyfikujący wskaźnik skłonności w trakcie uczenia się na przykładach. Quinlan [18, rozdział 6] rozważył wpływ szumów (zakłóceń) zawartych w analizowanych przykładach na proces odkrywania reguł klasyfikacyjnych i ich dokładność. Podał też szereg ciekawych propozycji, jak należy formułować cel uczenia się w wypadku, gdy przykłady będą zawierały szum.

Sammut i Banerji [18, rozdział 7] badali rolę przyswojonych już pojęć w uczeniu się nowych pojęć oraz zagadnienie uczenia się przez indukcję w wypadku **aktywnego** ucznia. Uczeń taki nie tylko akceptuje biernie przykłady pochodzące od nauczyciela, lecz również sam je generuje, pytając nauczyciela o ich przynależność do badanej klasy. Podobny problem rozważył Lebowitz [18, rozdział 8], który badał możliwość wykorzystania przechowywanych w pamięci pojęć w procesie uogólniania złożonych opisów strukturalnych. Jego **metoda pamięci opartej na uogólnianiu** (ang. generalisation-based memory) pozwala wybrać pojęcia do nauczenia się oraz sformułować ich definicje. Do jej realizacji służą dwa programy — program oceniający pojęcia oraz program uogólniający złożone opisy strukturalne. Konratoff i Ganascia [18, rozdział 9] rozważyli różne teoretyczne aspekty procesu uogólniania i pokazali, jak można przeprowadzić uogólnianie tworząc związki między uczeniem się na przykładach (połączenia są reprezentowane przez wiązanie zmiennych). Przegląd innych metod uogólniania można znaleźć w [2] i [16].

W uogólnianiu na zasadzie **część-całość** dane są wybrane części obiektu (np. sceny, sytuacji, procesu), a celem jest wywnioskowanie opisu całego obiektu. Przykładem takiego wnioskowania może być rekonstrukcja wyglądu całego pomieszczenia na podstawie serii zdjęć jego części. Innym przykładem takiej formy uogólniania jest ustalenie reguły charakteryzującej ciąg obiektów lub proces na podstawie kilku kolejnych obiektów tego ciągu lub części procesu. Podstawy teoretyczne i metodologię uogólniania na zasadzie **część-całość** przedstawiono w [18, rozdział 4]. Opisano tu ogólną metodę opartą na kilku modelach regułowych, które pozwalają znaleźć regułę charakteryzującą dany ciąg obiektów i wyznaczającą dopuszczalne ciągi obiektów. Każdy obiekt ciągu jest opisany przez pewną liczbę atrybutów, z których niektóre są podane a priori, inne zaś wyprowadzane na podstawie reguł wnioskowania i transformacji ciągów. Z zagadnieniami uogólniania na zasadzie **część-całość** są ściśle powiązane badania nad **przewidywaniem procesów jakościowych** [17].

W uczeniu się przez obserwację i odkrywanie (zwanym również **uogólnianiem opisowym**) uczeń poszukuje bez pomocy nauczyciela reguł wyjaśniających zaobserwowane fakty. Ta forma uczenia się obejmuje **grupowanie koncepcji** (ang. conceptual clustering), czyli tworzenie klas obiektów dających się opisać przez proste pojęcia, klasyfikowanie, odkrywanie praw wyjaśniających zbiór obserwacji i formułowanie teorii opisujących działanie systemu. Jako warianty tej strategii można traktować **algorytmy genetyczne** [18, rozdział 20] i **algorytmy przewidywania empirycznego** [28]. Rozmaite jej aspekty przedyskutowano w [18].

Langley i in. [18, rozdział 16] opisali cztery systemy dotyczące różnych rodzajów odkryć naukowych. Program BAKON6 formułuje prawa empiryczne charakteryzujące dowolne liczbowe dane doświadczalne, GLAUBER odkrywa jakościowe prawa reakcji chemicznych, STAHL zajmuje się określeniem składników substancji w tych reakcjach, a

DALTON koncentruje się na formułowaniu strukturalnych modeli takich reakcji. W [18, rozdział 17] omówiono prace nad grupowaniem koncepcji, tzn. klasyfikowaniem obserwacji przez idetyfikowanie podklas odpowiadających prostym pojęciom. W przeciwieństwie do poprzednich prac nad niezależną od celu (ang. goal-free) klasyfikacją obiektów niestrukturalnych, nowe podejście zajmuje się ukierunkowaną (ang. goal-oriented) klasyfikacją obiektów strukturalnych. Autorzy pokazali na przykładach, w jaki sposób wykorzystuje się koncepcje uczenia i reguły wnioskowania do tworzenia takich klasyfikacji. Amarel rozważał zagadnienia formowania teorii w kontekście syntezy programów. Swą metodę zilustrował w [18, rozdział 18] za pomocą zadania „wywnioskowania" programu z powiązań danych wejściowo-wyjściowych, określonych w dziedzinie częściowo uporządkowanych struktur. Metoda ta akcentuje rolę modeli algebraicznych i geometrycznych oraz znaczenie wprowadzenia reprezentacji problemu do zadania syntezy programu. Inny kierunek obrał De Jong [18, rozdział 19]. Jego metoda uczenia się przez obserwację polega na kierowaniu procesem uogólniania na podstawie pojedynczego przykładu, przy wykorzystaniu wewnętrznych ograniczeń między pojęciami zawartymi w podstawowej wiedzy systemu. Jako przykłady dobrał on opowiadania o zachowaniu się ludzi przy rozwiązywaniu zadań.

Podstawą uczenia się przez indukcję i analogię jest wnioskowanie indukcyjne. Wnioskowanie indukcyjne opiera się na zbiorze faktów (obserwacji) oraz ewentualnym zbiorze hipotez a priori dotyczących tych faktów, a w wyniku daje najprawdopodobniejsze uogólnienie wyjaśniające te fakty. Jak już wspomniano, jest to wnioskowanie nie zachowujące prawdy, które przeprowadza się za pomocą reguł wnioskowania uogólniającego [14]. Jak stwierdził Popper [21], „czysta" indukcja, tzn. bezpośrednie wywnioskowanie teorii z faktów bez pomocy pojęć wyjaśniających, jest niemożliwa. Pojęcia te są potrzebne do opisu obserwacji i stanowią część wiedzy podstawowej ucznia. Ta wiedza jest niezbędnym składnikiem każdego procesu indukcyjnego. Zawiera ona również cele uczenia się, ograniczenia specyficzne dla danej dziedziny, związki przyczynowo-skutkowe, heurystyki i skłonności ukierunkowujące proces uogólniania oraz kryteria oceny stawianych hipotez.

Można wyróżnić dwie techniki kierowania i ograniczania procesu uogólniania: technikę opartą na podobieństwach lub na ograniczeniach. **Technika oparta na podobieństwach** bada powiązania między przykładami, tzn. analizuje pozytywne i negatywne przykłady pojęcia w celu stworzenia jego opisu. Poszukuje się cech wspólnych dla faktów lub przykładów tej samej klasy, jak również związków przyczynowo-skutkowych i wyjaśnień, dzięki którym pozornie różne przykłady można zaliczyć do tej samej klasy. Uogólnianie następuje w wyniku ignorowania różnic między przykładami lub przez formułowanie pojęć określających te różnice. Kilka wczesnych metod wykorzystujących tę technikę opisano w [18, rozdział 3].

Technika oparta na ograniczeniach bada związki między przykładami, które ograniczają pojęcia wyjaśniające dane przykłady. Każde z zastosowanych uogólnień musi spełniać te ograniczenia. Na przykład, uogólniając fakt „pudełko leży na stole" trzeba uwzględnić ograniczenie, że przedmiot leżący na stole nie może być ani zbyt ciężki (żeby nie załamał stołu), ani zbyt duży (żeby zmieścił się na stole). Jeden z wariantów tej techniki opisano w [1], wprowadzając pojęcie **uzasadniania** hipotezy. Inny wariant jest nazywany przez niektórych autorów **uogólnianiem opartym na wyjaśnianiu**, w celu podkreślenia znaczenia stosowanej wiedzy wyjaśniającej (termin ten nie jest zbyt precyzyjny, gdyż każdy rodzaj uczenia się przez indukcję zawiera w definicji poszukiwanie wyjaśnień stwierdzonych faktów). W [18, rozdział 19] De Jong omówił metodę wykorzystania tej techniki do rozumienia opowiadań. Techniki oparte na podobieństwach i ograniczeniach dopełniają się wzajemnie i w systemach uczących się mogą być stosowane łącznie.

ORIENTACJE UCZENIA SIĘ

W poprzednich punktach podano dwa ważne kryteria klasyfikacji badań nad maszynowym uczeniem się: paradygmaty uczenia się i strategie uczenia się. Pierwsze kryterium dotyczy rodzaju wiedzy reprezentowanej i przetwarzanej w systemie, drugie jest związane z rodzajem wykorzystanego wnioskowania. W tym punkcie omówiono pokrótce jeszcze jedno kryterium klasyfikacji, a mianowicie orientacje badań, które określają zakres i temat prowadzonych prac. Używając obrazowego porównania, paradygmatowi odpowiada punkt startowy i teren, po którym się poruszamy, strategia określa środek lokomocji, a orientacja — kierunek jazdy.

W badaniach dotyczących maszynowego uczenia się można wyróżnić trzy wzajemnie powiązane orientacje:

- analiza teoretyczna i rozwój ogólnych algorytmów uczenia się,
- rozwój obliczeniowych modeli opisujących procesy ludzkiego uczenia się,
- badanie zastosowań wraz z tworzeniem specjalistycznych systemów uczących się (zwane również orientacją inżynierską).

Pierwsza orientacja bada teoretyczne (lub uproszczone rzeczywiste) zadania uczenia się i rozwija algorytmy, które realizują te zadania niezależnie od dziedziny zastosowań. Nie wprowadza się żadnych ograniczeń co do rodzaju konstruowanego algorytmu. Nie musi on być zgodny z metodami uczenia się stosowanymi przez ludzi. Niektórzy autorzy postulują jednak, aby przynajmniej struktury wiedzy powstającej w wyniku maszynowego uczenia się były podobne do tych, które może utworzyć człowiek; same procesy tworzenia struktur wiedzy mogą być różne [15]. Ten kierunek badań zmierza do określenia teoretycznej przestrzeni algorytmów uczenia się. Prace typowe dla tej orientacji przedstawiono w [18, rozdziały 3, 5, 7 i 9].

Druga orientacja, zwana również **modelowaniem poznawczym** (ang. cognitive modeling), koncentruje się na sposobach uczenia się przez człowieka, konstruując teorie obliczeniowe i modele eksperymentalne. Badania te mogą mieć znaczny wpływ na rozwój technik budowy systemów maszynowego uczenia się, jak też na szeroko rozumianą edukację człowieka. Rosenbloom i Newell [18, rozdział 10] opisali pewne podejście do modelowania procesów leżących u podstaw zwiększania umiejętności przez praktykę. Ich model praktyki jest oparty na pojęciu grupowania podcelów w cele. Pokazali oni, że zaproponowany model odzwierciedla znaną zasadę mówiącą o skuteczności ludzkiej praktyki. Anderson [18, rozdział 11] rozważył mechanizmy uczenia się wykorzystywane w **kompilacji wiedzy**, tzn. w procesie przechodzenia od deklaratywnej reprezentacji umiejętności do reprezentacji proceduralnej. Pokazał on, jak mechanizmy **kompozycji** (łączenia kilku reguł produkcji w jedną) i **proceduralizacji** (tworzenia reguł produkcji na podstawie wiedzy deklaratywnej) mogą symulować początkowe stadia zdobywania umiejętności w dziedzinie uczenia się programowania.

Trzecia orientacja badawcza koncentruje się na praktycznych zadaniach uczenia się i budowie systemów realizujących te zadania. Jej przykładem może być program uczący się rozpoznawać warunki niebezpieczne dla samolotu w czasie lotu. W tego rodzaju zadaniach pojawiają się na ogół inne problemy, nie związane bezpośrednio z uczeniem się, np. interpretacja sygnałów wejściowych i przetwarzanie danych. W orientacji tej korzysta się z osiągnięć dwóch pozostałych orientacji badawczych. Często po znalezieniu rozwiązania konkretnego problemu, podejmuje się próbę uogólnienia go w celu skonstruowania metody rozwiązywania szerszej klasy podobnych problemów. Przykłady takich badań opisano w [18, rozdział 4].

Powyższe trzy orientacje tworzą triadę wzajemnie powiązanych i uzupełniających się kierunków badań w dziedzinie maszynowego uczenia się. Taki podział przeniknął również do pozostałych działów sztucznej inteligencji.

* * *

Niniejszy artykuł był w pierwotnej wersji wprowadzeniem do książki [18]. Jej zawartość dotyczy głównie paradygmatów symbolicznego przyswajania pojęć oraz uczenia się w wąskiej dziedzinie z intensywnym wykorzystaniem wiedzy i jest poświęcona omówieniu strategii uczenia się przez indukcję i przez analogię. Reprezentowane są w niej oba główne rodzaje indukcyjnego uczenia się, tj. uczenie się na przykładach oraz uczenie się przez obserwację i odkrywanie.

Książka zawiera również bibliografię prac na temat maszynowego uczenia się, począwszy od 1980 roku, z kilkoma ważniejszymi uzupełnieniami z lat wcześniejszych (obszerną bibliografię dotyczącą poprzedniego okresu zawarto w

książce [13]). Bibliografia zawiera też skorowidz zawartości, opracowany według strategii uczenia się, dziedziny zastosowań i metody badawczej. Do książki włączono także słowniczek terminów używanych w dziedzinie maszynowego uczenia się oraz noty bibliograficzne wszystkich autorów.

Artykuł powstał podczas pobytu autora w Artificial Intelligence Laboratory, w Massachusetts Institute of Technology. Autor pragnie podziękować Patrickowi Winstonowi za zaproszenie do współpracy oraz wiele zapładniających uwag i komentarzy na temat poprzednich wersji tego artykułu. Niepowtarzalna, twórcza atmosfera AT Laboratory i dyskusje prowadzone w Learning Group były pomocne w opracowaniu przedstawionych tu idei. Częściową pomoc finansową uzyskano w ramach kontraktu N00014-80-C-0505 z Advanced Research Projects Agency.

Autor dziękuje współredaktorom książki Jaime Carbonellowi i Tomowi Michelowi za współpracę i uwagi krytyczne. Dyskusje z Randy Davisem, z MIT AI Laboratory, były bardzo pomocne w udoskonalaniu artykułu. Zaproponował on zastosowanie maszyn uczących się do zabezpieczenia się przed niewłaściwym wykorzystaniem ich samych. Richard Doyle, Michel Kaskhet, Boris Katz i David Kirsh, z MIT AI Laboratory, oraz Allan Collins, z firmy Bolt, Beranek and Newman Inc. przedstawili ważne opinie i komentarze dotyczące pierwotnej wersji artykłu. Autor wyraża wdzięczność Bobowi Steppowi, Larry'emu Rendellowi, Jeffowi Backerowi, Bruce'owi Katzowi i Brianowi Stoutowi, z AI Laboratory Uniwersytetu stanu Illinois, za wartościowe sugestie i konstruktywną krytykę. Wiele pożytecznych uwag przedstawił Ken Forbus, z Uniwersytetu stanu Illinois. Autor jest zobowiązany Gail Thornburg z Wyższej Szkoły Bibliotekarskiej za cenne uwagi i krytykę. Ważne sugestie przedstawił Jan Gorecki z Wydziału Socjologii Uniwersytetu stanu Illinois. Praca ta była finansowana przez National Science Fundation w ramach umowy DCR-8406801.

Tłum. i oprac.

EDMUND PIERZCHAŁA
PIOTR ZIELCZYŃSKI

LITERATURA

[1] Andreae P. M., Constraint Limited Generalization — Acquiring Procedures from Examples. Proc. AAAI-84, Austin (TX), August 1984
[2] Cohen P. R., Feigenbaum E. A. (eds.): The Handbook of Artificial Intelligence, Vol. III. Kaufmann, Los Altos (CA), 1982
[3] Buchanan B. G., Shortliffe E. H. (eds.): Rule-based Expert Systems. Addison-Wesley, Reading (MA), 1984
[4] Caianiello E. R., Musso G.: Cybernetic Systems — Recognition. Learning, Self-Organization. Rerearch Studies Press, Letchworth (Hertfordshire), Wiley, New York, 1984
[5] Conrad M.: Adaptibility. Plenum Press, New York 1983
[6] Fogel L., Owens A., Walsh M.: Artificial Intelligence Through Simulated Evolution, Wiley, New York, 1966
[7] Hinton G. E., Sejnowski T. J., Ackley D. H.: Machines — Constraint Satisfaction Networks that Learn. Technical Report CMU-CS-84-119, Carnegie-Mellon University, Pittsburgh (PA), 1984
[8] Holland J.: Adaptation in Natural and Artificial Systems, niversity of Michigan Press, Ann Arbor, 1975
[9] Langley P., Carbonell J. G.: Approaches to Machine Learning. Journal of the American Society for Information Science, Vol. 35, No. 5, 1984
[10] Lenat D. G.: The Role of Heuristics in Learning by Discovery — Three Case Studies. [18]
[11] Michalski R. S.: Variable-valued Logic and Its Applications to Patteran Recognition and Machine Learning. Pp. 506—534, Computer Science and Multiple-valued Logic — Theory and Applications, D. C. Rine (ed.), North-Holland, Amsterdam, 1975
[12] Michalski R. S.: Pattern Recognition as Rule-Guided Inductive Inference. IEEE Trans. on Pattern Analysis and Machine Intelligence. Vol. PAMI-2, No. 4, pp. 349—361, July 1980
[13] Michalski R. S., Carbonell J. G., Mitchell T. M. (eds.): Machine Learning — An Artificial Intelligence Approach. TIOGA, Palo Alto (CA), 1983
[14] Michalski R. S.: A Theory and Methodology of Inductive Learning. Artificial Intelligence, No. 20, 1983
[15] Michalski R. S. (ed.): Proc. of the International Machine Learning Workshop, Allerton House, University of Illinois, 22—24 June 1983
[16] Michalski R. S.: Learning Strategies and Automated Knowledge Acquisition — An Overview. Knowledge-Based Learning Systems, Bolc L. (ed.), Springer-Verlag, 1985
[17] Michalski R. S., Ko H., Chen K.: Qualitative Process Prediction — A Method and a Program SPARC/G, Report ISG-12, Dept. of Computer Science, University of Illinois, Urbana (IL), 1985
[18] Michalski R. S., Carbonell J. G., Mitchell T. M. (eds.): Machine Learning. Vol. 2. Morgan and Kaufmann Publishers, 1986
[19] Mitchell T. M., Utgoff P. E., Banerji R.: Learning by Experimentation — Acquiring and Refining Problem-Solving Heuristics. [18]
[20] Nillson N. J.: Maszyny uczące się, PWN, Warszawa, 1968
[21] Popper K. R.: Objective Knowledge — An Evolutionary Approach, Revisededition. Oxford, 1981
[22] Quinlan J. R.: Discovering Rules form Large Collections of Examples — A Case Study. Expert Ssystems in the Microelectronics Age, D. Michie (ed.), Edinburgh niversity Press, 1979
[23] Rosenblatt F.: The Perceptron — A Probabilistic Model for Information Storage and Organization in the Brain. Psychological Review, Vol. 65, p. 386—407, 1958
[24] Selfridge M.: Pandemonium — A paradigm for Learning. Pp. 511—529, Proc. of the Symposium on Mechanization of Thought Processes, D. Blake, A. Uttley eds., HMSO, London, 1959
[25] Cypkin J. Z.: Podstawy teorii układów uczących się. WNT, Warszawa, 1973
[26] Winston P. H.: Learning Structural Descriptions from Examples. The Psychology of Computer Vision, P. H. Winston (ed.), McGraw-Hill, New York, 1975
[27] Winston P. H.: Learning and Reasoning by Analogy. Communications of the ACM, Vol. 19, No. 3, 1982
[28] Zagoruiko N.: Empirical Prediction Algorithms, Computer Oriented Learning Processes, J. C. Simon (ed.), Noordhoff, Leyden, 1976.

Pamięci dyskowe Mazovii 16

dokończenie ze str. 22

Normalnym sposobem pracy sterownika z drukarką jest początkowe zatrzaśnięcie informacji (8 bitów), a następnie wysterowanie linii Strobe, które powoduje wprowadzenie danych lub rozkazów do drukarki. Następnie można odczytać stan drukarki informujący, kiedy można rozpocząć następne przesłanie.

Tabela 4. Wykaz linii sprzęgu sterownika drukarki

Nazwa i polaryzacja sygnału	Numer kontaktu złącza
— Strobe	1
+Dane bit 0	2
+Dane bit 1	3
+Dane bit 2	4
+Dane bit 3	5
+Dane bit 4	6
+Dane bit 5	7
+Dane bit 6	8
+Dane bit 7	9
—Acknowledge	10
+Busy	11
+P.End	12
+Select	13
—Auto Feed	14
—Error	15
—Initialize Printer	16
—Select Input	17
Masa	18—25

Ponieważ wszystkie linie wyjściowe ze sterownika mogą być również odczytywane, istnieje możliwość testowania samego sterownika bez dołączania urządzenia zewnętrznego. W tabeli 4 przedstawiono wykaz linii sprzęgu równoległego sterownika drukarki wraz z przyporządkowaniem sygnałów do kontaktów złącza.

MICHAŁ KIRPLUK
PIOTR SOBOLEWSKI
Instytut Informacji
Naukowej, Technicznej i Ekonomicznej
Warszawa

Implementacja dedukcyjnej bazy danych Holmes (2)

W drugiej części artykułu przedstawiono szczegóły implementacyjne systemu Holmes.

IMPLEMENTACJA

Struktura systemu Holmes jest dwupoziomowa. Poziom wewnętrzny to program napisany w Micro-Prologu. Program ten stanowi jądro systemu zarządzające bazą danych i realizujące mechanizmy dedukcji. Wszelki kontakt jądra systemu z otoczeniem odbywa się przez drugi poziom. Poziom ten zaimplementowany w języku Turbo Pascal zapewnia sterowanie działaniem jądra, dostarczając mu wszystkich informacji niezbędnych do wykonania konkretnego polecenia. W tym celu na poziomie Pascala istnieje schemat bazy danych równoległy do schematu w Prologu. Funkcją poziomu Pascala jest też komunikacja z użytkownikiem. W trakcie konwersacji z użytkownikiem system uzyskuje od niego niezbędne informacje, odbiera polecenia, a po ich wykonaniu wysyła na ekran odpowiedzi w czytelnej postaci. Ogólną strukturę systemu pokazuje rysunek 1.

Zrealizowany system jest sterowany przez plik komend. Komendy wchodzące w skład pliku uruchamiają programy w języku Turbo Pascal oraz interpreter języka Micro-Prolog. Wywołania głównego programu w Pascalu i interpretera Prologu odbywają się w pętli, dla której warunkiem zakończenia jest brak tzw. pliku stanu. Plik stanu jest tworzony w momencie rozpoczęcia pracy z systemem i istnieje aż do jej zakończenia. Program główny w języku Pascal nadzoruje pracę poziomu Prologu. Wykorzystywane są przy tym duże możliwości Pascala w zakresie operacji plikowych i tekstowych. Za pośrednictwem plików program główny steruje poziomem Prologu. Przekazują one do Prologu trzy grupy informacji: wywołaną procedurę, postać formuły czytelną dla Prologu oraz ciąg nazw relacji i typów zbędnych w bieżącym wywodzie Prologu. Pliki są też używane w celu przesłania wyników działania programu prologowego do programu w Pascalu.

Struktura baz danych systemu

System Holmes służy do zarządzania dedukcyjną bazą danych. Jej cechą charakterystyczną jest oddzielenie właściwej bazy danych od schematu. Nakłada się na to podział na główną bazę danych i bazę danych użytkownika. Dodatkowo w omawianej implementacji część elementów schematu jest zawarta w programie pascalowym, a część — w programie prologowym. Właściwa baza danych znajduje się tylko w Prologu. Obraz dedukcyjnej bazy danych w systemie Holmes przedstawia rysunek 2.

Schemat bazy danych na poziomie Pascala zawiera słownik nazw wszystkich relacji, typów i zasad spójności, jak również ich pełny opis. Schemat ten będzie w dalszym ciągu nazywany schematem PA. Schemat na poziomie Prologu składa się z definicji relacji, typów i zasad spójności. Definicje te zawierające nazwy w postaci wewnętrznej są tworzone przez program w Pascalu.

Właściwa baza danych zawiera tylko obiekty i fakty. Ze względu na potrzebę efektywnego posługiwania się dwupoziomową strukturą bazy danych (poziom Pascala i Prologu) celowe stało się przechowywanie każdej nazwy relacji, typu, zasady spójności i zmiennej w trzech postaciach. Pierwszą postać stanowi nazwa podana przez użytkownika (w języku Holmes), drugą — niepowtarzalny numer wewnętrzny wygodny przy zarządzaniu schematem PA, zaś trzecią — nazwa wewnętrzna przekazywana do Prologu.

Poziom Pascala

Programy w języku Pascala realizują pięć grup zadań:

- komunikację systemu Holmes z użytkownikiem (przyjmowanie poleceń i udzielanie na nie odpowiedzi),
- zarządzanie plikami użytkowymi,
- tworzenie i utrzymywanie schematu bazy danych, równolegle do istniejącego w Prologu (schemat istniejący na poziomie Pascala umożliwia analizę syntaktyczną i semantyczną poleceń kierowanych do systemu, jak również optymalizację pracy Prologu dzięki selekcji nazw dostarczanych do wywodu),
- przetwarzanie schematu oraz realizacja komend, wraz z formułowaniem odpowiednich zadań dla programu w Prologu,
- translację formuł w języku Holmes na formuły akceptowalne przez interpreter Prologu.

Jak już wspomniano, programy w Pascalu i Prologu są naprzemian ładowane do pamięci. W związku z tym pojawia się konieczność pamiętania większości informacji w plikach. Duża liczba plików roboczych zwiększa znaczenie sprawnego zarządzania nimi. Operowanie plikami można podzielić na trzy fazy:

1) stworzenie wersji roboczych z plików źródłowych (pliki źródłowe są wybierane na podstawie nazw podanych przez użytkownika);
2) modyfikowanie wersji roboczych zgodnie z realizowanymi komendami, przy czym program pascalowy operuje tylko swoimi plikami, a prologowy swoimi;
3) zapamiętanie wersji roboczych po zakończeniu sesji na dysku użytkownika (wersje robocze są zapamiętywane w

Mgr inż. MICHAŁ KIRPLUK ukończył w 1987 r. studia na Wydziale Elektroniki Politechniki Warszawskiej w Instytucie Informatyki. Obecnie pracuje w Instytucie Informacji Naukowej Technicznej i Ekonomicznej. Zajmuje się problematyką związaną z systemami dedukcyjnych baz danych.

Mgr inż. PIOTR SOBOLEWSKI ukończył w 1987 r. studia na Wydziale Elektroniki Politechniki Warszawskiej w Instytucie Informatyki. Obecnie pracuje w Instytucie Informacji Naukowej Technicznej i Ekonomicznej. Zajmuje się problematyką związaną z systemami dedukcyjnych baz danych i programowaniem w języku logiki.

plikach o nazwach odpowiadających im plików źródłowych).

Na podstawie roboczych wersji plików Pascala tworzone są struktury listowe, co umożliwia sprawniejszy dostęp do danych niż w wypadku plików. Struktury te zawierają postać wewnętrzną schematu PA. Wśród nich można wyróżnić następujące trzy odrębne grupy:

- listy zawierające słowniki wszelkich nazw znanych systemowi;
- listy opisujące szczegółowo poszczególne nazwy (o ile jest to konieczne);
- listy zawierające definicje typów, relacji i zasad spójności przeznaczone do edycji.

W wyniku interpretacji schematu powstają, zarówno w Pascalu jak i w Prologu, jego postacie wewnętrzne. W trakcie realizacji komend są one odpowiednio analizowane bądź modyfikowane.

Zadanie, które stoi przed programem w Pascalu jest następujące: stworzyć na podstawie linii schematu (lub komendy) oraz informacji zawartych w strukturach listowych schematu PA polecenie dla Prologu. Pomimo różnej składni interpretacja linii schematu jak i komend przebiega podobnie. Wspólnym elementem jest ta sama w obu wypadkach postać formuł oraz plików wejściowych generowanych dla Prologu. Schemat ogólny znajduje się też na poziomie Prologu. Występuje on tam jako zbiór definicji typów i relacji pochodnych (ang. derived relationship) oraz zasad spójności. Jednak postać definicji w języku Holmes dalece odbiega od tej, która może być zaakceptowana przez interpreter Prologu. Dlatego drugim etapem przetwarzania schematu jest translacja formuł stanowiących treść definicji. Przykładowo, postać wewnętrzna definicji relacji pochodnej

JEST_STACJA_DOCELOWA_DLA_POCIĄGU

z pierwszej części artykułu jest następująca:

((ABBA X Y))

(AND ((ABAG Y Z X)) ((Not ((EXIS ((ABAC X)) ((AND ((AND ((ABAG Y y x)) ((ABAG Y Z X)) ((GREA y Z)))))))))

Jak łatwo zauważyć, ABBA jest nazwą wewnętrzną nazwy STACJA_DOCELOWA, ABAG jest nazwą wewnętrzną nazwy relacji JEST_STACJA_DOCELOWA_DLA_POCIĄGU, itd.

Wykonanie większości komend składa się z czterech faz:

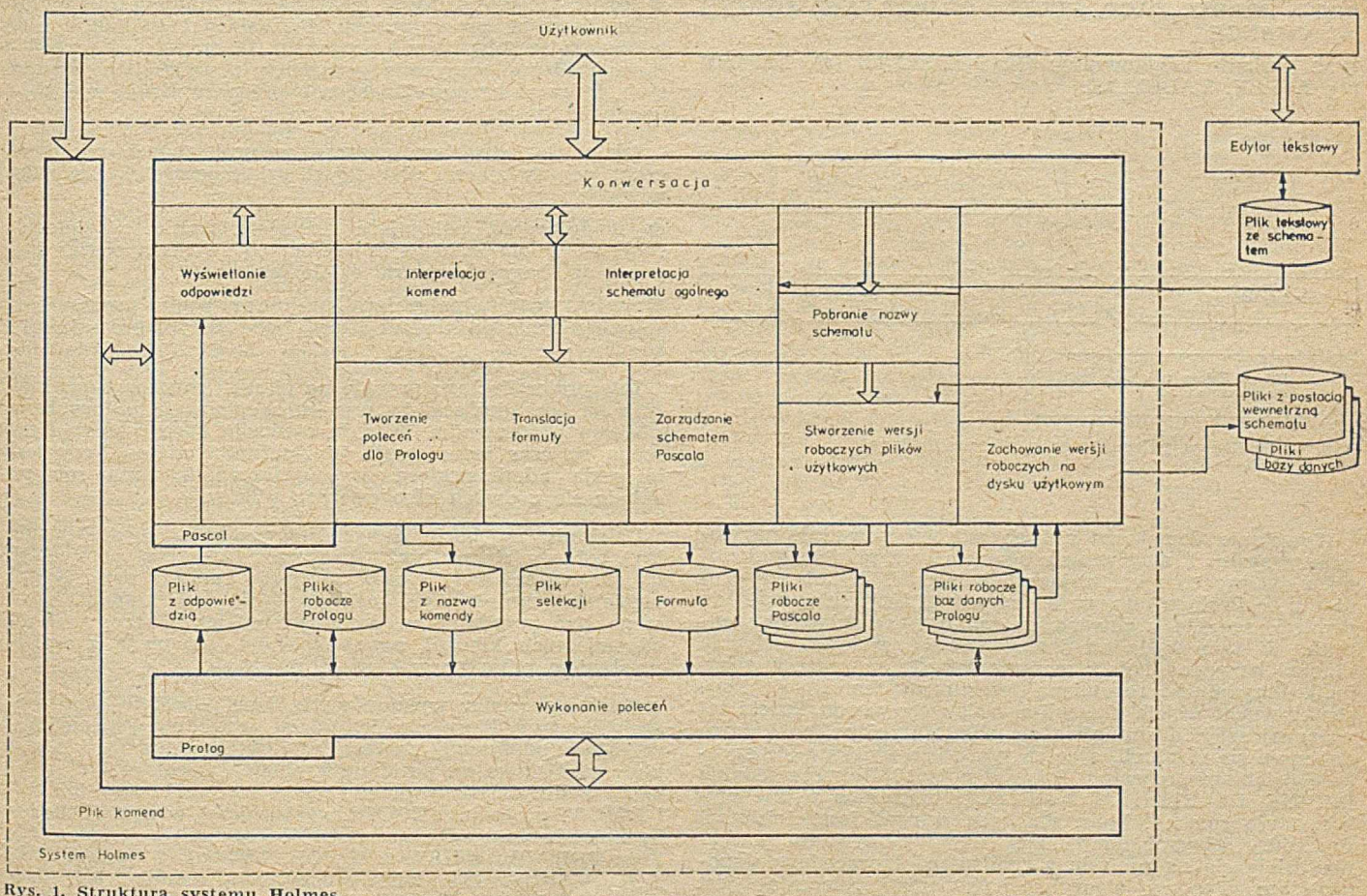
1) rozpoznania komendy i zrealizowania czynności związanych ze schematem PA,
2) wygenerowania polecenia dla interpretera Prologu,
3) zasadniczego wykonania komendy przez interpreter Prologu,
4) interpretacji wyników.

Polecenie dla interpretera Prologu jest przekazywane za pomocą trzech plików, zawierających następujące informacje: nazwę polecenia, formułę w postaci prologowej oraz nazwy relacji, typów i zasad spójności zbędnych w prowadzonym wywodzie.

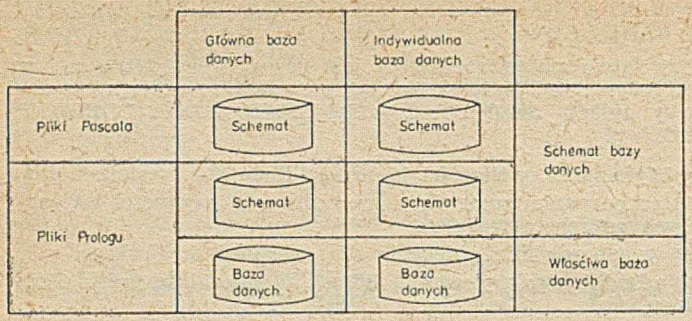
Część komend systemu jest wykonywana tylko na poziomie Pascala. Do komend tych należą: SUSPEND, ACTIVATE, LIST NAMES i LIST DEFINITIONS. W tych wypadkach nie dochodzi do wywołania interpretera Prologu, nie są tworzone pliki wejściowe, brak też plików odpowiedzi. Program pascalowy samodzielnie tworzy odpowiedź dla użytkownika na podstawie schematu PA. Do zrealizowania pozostałych komend konieczne jest wywołanie interpretera Prologu. Końcowa odpowiedź jest wtedy tworzona na podstawie plików odpowiedzi Prologu.

Oprócz błędów wykonania, dla każdej komendy mogą wystąpić błędy translacji, które są wykrywane przed rozpoczęciem wykonywania komendy. W wypadku tych błędów na ekranie pojawia się informacja o miejscu wystąpienia błędu i uwagi ułatwiające jego wyeliminowanie. System umożliwia poprawienie komendy przez jej powtórne zredagowanie.

Zasadniczym składnikiem większości komend systemu Holmes jest formuła. Zadaniem etapu translacji formuły



Rys. 1. Struktura systemu Holmes



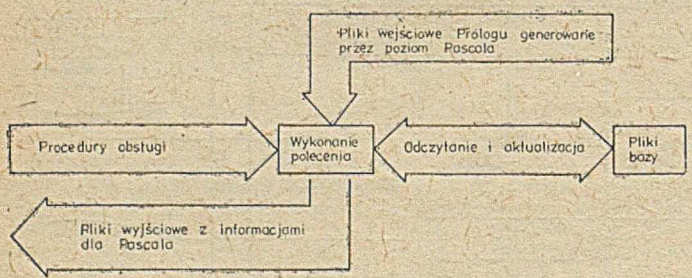
Rys. 2. Struktura wewnętrzna bazy danych systemu Holmes

jest przekształcenie postaci widocznej dla użytkownika na postać akceptowalną dla Prologu. W trakcie tego procesu tworzone są trzy postacie pośrednie. W zaimplementowanej metodzie translacji języka formuł systemu Holmes występują cztery klasyczne fazy translacji, przy czym fazy analizy syntaktycznej i semantycznej praktycznie są ze sobą połączone. Jest to spowodowane stosunkowo niewielkim zakresem analizy semantycznej, co wynika stąd, że badanie poprawności logicznej wykonywane jest przez interpreter Prologu. W translacji formuł dużą rolę odgrywa przygotowanie do generowania przekładu. Ponieważ docelowa postać formuły dla każdej komendy czy też linii schematu jest wysyłana do interpretera Prologu oddzielnie, nie istnieje potrzeba pamiętania ciągu postaci wewnętrznych ani ciągu ich postaci docelowych. Program w Pascalu pamięta tylko postać wewnętrzną bieżącej formuły. Natomiast postacie docelowe wszystkich formuł przechowuje w swoich zbiorach interpreter Prologu.

Poziom Prologu

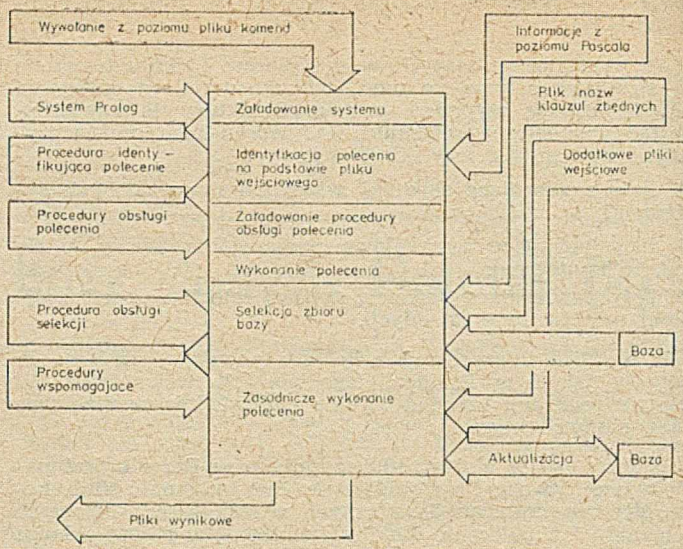
Każde polecenie systemu Holmes wykonujące operacje na bazie danych jest zaimplementowane w języku Micro-Prolog. Ogólny schemat realizacji poleceń Holmesa na poziomie Prologu przedstawia rysunek 3. Do zbioru tych poleceń zalicza się:

- z poziomu administratora: APPLY, VALIDATE, ADD FACT, DELETE FACT, DELETE OBJECT, ADD OBJECT (*), CHANGE
- z poziomu użytkownika: FIND, DELETE (*), DEFINE (*), APPLY, VALIDATE.



Rys. 3. Realizacja polecenia na poziomie Prologu

Wywołanie dowolnego z tych poleceń powoduje wywołanie systemu Prolog wraz z odpowiednim plikiem wejściowym dla procedur realizujących polecenie. Wykonanie programu w Prologu odbywa się na części bazy ściągniętej do pamięci, zgodnie z zawartością pliku nazw klauzul zbędnych. W tym wypadku jest stosowana wstępna selekcja klauzul bazy pod kątem ich przydatności w procesie rezolucji. Przydatność rozumiana jest tu jako możliwość konstruktywnego użycia klauzuli w drzewie wywodu. Dzięki tej wstępnej selekcji zmniejsza się objętość zbioru, na którym dokonuje się rezolucji, a co za tym idzie zwiększa się szybkość działania programu. Spośród poleceń Holmesa jedynie DEFINE, DELETE i ADD OBJECT (oznaczone *) nie wykorzystują selekcji, która dla tych poleceń jest nieistotna. Procedury ściągane do pamięci przy wykonywaniu polecenia dzielą się na procedury realizujące i procedury wspomagające. Typowy algorytm wykonania polecenia jest przedstawiony na rysunku 4.



Rys. 4. Algorytm wykonania polecenia

Wykonanie polecenia zaczyna się od załadowania systemu Prolog, a następnie procedury identyfikującej wykonywane polecenie, która z kolei ściąga do pamięci odpowiednią procedurę obsługi i rozpoczyna jej wykonanie. W zależności od rodzaju polecenia, wykonywana jest selekcja danych z bazy bądź nie. Zasadnicza część polecenia jest wykonywana dopiero wtedy, gdy w pamięci znajdzie się odpowiednia część bazy. Większość poleceń korzysta w tym momencie z dodatkowych procedur wspomagających oraz z plików wejściowych. Wykonanie poleceń kończy się aktualizacją bazy i wygenerowaniem plików wyjściowych z wynikami i informacjami dla poziomu Pascala.

Program realizujący polecenie składa się zazwyczaj z jednej lub dwóch procedur, najczęściej w postaci modułu. Moduł dzieli de facto pamięć na dwie części: część czynną — czyli moduł z procedurą, i część bierną — czyli bazę ściągniętą do pamięci. Taki podział ułatwia manipulowanie bazą w prosty sposób. Pozwala też uniknąć przemieszania ze sobą bazy i programu. W moduły są zorganizowane niemal wszystkie programy, z wyjątkiem jedynie tych, które w bezpośredni sposób przeprowadzają wnioskowanie na bazie.

Jak widać, struktura programów prologowych ma dwie cechy poprawiające i ułatwiające działanie systemu, a mianowicie:

- **modularność,** polegająca na zastosowaniu oddzielnych programów dla każdego polecenia, co zwiększa obszar pamięci dla bazy i szybkość działania,
- **selektywność,** polegającą na ograniczeniu procesu rezolucji jedynie do klauzul, które mogą w tym procesie wystąpić.

Interpreter Prologu pracuje na plikach roboczych powstałych na początku sesji w wyniku skopiowania plików bazy z dysku użytkownika. Pliki te stanowią bazę roboczą tworzoną na czas jednej sesji. W zależności od tego, czy jest to sesja administratora czy użytkownika, baza robocza składa się z dwóch albo czterech plików.

Działanie na bazie danych (dowodzenie, wyszukiwanie itp.) odbywa się na części bazy rezydującej w pamięci tj. na obrazie tej bazy. Obraz ten jest tworzony przez ściągnięcie do pamięci z plików bazy tylko tych klauzul, które mogą być przydatne w procesie rezolucji. Proces wyboru odpowiednich klauzul zwany selekcją, przebiega z wykorzystaniem pliku nazw klauzul zbędnych.

Oprócz procedur wykonujących dane polecenie, w procesie wykonania biorą również udział procedury wspomagające, które odgrywają znaczną rolę w implementacji logiki systemu Holmes na poziomie Prologu. Do zbioru tych procedur zaliczyć należy przede wszystkim procedury TOOLS i ERR. W procedurze TOOLS, występującej w postaci modułu, zaimplementowane są następujące operatory systemu Holmes: FORALL, EXIST, AND, OR, LESS, LE, GREAT, GE, NOT, IMPLY, IFF. Listą eksportową modułu jest lista symboli relacji i kwantyfikatorów, w takiej formie, jaką

generuje poziom Pascala, gdy tworzy formułę wejściową dla Prologu. Przy implementacji poszczególnych operatorów wykorzystuje się standardowe relacje Micro-Prologu, takie jak EQ, NOT i LESS.

Inną ważną z punktu widzenia systemu Holmes procedurą jest ERR. Obsługuje ona programowo wystąpienie błędu w programie prologowym. Istnienie tej procedury jest dla systemu bardzo ważne, gdyż umożliwia wyjście z poziomu Prologu do systemu operacyjnego, np. DOS, w wypadku wystąpienia błędu. Procedura ERR spełnia także drugą, jeszcze ważniejszą rolę, a mianowicie zapewnia logiczną spoistość całej bazy prologowej. Otóż baza na poziomie Prologu nie zawiera deklaracji typów i relacji pierwotnych. Istnienie tych deklaracji w bazie równałoby się istnieniu reprezentowanych przez te deklaracje obiektów pustych. Nastręczałoby to wiele kłopotów podczas wykonywania poleceń aktualizujących zbiór faktów. Program musiałby rozpoznawać obiekt pusty i traktować go w sposób szczególny. Takiego obiektu nie można by na przykład usunąć za pomocą polecenia DELETE OBJECT. W systemie zastosowano prostsze rozwiązanie. Zamiast programować istnienie obiektu pustego w wielu procedurach realizujących różne polecenia, zaprogramowano możliwość wystąpienia błędu w drzewie wywodu podczas wywołanie nie istniejącego typu czy relacji, co pozwoliło w ogóle zrezygnować z istnienia obiektów pustych.

* * *

Zrealizowany system można traktować jedynie w kategoriach wersji prototypowej. Ze względu na ograniczenie, tak szybkości działania jak i wielkości bazy, może on być wykorzystany jedynie w celach eksperymentalnych. Podstawowe ograniczenia wynikają z przyjęcia dla realizacji procesów dedukcji tak słabego narzędzia jakim jest język Micro-Prolog. Język ten, pierwotnie zaimplementowany na procesorze Z80 dla systemu operacyjnego CP/M, został następnie przeniesiony na mikrokomputer klasy IBM PC. Pamięć, którą zajmuje system Prolog, stos, stóg oraz sam program jest ograniczona do 64 KB.

Celem realizacji systemu Holmes było przede wszystkim zbadanie możliwości implementacji tego typu systemu na mikrokomputerze. Dzięki modułowej budowie (dwa poziomy) istnieje możliwość wymiany jądra logicznego systemu (poziom Prologu) na inny, np. Turbo Prolog, który niewątpliwie umożliwiłby zrealizowanie doskonalszego systemu, tak pod względem szybkości działania (co najmniej o rząd wielkości szybszego) jak też wielkości bazy.

Możliwość uzyskania lepszej implementacji systemu Holmes na podstawie języka Prolog zależy od dwóch czynników: oprogramowania i sprzętu.

Jeżeli chodzi o oprogramowanie, to ewentualnej poprawy możliwości systemu można oczekiwać przede wszystkim w związku z zastosowaniem doskonalszej wersji samego języka. Dobrym kandydatem byłby język Turbo Prolog. Pozwoliłby on zrezygnować z użycia Pascala, przez co konstrukcja całego systemu stałaby się bardziej klarowna. Szczególnie cenny byłby bogaty zestaw poleceń Turbo Prologu do grafiki, operacji dyskowych i redakcyjnych. W tym zakresie możliwości Turbo Prologu pod pewnymi względami przewyższają odpowiednie możliwości języka Turbo Pascal [3].

Zastąpienie poziomu Pascala przez silny Prolog dałoby zysk programistyczny (mniej kodu i łatwość jego tworzenia) oraz użytkowy (łatwiejsza komunikacja z użytkownikiem), a także pozwoliłoby uzyskać łagodne i naturalne przejście między sprzężeniem użytkowym a jądrem logicznym systemu. Zastosowanie Turbo-Prologu zwiększyłoby również szybkość działania systemu.

Dla systemów dedukcyjnych problem szybkości działania jest niezwykle ważny. Szybkość obliczeniową systemów dedukcyjnych określa się w jednostkach zwanych LIPS (ang. Logical Inferences Per Second) [2]. Jednostka LIPS jest równa liczbie przejść przez węzeł w drzewie wywodu na sekundę. Przejście przez węzeł w drzewie wywodu, określane mianem konkluzji logicznej, jest równoznaczne pojedynczemu zastosowaniu zasady rezolucji na parze klauzul. Dla języka Micro-Prolog na mikroprocesorze Z80 szybkość obliczeniowa systemów dedukcyjnych wynosi około 120 LIPS [1]. W wypadku komputera klasy IBM PC (procesor 8088), szybkość tę ocenić można maksymalnie na około 290 LIPS. Turbo Prolog jest około 10-krotnie szybszy od Micro-Prologu, przy czym porównanie dotyczy standardowych mechanizmów działania, tzn. dla Micro-Prologu — interpretacji w systemie, a dla Turbo Prologu konwersacji z uruchomionym programem (skompilowanym). Dla porównania, szybkość interpretera Prologu dla minikomputerów serii VAX-11/780 mieści się w zakresie 1500—15 000 LIPS, a dla dużych komputerów firmy IBM w granicach setek tysięcy LIPS [1]. Możliwość dalszego zwiększenia szybkości działania upatruje się w zrównolegleniu procesu rezolucji. W takim kierunku prowadzone są prace przy projekcie komputerów piątej generacji. Równoległe wykonywanie rezolucji można sobie wyobrazić jako przetwarzanie przez oddzielne procesy różnych gałęzi drzewa wywodu. Stopień zwiększenia szybkości działania systemu byłby równy co najmniej wielokrotności liczby użytych procesów.

Oczywiście, realizacja równoległego wykonywania rezolucji miałaby sens jedynie w układzie wieloprocesorowym. Biorąc pod uwagę tendencje rozwojowe sprzętu, większych możliwości należałoby upatrywać w specjalizowanych procesorach języków wysokiego poziomu, w wypadku systemu Holmes — języka Prolog. Przykładem takiego procesora jest procesor firmy Xenologic Inc., który działając w systemie VMEbus, w charakterze procesora podległego, pozwala osiągać szybkość rzędu 200 000—300 000 LIPS [1]. Jest to wielkość porównywalna z szybkościami uzyskiwanymi na komputerach o największych mocach obliczeniowych, a trzeba zaznaczyć, że omawiany sprzęt mieści się w klasie mikrokomputerów.

Następnym etapem w zwiększeniu szybkości działania systemów dedukcyjnych byłoby zastosowanie pamięci asocjacyjnych. Proces rezolucji idealnie wprost nadaje się do wykorzystania pamięci asocjacyjnej. Obecnie w wielu laboratoriach na świecie prowadzi się badania nad tymi pamięciami i wydają się one być o krok od wdrożenia.

LITERATURA

[1] Cole C.: A pride of new CPU-s runs high-level languages. Electronics Week, 25 November 1985
[2] Colmerauer A.: Prolog in 10 figures. Communications of the ACM, Vol. 28, No. 12, 1985
[3] Turbo Prolog Reference Manual. Borland International Inc., 1986.

KONFERENCJE

INFOPROD '88

Oddział Wojewódzki Polskiego Towarzystwa Ekonomicznego w Bydgoszczy, wspólnie z Instytutem Badań Systemowych PAN, Instytutem Organizacji Przemysłu Maszynowego „Orgmasz", Zakładem Informatyki Stosowanej Akademii Techniczno-Rolniczej w Bydgoszczy — organizują w **Ciechocinku w dniach 5—10 września 1988 r.**, a w wypadku dużej liczby zgłoszeń w dniach: **5—7 września 1988 r. (pierwszy termin) i 8—10 września 1988 r. (drugi termin)** ogólnopolską konferencję naukową pn. **„Techniki komputerowe w zarządzaniu produkcją — INFOPROD '88".**

Celem konferencji jest ocena zastosowań komputerów, mini- i mikrokomputerów w zarządzaniu produkcją oraz przedyskutowanie i wytyczenie obszarów i metod dalszych ich wdrożeń, w tym w szczególności:

- zastosowanie komputera R-34 oraz ME-29 w zarządzaniu produkcją,
- zaprezentowanie przykładów praktycznych zastosowań minikomputerów SM-4 i MERA 9150 we współpracy z dużymi komputerami,
- prezentacja możliwości i wykorzystania mikrokomputerów IBM PC/XT i AT oraz MERA 660,
- prezentacja sprzętu i oprogramowania (wystawa sprzętu),
- przedyskutowanie koncepcji zmian w metodach sterowania produkcją przez szefów produkcji, z wykorzystaniem sprzętu komputerowego, mini- i mikrokomputerów.

Konferencja jest adresowana głównie do dyrektorów produkcji i dyrektorów ekonomicznych przedsiębiorstw, szefów produkcji przedsiębiorstw, kierowników wydziałów produkcyjnych oraz służb: kompletacyjnych i planowania produkcji, gospodarki materiałowej, ekonomicznych oraz informatycznych.

Przewidziany koszt uczestnictwa wynosi 26 900 zł i obejmuje: komplet referatów i materiały pokonferencyjne, zakwaterowanie (2 noclegi), wyżywienie (3 dni) w ramach diet.

Zgłoszenia prosimy kierować pod adresem:
Polskie Towarzystwo Ekonomiczne, Oddział Wojewódzki
ul. Długa 34, 85-034 Bydgoszcz
tel. 22-32-55 lub 22-67-81.

JAN BIELECKI
Instytut Informatyki
Politechnika Warszawska

Język C

Coraz bliżej normy (I)

Utrzymujące się zainteresowanie językiem C, profesjonalnym narzędziem wykorzystywanym do programowania mikrokomputerów, zachęciło amerykański instytut ANSI do powołania grupy roboczej, której zadaniem jest opracowanie normy tego języka. Wiosną 1986 roku opublikowano projekt takiej normy. Zakończenie prac normalizacyjnych jest planowane na rok 1988. Mimo to, pojawiły się już na rynku kompilatory zgodne z projektem. Dwa najważniejsze z nich — to Turbo C (firmy Borland) i Quick C (firmy Microsoft). Niniejsze opracowanie zawiera krótki opis wybranych elementów ANSI C.

Słowa kluczowe

Zdefiniowano następujące nowe słowa kluczowe: const, enum, signed, void, volatile. Ich interpretacja zostanie podana dalej.

```
#include <stdio.h>
enum{Iza = 10,Ewa};
const volatile signed char Fix = —8;
main ()
{
  (void)printf("%d", Iza + Fix + Ewa);
}
```

Wykonanie programu powoduje wyprowadzenie liczby 13.

Identyfikatory

Co najmniej 31 znaków identyfikatora jest istotnych. Wyjątek stanowią identyfikatory międzymodułowe, w których istotnych jest co najmniej 6 znaków. Dokładne liczby znaków istotnych, z uwzględnieniem wymienionych wymagań, są określone przez implementację.

```
#include <stdio.h>
main()
{
  OutputAllDigits (1.0 / 3);
}
```

Jeśli w pewnej implementacji ograniczono liczbę znaków identyfikatorów międzymodułowych do 9, a litery duże i małe są uznawane za różne, to przytoczony program będzie kompletny, jeśli zostanie skonsolidowany, na przykład, z funkcją OutputAllPrecision.

Przestrzenie nazw identyfikatorów

Jeśli w pewnym punkcie programu są widoczne deklaracje kilku identyfikatorów, to identyfikatory te mogą być takie same, jeśli należą do różnych przestrzeni nazw. Ogółem zdefiniowano 4 rodzaje przestrzeni nazw: przestrzeń etykiet, przestrzeń oznaczników, przestrzeń pól struktur i unii, przestrzeń pozostałych identyfikatorów. W odróżnieniu od wersji podstawowej języka przyjęto, że przestrzenie pól są lokalne w obrębie deklaracji struktury albo unii, co oznacza, że przemieszczenia pól o tej samej nazwie nie muszą być identyczne.

```
#include <stdio.h>
struct Str{
  char Str;
} Str = { 'E' } ;
main()
{
  Str:
  putchar(Str.Str);
}
```

Wykonanie programu powoduje wyprowadzenie litery E. W miejscu wywołania funkcji putchar identyfikator Str jest etykietą, oznacznikiem, nazwą struktury i polem struktury.

Literały znakowe i napisowe

W obrębie literału znakowego i napisowego może wystąpić opis znaku /a*). Jego zinterpretowanie powoduje wygenerowanie sygnału dźwiękowego. W literałach mogą wystąpić także opisy znaków o postaci /xh w której h jest ciągiem od jednej do trzech cyfr szesnastkowych. Ponadto przyjmuje się, że dwa następujące po sobie literały napisowe są równoważne literałowi składającemu się ze wszystkich znaków pierwszego literału i następujących bezpośrednio po nich wszystkich znaków drugiego.

```
#include <stdio.h>
main()
{
  printf("Ring"
         " a /a"
         "bell");
}
```

Wykonanie programu powoduje wyprowadzenie napisu:

Ring a bell

W trakcie wyprowadzania tego napisu rozlegnie się sygnał dźwiękowy.

Literały stałopozycyjne

Literał stałopozycyjny zakończony (małą lub dużą) literą U jest traktowany tak, jakby jego nazwa typu zawierała modyfikator unsigned. Taki literał zakończony (małą albo dużą) literą L jest traktowany tak, jakby jego nazwa typu zawierała modyfikator long.

Przykład

Literał	Typ
5	int
5u	unsigned int
5L	long int
5ul	unsigned long int

Literały zmiennopozycyjne

Literał zmiennopozycyjny zakończony (małą albo dużą) literą F jest typu (float). Taki literał zakończony (małą albo dużą) literą L jest typu (long double). Pozostałe literały zmiennopozycyjne są typu (double).

Przykład

Literał	Typ
4.5	double
4.5f	float
5e2L	long double

*) Przypominamy, że z przyczyn technicznych zamiast ukośnika używamy w INFORMATYCE znaku dzielenia (przyp. red.).

Nazwy typów

Słowa kluczowe signed, unsigned, short i long mogą być nie tylko używane jako przymiotniki, ale również jako nazwy typu. W takim wypadku domniemywa się oznaczenie typu int. Oznaczenie to domniemywa się także w wypadku użycia modyfikatorów const i volatile.

```
#include <stdio.h>
main()
}
  unsigned const Fix = 13;
  printf("%u", Fix);
{
```

Zmienna Fix jest typu (unsigned const int). Wykonanie programu powoduje wyprowadzenie liczby 13.

Prototypy

Prototypem jest deklarator funkcji, w którym jawnie wyszczególniono typy parametrów funkcji. Każdy z typów musi mieć postać nazwy typu albo postać nazwy typu zawierającej identyfikator parametru. Jeśli pewne określenie typu nie zawiera identyfikatora, to i pozostałe nie mogą go zawierać.

```
#include <stdio.h>
char Arr[3] = "Jan";
main()
{
  char * fun(char *One, int Two);
  printf("%c", *fun(Arr, 2.9));
}
char *
fun(char *Ptr,int Num)
{
  return Ptr + Num;
}
```

Wykonanie programu powoduje wyprowadzenie litery n.

Ponieważ posłużono się prototypem:

fun(char *One, int Two)

argument 2.9 zostanie poddany niejawnej konwersji na argument 2.

Modyfikatory signed i unsigned

Oznaczenie typu char, int i long int może wystąpić wraz z modyfikatorem signed albo unsigned. O tym, czy dane typu (char) są w istocie typu (signed char) czy (unsigned char) decyduje implementacja.

```
#include <stdio.h>
main()
{
  signed char eof = EOF;
  printf("%d", eof);
}
```

Wykonanie programu powoduje wyprowadzenie liczby ujemnej.

Modyfikatory const i volatile

Zmienna zadeklarowana z modyfikatorem const zachowuje się jak stała. Takiej zmiennej można przypisać daną jedynie podczas opracowywania inicjatora. Nie wyklucza to jednak możliwości przypisania jej danej przez czynnik znajdujący się poza programem.

Zmienna zadeklarowana z modyfikatorem volatile ma tę właściwość, że przypisywanie jej danych może zachodzić w sposób nie kontrolowany przez implementację.

Zmienna zadeklarowana z modyfikatorem const, ale nie zadeklarowana z modyfikatorem volatile, której inicjator jest wyrażeniem stałym, może być przechowywana w pamięci ROM.

```
#include <stdio.h>
main( )
{
  const unsigned Fix = 13;
  printf("%d", Fix);
}
```

Wykonanie programu powoduje wyprowadzenie liczby 13.

Typ long double

Typ (long double) ma się tak do typu (double) jak typ (double) do typu (float). Zbiór wartości danych związanych z typem (double) jest podzbiorem wartości danych związanych z typem (long double). Poprawne są implementacje, w których typy (float), (double) i (long double) są identyczne.

```
#include <stdio.h>
main()
main()
  printf("%d %d %d", sizeof(float),
                     sizeof(double),
                     sizeof(long double);
}
```

Wykonanie programu powoduje wyprowadzenie liczb określających rozmiary zmiennych typu (float), (double) i (long double).

Typy wyliczeniowe

Specyfikacja typu wyliczeniowego ma postać:

enum oznacznik { lista-wyliczeniowa }

albo

enum oznacznik

w której oznacznik jest identyfikatorem, a elementami listy wyliczeniowej są identyfikatory albo przypisania o postaci:

identyfikator = wyrażenie-stałe

Opracowanie przytoczonej specyfikacji powoduje utworzenie typu (enum oznacznik) związanego ze zbiorem danych typu (int), które w programie są identyfikowane przez identyfikatory listy wyliczeniowej. Jeśli element listy wyliczeniowej zawiera wyrażenie stałe, to następuje związanie identyfikatora z daną o wartości tego wyrażenia. W przeciwnym razie następuje związanie go z daną o wartości o jeden większej niż wartość danej, z którą związano poprzedni identyfikator, a jeśli jest to pierwszy identyfikator listy, to następuje związanie go z daną o wartości 0.

```
#include <stdio.h>
enum tag{ First,Second = 3,Third };
main()
{
  enum tag Fix = Third;
  printf("%d", Fix + Second);
}
```

Wykonanie programu powoduje wyprowadzenie liczby 7. First, Second i Third są nazwami literałów reprezentujących dane o wartościach 0, 3 i 4. Zmiennej Fix typu (enum tag) można przypisywać jedynie dane całkowite o tych wartościach.

Język Smalltalk-80

dokończenie ze s. 7

LITERATURA

[1] BYTE, Vol. 6, No. 8, August 1981, specjalny numer poświęcony systemowi Smalltalk-80

[2] Goldberg A., Robson D.: Smalltalk-80 — the Language and its Implementation. Addison-Wesley, Reading (Mass.), 1983

[3] Goldberg A.: Smalltalk-88 — the Interactive Programming Environment, Addison-Wesley, Reading (Mass.), 1983

[4] Krasner G.: Smalltalk-80 — Bits of History, Words of Advice, Addison-Wesley, Reading (Mass.), 1984

[5] Krępski A., Kopeć B.: Ocena Systemu Smalltalk-80 jako bazy do budowy warsztatu programisty. Raport roboczy, Instytut Podstaw Informatyki PAN, Warszawa, 1986

[6] Oktaba H., Ratajczak W.: Simula 67, WNT, Warszawa, 1980

[7] Smalltalk/V. Tutorial and Programming Handbook. Digitalk Inc., Los Angeles (California), 1986.

ANNA SYFERT
ANDRZEJ RAZNOWIECKI
oprac.
Warszawa

Struktura systemu operacyjnego PC-DOS (6)

W niniejszej, ostatniej części artykułu opisano funkcje zarządzania pamięcią i programem oraz pozostałe funkcje dotyczące plików. Użycie funkcji zilustrowano przykładowym programem.

FUNKCJE ZARZĄDZANIA PAMIĘCIĄ I PROGRAMEM

Funkcja 48H — alokacja pamięci

Powyższa funkcja przydziela obszar pamięci, przy czym jego wielkość jest podawana w tzw. paragrafach (po 16 bajtów).

Parametry wejściowe:
AH — 48H
BX — liczba rezerwowanych paragrafów
Parametr wyjściowy (bit przeniesienia nieustawiony):
AX — adres segmentu, w którym od wyrównania 0 zaczyna się przydzielany obszar pamięci
Komunikaty błędów (przeniesienie ustawione):
AX — 07H, jeśli jakiś program zapisze obszar pamięci, który jemu nie został przyporządkowany
AX — 08H, jeśli nie występuje spójny obszar pamięci o zadanej wielkości
BX — liczba paragrafów występujących w największym rozpoznawalnym spójnym obszarze pamięci.

Funkcja 49H — dealokacja pamięci

Powyższa funkcja zwalnia obszar pamięci przydzielony za pomocą funkcji 48H.

Parametry wejściowe:
AH — 49H
ES:0 — adres początkowy zwalnianego obszaru pamięci (wielkość obszaru jest znana systemowi operacyjnemu)
Komunikaty błędów (przeniesienie ustawione):
AX — 07H, jeśli został zniszczony blok zarządzania pamięcią, np. jakiś program zapisał obszar pamięci, który nie został jemu przydzielony
AX — 09H, jeśli znaleziony obszar pamięci nie został zarezerwowany przy użyciu funkcji 48H

Funkcja 4AH — zmiana wielkości przydzielonej pamięci

Powyższa funkcja umożliwia zmianę wielkości obszaru pamięci już zarezerwowanego.

Parametry wejściowe:
AH — 4AH
BX — liczba rezerwowanych paragrafów
ES:0 — adres początkowy przydzielonego obszaru pamięci
Komunikaty błędów (przeniesienie ustawione):
AX — 07H, jeśli jakiś program zapisał obszar pamięci, który nie został jemu przyporządkowany
AX — 08H, jeśli nie występuje spójny obszar pamięci o zadanej wielkości
AX — 09H, jeśli przydzielony obszar pamięci nie został zarezerwowany przy użyciu funkcji 48H
BX — maksymalna liczba paragrafów, jakimi można rozporządzać w odnośnym obszarze pamięci.

Funkcja 4BH — ładowanie i wykonanie programu

Powyższa funkcja niszczy zawartość wszystkich rejestrów. Rejestry muszą zostać przechowane w komórkach pamięci nieosiągalnych przez program wywołany. Należy pamiętać o tym, że również zawartości rejestru segmentowego stosu SS oraz wskaźnika stosu zostaną zniszczone.
Parametry wejściowe:
AH — 4BH
AL — 00H, ładowanie i wykonanie programu
AL — 03H, ładowanie programu (w ogólności — nakładki)
ES:BX — adres bloku parametrów (patrz poniżej)
DS:DX — adres nazwy pliku zakończonej przez 00H; w nazwie może być zawarta ścieżka dostępu
Komunikaty błędów (przeniesienie ustawione):
AX — 01H, jeśli AL zawiera nieznany kod funkcyjny

```
SEGMENT 'CODE'

ASSUME  CS:CODE, DS:CODE, ES:NOTHING, SS:STACK

BELL            EQU     07H     ;SYGNAL AKUSTYCZNY
CHANGE_MODE     EQU     43H     ;DOS:ODCZYTANIE/USTAWIEN
                                ;ATRYBUTOW
CON_OUT_HANDLE  EQU     01H     ;KANAL ZDEFINIOWANY PRZEZ
                                ;MSDOS DLA WYPROWADZANIA NA
                                ;KONSOLE
CR              EQU     0DH     ;POWROT KARETKI
DOS             EQU     21H     ;PRZERWANIE DLA WYWOLANIA
                                ;FUNKCJI
DUPLICATE       EQU     45H     ;FUNKCJA DOS:ZALOZENIE DRUGIEGO
                                ;KANALU
FALSE           EQU     00H
FIND_FIRST      EQU     4EH     ;FUNKCJA DOS:SZUKANIE PIERWSZE-
                                ;GO WEJSCIA DANEGO ZBIORU
FIND_NEXT       EQU     4FH     ;SZUKANIE NASTEPNEGO WEJSCIA
                                ;(ENTRY)
FORCE_HANDLE    EQU     46H     ;FUNKCJA DOS:PRZELACZENIE KANALU
GET_DTA         EQU     2FH     ;FUNKCJA DOS:ODCZYTANIE ADRESU
                                ;DTA
LF              EQU     0AH     ;LINEFEED
PARSE           EQU     29H     ;FUNKCJA DOS:SZUKANIE NAZWY
                                ;ZBIORU
PRINTER_CHANNEL EQU     04H     ;STANDARDOWY KANAL DRUKARKI
SET_DTA         EQU     1AH     ;FUNKCJA DOS:ZMIANA BUFORA DYSKU
TERMINATE       EQU     4CH     ;FUNKCJA DOS:ZAKONCZENIE
                                ;PROGRAMU
TRUE            EQU     0FFH
WRITE_HANDLE    EQU     40H     ;FUNKCJA DOS:WYPROWADZENIE
                                ;CIAGU ZNAKOW
...
ATTRIBUTE_MASK1 DB      FALSE   ;MASKA DLA PRZETWARZANIA ATRYBU-
                                ;TOW ZBIORU (USTAWIANIE ATRYBU-
                                ;TOW)
ATTRIBUTE_MASK2 DB      FALSE   ;MASKA DLA PRZETWARZANIA ATRYBU
                                ;TOW ZBIORU (KASOWANIE ATRYBU-
                                ;TU)
COMMAND_BUFFER  DW      128 DUP(?)
                                ;ZAWIERA PRZEKAZANA KOPIE LINII
                                ;ROZKAZOWEJ
COMMAND_LENGTH  DB      ?       ;ZAWIERA DLUGOSC LINII ROZKAZOWEJ
DUMMY_FCB       DB      37 DUP(?)
                                ;FCB WYKORZYSTYWANE PRZEZ
                                ;FUNKCJE PARSE (BEDZ KTORE NIE
                                ;JEST WYKORZYSTYWANE DLA PROGRA
                                ;MU)
FILE_LENGTH     DB      ?       ;DLUGOSC PRZEKAZANEJ NAZWY
                                ;ZBIORU
FILE_NAME       DB      13 DUP(?)
                                ;BUFOR DLA PRZEKAZANEJ NAZWY
                                ;ZBIORU
OUT_HANDLE      DW      ?       ;PAMIEC KANALU UZYWANEGO DLA
                                ;WYPROWADZANIA (DRUKARKA LUB
                                ;KONSOLA)
PRINT_FLAG      DB      FALSE   ;OKRESLA URZADZENIE WYJSCIA
                                ;(FALSE => CON, TRUE => PRN)
SECOND_HANDLE   DW      ?       ;ZAWIERA KANAL DLA WYPROWADZENIA
                                ;NA KONSOLE, JESLI WYPROWADZENIE
                                ;ZOSTALO PRZELACZONE NA DRUKARKE


START:  MOV     AH,GET_DTA      ;DOS: ODCZYTANIE ADRESU BUFORA
                                ;DYSKU (TAM ZNAJDUJE SIE WYWOLU-
                                ;JACA LINIA ROZKAZOWA)
        INT     DOS
        ...
DECODE: MOV     AL,[DI]         ;ZALADOWANIE ROZKAZU
        PUSH    DI              ;ZACHOWANIE ADRESU ROZKAZU
        MOV     DI,OFFSET COMMANDS
                                ;ADRES WLASCIWEGO ROZKAZU
        MOV     CX,OFFSET END_COMMANDS + 1
        SUB     CX,DI           ;OBLICZENIE DLUGOSCI + 1 TABLICY
                                ;ROZKAZOW
        REPNE   SCASB           ;PRZESZUKANIE TABLICY
        CMP     CX,0            ;CZY ROZKAZ ZOSTAL ZNALEZIONY ?
        CALL    OUTPUT_HELP     ;EWENT.WYPROWADZENIE HELP TEXT
        POP     DI              ;ODTWORZENIE ADRESU ROZKAZU
        MOV     SI,OFFSET END_MASK
                                ;OBLICZENIE ADRESU MASKI
                                ;ATRYBUTU
        SUB     SI,CX
        MOV     AL,[SI]         ;ZALADOWANIE MASKI ROZKAZU
```

AX — 02H, jeśli program nie został znaleziony lub podana ścieżka jest niepoprawna
AX — 08H, jeśli program nie może zostać załadowany, ponieważ wolny obszar pamięci jest zbyt mały
AX — 0AH, jeśli program nie może być wykonany w aktualnym środowisku
AX — 0BH, jeśli program wywołujący jest plikiem typu EXE mającym niezgodną zawartość
Działanie funkcji 4BH jest wyznaczone przez zawartość rejestru AL.

a) AL = 0

Ponieważ w procesie ładowania na ogół cała pamięć jest rezerwowana na program (jeśli maksymalna wielkość programu nie została ograniczona w procesie łączenia), przed załadowaniem programu należy użyć funkcji 4AH w celu zapewnienia odpowiednio dużego obszaru wolnej pamięci. Najpierw zakładany jest obszar PSP dla ładowanego programu. Wektory przerwań 22H i 23H zostają ustawione na adres rozkazu, który występuje w programie ładującym bezpośrednio po wywołaniu funkcji ładującej (CALL 50H w PSP lub INT 21H), dzięki czemu zapewnia się poprawne kontynuowanie programu wywołującego.

Ponieważ program ładujący rozporządza wszystkimi otwartymi już plikami procesu wywołującego, proces ten może ograniczyć dostęp do kanałów wyjściowych (CON, AUX, PRN) przez ich adresowanie. Oznacza to, że program wykonywany samodzielnie, który dokonuje operacji wprowadzania i wyprowadzania informacji przez kanały 00H i 01H (standardowe we-wy; CON), w wyniku przeadresowania wejścia i wyjścia (np. na pewien plik) może zostać użyty do przetwarzania lub filtrowania danych programu wywołującego. Podobne działanie występuje, na przykład, podczas stronicowego, alfabetycznego wyprowadzania na urządzenie zewnętrzne skorowidza plików (DIR | SORT | MORE), przy czym do dalszego przetwarzania zostają użyte potoki (ang. pipes).

Niniejszy program ilustruje użycie funkcji typu Xenix do zmieniania atrybutów pliku i przeadresowywania; mniej istotne fragmenty zostały pominięte

```
        CALL    GET_MASK        ;ZMIANA MASKI ATRYBUTU LUB
                                ;SKIEROWANIE WYPROWADZENIA NA
                                ;DRUKARKE
        CMP     DX,0            ;CZY LANCUCH ZNAKOW ROZKAZU
                                ;JEST ZAKONCZONY ?
        JNZ     RECORR          ;NIE
        CALL    REDIRECT_IO     ;SKIEROWAC WYPROWADZANIE NA
                                ;DRUKARKE, JEZLI PRINT_FLAG =
                                ;= TRUE
        CALL    PROCESS_FILES   ;ZMIENIC ATRYBUTY
        CALL    RESTORE_IO      ;ODTWORZYC WYPROWADZANIE NA
                                ;KONSOLE
        XOR     AL,AL           ;RETURNCODE OOH
        MOV     AH,TERMINATE    ;ZAKONCZENIE PROGRAMU
        INT     DOS

GET_MASK        PROC    NEAR
        CMP     AL,TRUE         ;CZY JEST TO ROZKAZ /P ?
        JNZ     GET_ATTRIBUTE   ;NIE
        MOV     PRINT_FLAG,AL   ;USTAWIENIE WSKAZNIKA PRINT
                                ;(WYPROWADZENIE NA DRUKARKE)
        MOV     BP,1            ;ODLEGLOSC NASTEPNEGO '/' OD P
        JMP     COMMON_SET      ;WSPOLNE DALSZE PRZETWARZANIE
GET_ATTRIBUTE:
        CMP     BYTE PTR [DI+1],'+'
                                ;CZY NASTEPUJACYM PO ROZKAZIE
                                ;ZNAKIEM JEST '+' ?
        JNZ     CHECK_MINUS     ;NIE
        OR      ATTRIBUTE_MASK,AL
                                ;NALEZY USTAWIC ATRYBUT
        JMP     NEXT_COMMAND
CHECK_MINUS:
        CMP     BYTE PTR [DI+1],'-'
                                ;CZY SKASOWAC ATRYBUT ?
        JZ      RESET_ATTRIBUTE ;TAK
OUT_HELP:
        TEST    AL,0            ;USTAWIENIE WSKAZNIKA Z
        CALL    OUTPUT_HELP     ;WYPROWADZENIE HELP_TEXT
RESET_ATTRIBUTE:
        OR      ATTRIBUTE_MASK,AL
                                ;SKASOWANIE ATRYBUTU
NEXT_COMMAND:
        MOV     BP,2            ;ODLEGLOSC NASTEPNEGO
                                ;ROZKAZU
COMMON_SET:
        CMP     BYTE PTR DS:[DI+BP],0
        JNZ     OUT_HELP        ;WYPROWADZENIE HELP_TEXT
        INC     BP
        ADD     DI,BP           ;ADRES NASTEPNEGO ROZKAZU
        SUB     BX,BP           ;LICZBA POZOSTALYCH ZNAKOW
        RET
GET_MASK        ENDP

REDIRECT_IO     PROC    NEAR
        CMP     PRINT_FLAG,TRUE ;CZY WYPROWADZIC NA DRUKARKE?
        JNZ     END_REDIRECT    ;NIE
        MOV     AH,DUPLICATE    ;ZALOZENIE DRUGIEGO KANALU
                                ;LOGICZNEGO
        MOV     BX,OUT_HANDLE   ;KANAL DUBLUJACY (KONSOLA)
        INT     DOS
        JC      ERROR_HANDLE    ;WYPROWADZENIE KOMUNIKATU
                                ;BLEDU
        MOV     SECOND_HANDLE,AX;ZAPAMIETANIE DRUGIEGO KANALU,
                                ;ABY MOC GO POTEM PRZELACZYC
                                ;SPOWROTEM NA PIERWSZY
        MOV     AH,FORCE_HANDLE ;DOS:WYMUSZONE PRZELACZENIE
                                ;KANALU
        MOV     BX,PRINTER_CHANNEL
                                ;KANAL DLA DRUKARKI
        MOV     CX,OUT_HANDLE   ;WYPROWADZENIE NA KONSOLE
        INT     DOS             ;WYPROWADZENIE PRZEZ KANAL
                                ;DLA KONSOLI ZOSTAJE PRZELA-
                                ;CZONE NA DRUKARKE
        JNC     END_REDIRECT    ;NIE WYSTAPIL ZADEN BLAD
ERROR_HANDLE:
        CALL    PROCESS_ERROR   ;OBSLUGA BLEDU
END_REDIRECT:
        RET
REDIRECT_IO     ENDP

RESTORE_IO      PROC    NEAR
        CMP     PRINT_FLAG,TRUE ;CZY SKIEROWAC WYPROWADZANIE NA
                                ;DRUKARKE ?
        JNZ     END_RESTORE     ;NIE
        MOV     AH,FORCE_HANDLE ;DOS:WYMUSZONE PRZELACZENIE
                                ;KANALU
        MOV     BX,SECOND_HANDLE;UZYCIE ZDUBLOWANEGO KANALU
                                ;DLA KONSOLI W CELU POZNIEJSZEGO
                                ;PRZELACZENIA WYPROWADZANIA Z
                                ;DRUKARKI SPOWROTEM NA KONSOLE
        MOV     CX,OUT_HANDLE   ;ZAWIERA NUMER KANALU DLA
                                ;PIERWOTNEGO WYPROWADZENIA NA
                                ;KONSOLE
        INT     DOS             ;DLA KONSOLI ZOSTAJE PRZELACZONE
                                ;NA DRUKARKE
        JNC     END_RESTORE     ;NIE WYSTAPIL ZADEN BLAD
        CALL    PROCESS_ERROR   ;OBSLUGA BLEDU
END_RESTORE:
        RET
RESTORE_IO      ENDP

PROCESS_FILES   PROC    NEAR
        MOV     AH,SET_DTA      ;DOS:USTAWIENIE BUFORA DYSKU
        MOV     DX,OFFSET COMMAND_BUFFER
        INT     DOS
        MOV     DX,OFFSET FILE_NAME
                                ;USTANIENIE ADRESU NAZWY ZBIO-
                                ;RU
        MOV     CX,00100111B    ;ATRYBUT DLA FUNKCJI SZUKANIA
        MOV     AH,FIND_FIRST   ;DOS:SZUKANIE PIERWSZEGO
                                ;TZW. WEJSCIA ZBIORU
        INT     DOS
        JC      FIND_ERROR      ;OBSLUGA BLEDOW.
FIND_LOOP:
        MOV     DX,OFFSET COMMAND_BUFFER + 1EH
                                ;ADRES ZNALEZIONEJ NAZWY
                                ;ZBIORU
        MOV     AL,00H          ;FUNKCJA:WCZYTANIE ATRYBUTU
        MOV     AH,CHANGE_MODE  ;DOS:ODCZYT/USTAWIENIE ATRYBU-
                                ;TU
        INT     DOS
        JC      CHANGE_ERROR    ;OBSLUGA BLEDOW
        MOV     AL,ATTRIBUTE_MASK
                                ;NEGACJA MASKI DLA USUNIECIA
                                ;ATRYBUTOW
        NOT     AL
        AND     CL,AL
        OR      CL,ATTRIBUTE_MASK
                                ;USTAWIENIE ZADANEGO ATRYBUTU
        MOV     AL,1            ;FUNKCJA:USTANTENIE ATRYBUTU
        MOV     AH,CHANGE_MODE  ;DOS:ODCZYT/USTAWIANIE ATRYBUTU
        INT     DOS
        JC      CHANGE_ERROR    ;OBSLUGA BLEDOW
        MOV     DI,OFFSET COMMAND_BUFFER + 1EH
        MOV     DX,DI           ;USTAWIENIE ADRESU NAZWY ZBIORU
        MOV     AL,0
        MOV     CX,1FH          ;MAKSYMALNA DLUGOSC NAZWY ZBIORU
        REPNE   SCASB.          ;SZUKANIE KONCA NAZWY ZBIORU
        DEC     DI              ;OBLICZENIE DLUGOSCI NAZWY
                                ;ZBIORU
        SUB     DI,DX
        MOV     CX,DI
        MOV     BX,OUT_HANDLE   ;USTAWIENIE KANALU DLA WYPROWA-
                                ;DZANIA
        MOV     AH,WRITE_HANDLE ;DOS:WYPROWADZENIE ZNAKU
        INT     DOS
        MOV     DX,OFFSET NEW_LINE
                                ;WYPROWADZENIE CR, LF
        MOV     CX,2
        MOV     BX,OUT_HANDLE
        MOV     AH,WRITE_HANDLE ;DOS:FUNKCJA DLA WYPROWADZANIA
        INT     DOS
        MOV     AH,FIND_NEXT    ;DOS:SZUKANIE NASTEPNEGO
                                ;TZW. WEJSCI ZBIORU
        INT     DOS
        JNC     FIND_LOOP       ;OBSLUGA BLEDU
        JMP     END_SET
FIND_ERROR:
CHANGE_ERROR:
        CALL    PROCESS_ERROR   ;OBSLUGA BLEDU
END_SET:
        RET
PROCESS_FILES   ENDP
CODE    ENDS
        END     START
```

Tabela 1. Budowa bloku parametrów wymaganego do ładowania i ewentualnego wykonania programu dla funkcji 4BH (AL = 0)

Wyrównanie	Długość	Znaczenie
00H	2	Adres segmentowy przekazywanego środowiska. Deklaracje zawarte w tym segmencie, począwszy od wyrównania 00H, muszą być zakończone przez dwukrotne 00H. Długość całkowita wszystkich deklaracji nie może przekraczać 32 KB. Jeśli adres segmentowy jest równy 0000H, to używane jest środowisko programu wywołującego.
02H	4	Segment i wyrównanie linii poleceń, która powinna zostać przekazana do wyrównania 80H (DTA) nowego PSP. Linia może mieć długość 128 bajtów.
06H	4	Segment i wyrównanie nieotwartego bloku FCB, który powinien zostać zapamiętany od wyrównania 5CH w nowym PSP (FCB1).
0AH	4	Segment i wyrównanie nieotwartego bloku FCB, który powinien zostać zapamiętany od wyrównania 6CH w nowym PSP (FCB2).

Dla każdego pliku może zostać utworzone specjalne środowisko (ang. environment), w którym jest, na przykład, podana nazwa interpretera poleceń i adres, pod którym może on zostać znaleziony. Wpisy w tym środowisku stanowią ciągi znaków zakończone przez 00H. Ostatni ciąg znaków jest zakończony przez 00H, 00H. Całkowity obszar pamięci dostępny dla środowiska jest ograniczony do 32 KB. Wszystkie wpisy do środowiska mają postać: „parametr = znaczenie". Ponieważ byłoby uciążliwe tworzenie oddzielnego środowiska dla każdego ładowania programu, istnieje możliwość użycia środowiska programu wywołującego. Bliższe informacje zawarto w tabeli 1, która przedstawia budowę bloku parametrów wymaganego przez tę funkcję w procesie ładowania.

b) AL = 3

Funkcji tej używa się do ładowania nakładki. PC-DOS nie wytwarza żadnego nowego PSP i przyjmuje, że program zostanie załadowany do obszaru pamięci udostępnionego przez program wywołujący. W tabeli 2 pokazano budowę bloku parametrów dla tej funkcji.

Tabela 2. Opis bloku parametrów potrzebnego do ładowania i wykonania programu dla funkcji 4BH (AL = 3)

Wyrównanie	Długość	Znaczenie
00H	2	Adres segmentowy, pod który powinien zostać załadowany program (od wyrównania 00H)
02H	2	Współczynnik relokacji — na ogół ma tę samą wartość co dwa poprzednie bajty

Funkcja 4CH — zakończenie programu

Wywołanie tej funkcji powoduje zakończenie programu, zamknięcie wszystkich plików otwartych przez ten program oraz udostępnienie stanu wywołanego programu. Podobnie jak w wypadku funkcji ODH wektory przerwań od 22H do 24H określone przez wyrównanie 0AH-15H zostają sprowadzone ponownie do stanu sprzed wywołania zakończonego programu.

Parametr wejściowy:
AH — 4CH

Funkcja 4DH — odczytanie stanu przekazanego przez funkcję 4CH

Funkcja ta umożliwia jednokrotne odczytanie stanu zakończonego programu. Jeśli funkcja będzie wywoływana więcej niż jeden raz, to efektem będzie oczekiwanie na zakończenie nigdy nie rozpoczętego programu. Funkcja może zostać wykorzystana w przyszłym wielozadaniowym systemie operacyjnym DOS do synchronizacji dwóch procesów współbieżnych.

Parametr wejściowy:
AH — 4DH
Parametry wyjściowe:
AL — ostatni stan programu zakończonego przy użyciu funkcji 4CH
00H — normalne zakończenie
01H — zakończenie przez CTRL-C
02H — zakończenie w wyniku poważnego błędu urządzenia
03H — zakończenie przy użyciu funkcji 31H, tj. zakończenie programu bez usuwania go z pamięci
04H-FFH — stan przekazany explicite przez zakończony program.

INNE FUNKCJE ZARZĄDZANIA PROGRAMEM

Z wyżej przedstawionymi funkcjami wiążą się trzy inne nie omówione dotychczas funkcje zarządzania programem.

Funkcja 00H — zakończenie programu

Funkcja kończy program. Wykonywana jest bezpośrednio lub przez wywołanie przerwania 20H. Ponieważ CS:0 musi wskazywać na PSP, ten rodzaj zakończenia programu nie jest zbyt wygodny, przynajmniej dla programów typu EXE. Dlatego też zaleca się używanie funkcji 4CH. Pliki zmodyfikowane powinny przed wywołaniem tej funkcji zostać zamknięte. Wektory przerwań 22H, 23H i 24H zostają sprowadzone (zgodnie z wyrównaniami od 0AH do 15H obszaru PSP) do stanu sprzed wywołania programu.
Parametry wejściowe:
AH — 00H
CS:0 — adresu obszaru PSP

Funkcja 26H — utworzenie duplikatu obszaru PSP

Powyższa funkcja zakłada pod danym adresem nowy obszar PSP. W tym celu pierwotny obszar PSP (włącznie z zawartością bufora dyskowego) zostaje skopiowany pod nowy adres. Adres bufora (DTA) nie ulega zmianie.
Parametry wejściowe:
AH — 26H
DX:0 — adres, pod którym powinien zostać założony nowy obszar PSP
CS:0 — adres kopiowanego obszaru PSP

Funkcja 31H — zakończenie programu i pozostawienie go w pamięci

Program znajdujący się w pamięci, zostaje zakończony, jednakże pamięć zajmowana przez ten program nie zostaje zwolniona, w wyniku czego zostaje on zachowany do ponownego wywołania.
Parametry wejściowe:
AH — 31H
DX — obszar pamięci niezbędny dla programu pozostającego jako rezydentny (wielokrotność 16 bajtów).

INNE FUNKCJE DOTYCZĄCE PLIKÓW

Spośród funkcji typu Xenix dotyczących operowania plikami pozostały do omówienia jeszcze trzy.

Tabela 3. Format bloku danych przekazywanego przez funkcje 4EH i 4FH

Wyrównanie adresu		Opis
początkowego	końcowego	
00H	14H	Informacje potrzebne do kolejnego przeszukiwania (obszar ten nie powinien zostać zmodyfikowany przez użytkownika)
15H	15H	Rzeczywisty atrybut znalezionego pliku
16H	17H	Czas ostatniego dokonanego zapisu
18H	19H	Data ostatniego dostępu do pliku dla zapisu
1AH	1DH	Wielkość pliku (bajty 1AH—1BH zawierają słowo mniej znaczące)
1EH	2AH	Nazwa pliku (nazwa i rozszerzenie są rozdzielone kropką) zakończona przez 00H

dokończenie s. 30

ANDRZEJ WARDA
Instytut Maszyn Matematycznych
Warszawa

Pamięci dyskowe Mazovii 1016 (I)

W pierwszej części artykułu omówiono sterownik dysków elastycznych oraz znajdujący się na tym samym pakiecie sterownik drukarki.

NAPĘD DYSKÓW ELASTYCZNYCH

Mikrokomputer Mazovia 1016 jest wyposażony w dwa 5,25-calowe napędy dysków elastycznych. Są to standardowe napędy dyskowe stosowane w komputerach kompatybilnych z IBM PC/XT, o maksymalnej pojemności dyskietek 360 kB, dwustronne o gęstości zapisu 48 ścieżek na cal, czyli 40 ścieżek na powierzchnię. Inne charakterystyczne parametry zestawiono w tabeli 1.

Tabela 1. Parametry napędu dysków elastycznych stosowanych w Mazovii 1016

Nazwa	Wartość
Czas pozycjonowania	6 ms (ze ścieżki na ścieżkę)
Czas uspokojenia głowicy	15 ms
Stopa błędu:	
— korygowalny błąd odczytu	$1/10^9$ bitów odczytanych
— niekorygowalny błąd odczytu	$1/10^{12}$ bitów odczytanych
— błąd pozycjonowania	$1/10^6$ szukanych ścieżek
Prędkość obrotowa dysku	300±1,5% obr./min
Czas rozruchu	0,5 s (maksymalnie)
Szybkość transmisji:	
— pojedyncza gęstość	125 kb/s
— podwójna gęstość	250 kb/s
Metody kodowania	FM, MFM
Zasilanie	+12 V±0,6 V, 0,9 A (średnio)
	+5 V±0,25 V, 0,6 A (średnio)

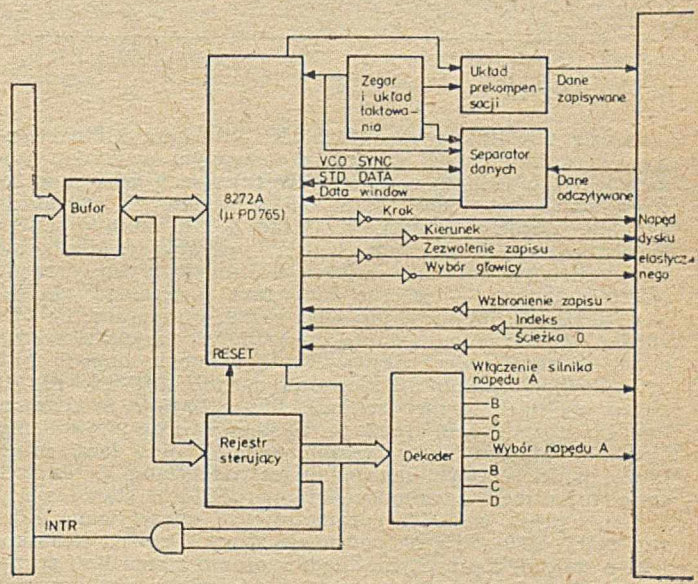
Głównymi elementami napędu są: mechanizm obrotu dyskietki, głowica zapisu-odczytu i mechanizm pozycjonujący głowicę. Do zapisu i odczytu danych w postaci cyfrowej zastosowano zmodyfikowaną modulację częstotliwości (MFM). Dane są odczytywane z dyskietki przez układ odczytu, w którego skład wchodzą: wzmacniacz sygnałów niskonapięciowych, układ różnicowy, detektor przejścia napięcia przez zero i układ wytwarzający cyfrową postać sygnału. Dekodowanie danych jest dokonywane przez sterownik dysków elastycznych.

Napęd dysków elastycznych jest wyposażony standardowo w następujące czujniki:

1. Czujnik ścieżki zerowej, który wykrywa obecność głowicy na ścieżce zerowej.
2. Czujnik otworu indeksującego, składający się z fototranzystora i ze źródła światła. Układ ten generuje aktywny sygnał, gdy na jego wysokości znajduje się otwór indeksujący dyskietki, określający początek ścieżki.
3. Czujnik blokujący układ zapisu w napędzie dysku elastycznego po wykryciu zabezpieczenia zapisu na dyskietce.

STEROWNIK DYSKÓW ELASTYCZNYCH

W podstawowej konfiguracji mikrokomputer Mazovia 1016 jest wyposażony w dwa napędy dysków elastycznych. Do sterowania tymi napędami służy sterownik, do którego można dołączyć maksymalnie cztery napędy dysków elastycznych (dwa zainstalowane wewnątrz jednostki centralnej i dwa zewnętrzne, dołączone do złącza wyprowadzonego na tylną płytę komputera). Napędy są łączone płaskim kablem wielożyłowym z łączówkami połączonymi równolegle. Jeden kabel obsługuje dwa napędy. Przy końcowej łączówce każdego z dwóch kabli jest zrobiony przeplot odpowiednich linii, aby zapewnić odpowiedni wybór napędu. Sterownik jest oparty na układzie LSI typu μPD765 firmy NEC lub jego funkcjonalnym odpowiedniku (np. 8272A firmy Intel lub bułgarskim CM609). W związku z tym parametry napędów mogą być programowane. Przy zapisie danych na dyskietce można korzystać z modulacji FM lub MFM oraz prekompensacji. Do prekompensacji danych przy zapisie i separacji danych przy odczycie służy układ WD9229BT. W celu przyspieszenia przesyłania danych sterownik wykorzystuje kanał bezpośredniego dostępu do pamięci (DMA). Zakończenie operacji jest sygnalizowane przerwaniem.



Rys. 1. Schemat blokowy sterownika dysków elastycznych

Z punktu widzenia programowego sterownik dysków elastycznych stanowią trzy rejestry: rejestr sterowania silnikami napędów i wyborem napędu oraz dwa wewnętrzne rejestry układu 8272A, rejestr stanu i rejestr danych. Adresy tych rejestrów w przestrzeni wejścia-wyjścia są następujące: 3F2H, 3F4H i 3F5H.

Rejestr sterowania silnikami napędów i wyborem napędu

Do tego rejestru można tylko wpisywać słowo sterujące. Znaczenie poszczególnych bitów przedstawiono w tabeli 2. Rejestr ten jest programowany bezpośrednio przez komputer. Po inicjalizacji systemu wszystkie bity tego rejestru przyjmują wartość logiczną „0".

Mgr inż. ANDRZEJ WARDA ukończył w 1984 r. studia na Wydziale Elektroniki Politechniki Warszawskiej. Pracuje w Instytucie Maszyn Matematycznych. Zajmuje się projektowaniem logicznym mikrokomputerów.

Tabela 2. Znaczenie bitów rejestru sterowania

Bit	Znaczenie
0, 1	Dekodowane układowo wybierają dany napęd, jeśli jego silnik jest włączony: 00 — napęd A, 01 — napęd B, 10 — napęd C, 11 — napęd D
2	Wartość logiczna ,,0'' na tej pozycji powoduje zerowanie układu 8272A
3	Wartość logiczna ,,1'' zezwala na transmisję DMA oraz generowanie przerwań przez układ 8272A
4, 5, 6, 7	Bity te sterują silnikami napędów; Wartość logiczna ,,0'' na danej pozycji powoduje wyłączenie odpowiedniego silnika i wybór tego napędu jest niemożliwy

Rejestr stanu

Jest to 8-bitowy, wewnętrzny rejestr układu 8272A. Zawiera podstawowe informacje o stanie układu i jest dostępny w dowolnym momencie. Rejestr ten może być tylko odczytywany, a jest wykorzystywany dla prawidłowej organizacji przesyłania danych do lub z rejestru danych. Znaczenie poszczególnych bitów podano w tabeli 3.

Tabela 3. Znaczenie bitów rejestru stanu

Bit	Znaczenie
0, 1, 2, 3	Odpowiedni napęd jest w trakcie wykonywania operacji pozycjonowania (SEEK)
4	Sterownik jest w trakcie wykonywania operacji zapisu lub odczytu
5	Sterownik pracuje bez trybu DMA
6	Określa kierunek przesyłania danych między sterownikiem dysków a procesorem. Wartość logiczna ,,1'' oznacza przesyłanie z rejestru danych do procesora
7	Wskazuje gotowość rejestru danych do przyjęcia lub wysłania danych z lub do procesora

Rejestr danych

Rejestr ten jest w rzeczywistości zestawem kilku rejestrów zorganizowanych w stos, z dostępnym w danym momencie tylko jednym rejestrem. W rejestrach tych są przechowywane dane, kody rozkazów, parametry i informacje o stanie sterownika dysków. Układ 8272A może wykonać 15 elementarnych operacji. Każda operacja zaczyna się wielobajtową (od 1 do 9 bajtów) transmisją z procesora do rejestru danych — jest to tzw. faza programowania. W fazie tej przekazywany jest kod rozkazu i parametry niezbędne dla wykonania danej operacji. W zależności od rodzaju operacji najczęściej występują: numer napędu, numer głowicy, sposób zakodowania danych (MF czy MFM), numer cylindra, numer sektora, liczba bajtów w sektorze itd. Po rozpoznaniu rozkazu następuje faza wykonania. Ostatnim etapem wykonywania operacji jest faza rezultatów, podczas której niezbędne jest odczytanie przez procesor informacji o zakończeniu operacji i o stanie sterownika. Oprócz wartości parametrów są odczytywane rejestry stanu ST0—ST3, na podstawie których można wnioskować o rodzaju i przyczynie błędów oraz o stanie linii sprzęgu między sterownikiem a napędem dysków elastycznych. W zależności od rodzaju operacji w fazie rezultatów jest przesyłane od 0 do 7 bajtów.

Lista rozkazów sterownika

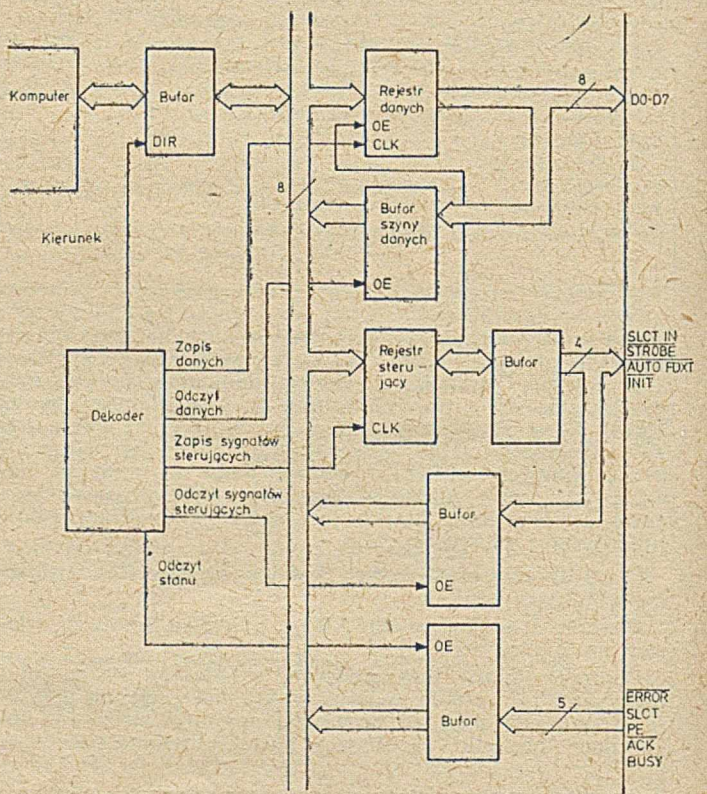
Read data (odczyt danych)
Format track (formatowanie ścieżki)
Scan low or equal (porównanie — mniejszy lub równy)
Scan high or equal (porównanie — większy lub równy)
Scan equal (porównanie, czy równy)
Recalibrate (ustawienie głowicy na ścieżce nr 0)
Sense interrupt status (pobranie informacji o przerwaniu)
Specify (definiowanie parametrów napędu i transmisji)
Sense drive status (odczytanie stanu napędu)
Seek (pozycjonowanie)
Invalid (rozkaz nielegalny)
Read deleted data (odczyt sektora odrzuconego)
Write data (zapis danych)
Write deleted data (zapis sektora ze znacznikiem sektora odrzuconego)
Read a track (odczyt całej ścieżki)
Read ID (odczyt nagłówka)

Linie sprzęgu sterownika z napędem dysku

Poniżej wymieniono sygnały odpowiadające liniom sprzęgu sterownika z napędem dysku elastycznego:
Drive Select A (B) — sygnał wyboru napędu
Motor Enable A (B) — sygnał włączania silnika
Step — przesuw głowicy o jedną ścieżkę po każdym impulsie na tej linii; kierunek przesuwu zależy od stanu linii Direction
Direction — kierunek przesuwu głowicy
Head Select — wybór głowicy
Write Enable — zezwolenie na zapis
Write Data — linia zapisu danych
Read Data — linia odczytu danych
Index — aktywny impuls przy każdym obrocie dyskietki
Write Protect — wybrany napęd uaktywnia tę linię w wypadku włożenia dyskietki z zabronionym zapisem
Track 0 — wybrany napęd uaktywnia tę linię, gdy głowica znajduje się nad ścieżką 0

STEROWNIK DRUKARKI

Na tym samym pakiecie co sterownik dysków elastycznych znajduje się sterownik drukarki. Służy on do sterowania drukarką z równoległym sprzęgiem (np. typu Centronics), ale może też być użyty jako równoległy, uniwersalny port wejścia-wyjścia.



Rys. 2. Schemat blokowy sterownika drukarki

Sterownik ma 12 wyjść TTL, które są zatrzaskiwane w rejestrze i mogą być zapisywane i odczytywane programowo przy użyciu instrukcji IN i OUT. Linie te są doprowadzone do złącza zainstalowanego na tylnej płycie komputera. Ponadto sterownik ma pięć wejść, których stan może być odczytywany przez procesor: Acknowledge (sygnał potwierdzenia z drukarki), Busy (sygnał zajętości), P.End (sygnalizacja końca papieru), Select (sygnał wyboru) i Error (sygnalizacja błędu). Jedno z wejść (Acknowledge) jest użyte do generowania przerwania, jeśli program zezwolił na jego wygenerowanie.

dokończenie na s. 11

MACIEJ M. SYSŁO
Instytut Informatyki
Uniwersytet Wrocławski

Uwagi o powszechnym kształceniu informatycznym studentów wyższych uczelni

Szybki rozwój elektroniki i masowa produkcja mikrokomputerów spowodowały prawdziwą rewolucję, której objawem jest przenikanie mikroprocesorów do prawie każdej dziedziny działalności człowieka. Stawia to przed szkolnictwem średnim i wyższym obowiązek przygotowania uczniów i studentów do nowych zagadnień w trakcie zdobywania wiedzy, a później w pracy zawodowej. Doceniając znaczenie wykształcenia informatycznego dla właściwego ukształtowania sylwetki przyszłych absolwentów szkół wyższych, specjalny zespół kierowany przez prof. St. Węgrzyna wraz z Ośrodkiem Informacji i Informatyki MNiSzW opracował *Program Rozwoju Zastosowań Techniki Komputerowej w Procesach Kształcenia w Szkołach Wyższych w latach 1986—1990* (nazywany dalej *Programem*), który pod koniec 1985 r. rozesłano do wszystkich szkół wyższych w kraju. W pierwszej części tego dokumentu przytoczono podstawę działania, którą stanowią oficjalne dokumenty rządowe, mówiące o potrzebie kształcenia informatycznego, scharakteryzowano stan edukacji informatycznej w szkołach wyższych w kraju i na świecie oraz określono cele do osiągnięcia. W drugiej, szczegółowej części omówiono założenia programu i aktualny stan informatyki w kraju oraz zaproponowano działania niezbędne do realizacji programu powszechnej edukacji informatycznej i zastosowania techniki komputerowej w procesach kształcenia w latach 1986—1990. Działania te obejmują: intensywne szkolenie kadry dydaktycznej, zmianę planów i programów (uwzględniając wprowadzenie metod informatycznych do przedmiotów zawodowych), opracowanie i wprowadzenie we wszystkich uczelniach przedmiotu „Podstawy informatyki" oraz wyposażenie uczelni w sprzęt komputerowy i oprogramowanie. W Programie oszacowano także wielkość środków finansowych potrzebnych do jego realizacji we wszystkich wyższych uczelniach w kraju; zamykają się one kwotą 23 mld złotych, w tym prawie 16 mln dolarów.

Autorzy Programu wyróżniają dwa rodzaje zajęć informatycznych na kierunkach nieinformatycznych: w jednych — narzędzia (takie jak komputer) i metody informatyczne są głównymi obiektami zajęć, a w drugich — nauczanie przedmiotów zawodowych (kierunkowych) jest wspomagane technikami komputerowymi. Zajęcia pierwszego rodzaju, noszące w Programie wspólne miano „Podstaw informatyki", powinny dostarczać wiadomości z informatyki niezbędne wszystkim absolwentom szkół wyższych. Ramowe programy zajęć z „Podstaw informatyki" mają być przygotowane przez odpowiednie Zespoły Dydaktyczno-Naukowe przy MNiSzW w czterech opcjach (dla uczelni technicznych i rolniczych, uniwersytetów, szkół ekonomicznych i wyższych szkół pedagogicznych) i stać się podstawą do opracowania szczegółowych programów zajęć na poszczególnych kierunkach i specjalnościach. Wykład, ćwiczenia i laboratorium z „Podstaw informatyki" mają być wprowadzone jako zajęcia obowiązkowe na I roku studiów, od roku akademickiego 1987—1988. W przypadku zajęć drugiego rodzaju, Program przewiduje, iż w latach 1986—1987 uczelnie opracują własne propozycje programów kształcenia informatycznego w przedmiotach zawodowych. W związku z tymi ostatnimi planami, wraz z tekstem Programu uczelnie otrzymały polecenie przesłania do Ministerstwa NiSzW w terminie do 30 czerwca 1986 r. programów przedmiotów kierunkowych, w których są stosowane metody i techniki komputerowe.

W artykule przedstawiono spostrzeżenia i wnioski zebrane w trakcie wykonywania tego ostatniego polecenia na terenie Uniwersytetu Wrocławskiego. Niektóre z obserwacji dotyczą także obecnej i przyszłej realizacji zajęć z „Podstaw informatyki". Omówiono także doświadczenia zebrane w zagranicznych uczelniach, zamieszczone w materiałach z konferencji w 1983 roku, poświęconej powszechnej edukacji informatycznej studentów uniwersytetów.

Informatyka w Uniwersytecie Wrocławskim

Na początku lat sześćdziesiątych powstała w Instytucie Matematycznym Uniwersytetu Wrocławskiego Katedra Metod Numerycznych, której pracownicy prowadzili zajęcia ze studentami metod numerycznych, jednej z czterech specjalizacji matematycznych obieranych po drugim roku studiów. W połowie lat siedemdziesiątych Katedra Metod Numerycznych przerodziła się w samodzielny Instytut Informatyki, a specjalizacja „Metody Numeryczne" — w „Informatykę", kierunek studiów z własnym i niezależnym naborem studentów. Sprzęt komputerowy Uczelni, którego podstawę stanowiły kolejno Elliott 803, Odra 1204 i Odra 1305 wraz z RC 3600 i SM 4, jest obsługiwany przez pracowników Centrum Obliczeniowego, wchodzącego w skład Instytutu Informatyki. Centrum Obliczeniowe, poza obsługą studentów i pracowników Instytutu, świadczy także u-

Doc. dr hab. MACIEJ M. SYSŁO ukończył w 1968 r. studia matematyczne w zakresie metod numerycznych w Uniwersytecie Wrocławskim. Doktoryzował się w 1974, a habilitował w 1980 roku z teorii grafów i jej algorytmów. W latach 1974—1976 był stypendystą rządu japońskiego w Uniwersytecie Tokijskim. W 1982—1984 przebywał w Uniwersytecie Bońskim jako stypendysta Fundacji Humboldta, a w 1981—1982 roku był profesorem wizytującym w Uniwersytecie stanu Washington w Pullman (USA).

Jest autorem i współautorem blisko stu publikacji, w tym dwóch książek wydanych przez PWN i PrenticeIHall (USA) poświęconych algorytmom optymalizacji i ich komputerowym implementacjom w językach Algol-60 i Pascal.

Obecne naukowe zainteresowania dotyczą kombinatorycznych i algorytmicznych aspektów zbiorów częściowo uporządkowanych. Zajmuje się także upowszechnianiem metod informatycznych w dydaktyce szkół średnich i wyższych.

Od 1984 roku kieruje Instytutem Informatyki Uniwersytetu Wrocławskiego.

sługi dla innych kierunków i administracji Uczelni oraz wykonuje prace na rzecz innych uczelni i zakładów przemysłowych, m.in. wiele zleceń obliczeń wchodzących w skład prac promocyjnych (doktorskich i habilitacyjnych). Należy zaznaczyć, że dotychczas żadna jednostka organizacyjna Uczelni, łącznie z Instytutem Informatyki, nie koordynowała i nie nadzorowała prac związanych z rozwojem technik komputerowych na terenie Uniwersytetu Wrocławskiego i jest to chyba ogólnokrajowa sytuacja.

Zanim przystąpiono do realizacji polecenia MNiSzW, rozesłano ankietę do wszystkich Instytutów i jednostek dydaktycznych uczelni, której celem było zebranie informacji na temat obecnej sytuacji i planów (na najbliższe dwa lata) związanych z wykorzystaniem metod i środków informatycznych w dydaktyce, wyposażeniem instytutów w środki informatyczne oraz przygotowaniem odpowiedniej kadry dydaktycznej. Na 50 jednostek, które otrzymały ankietę, odpowiedziało 31, a pozostałe albo jej nie zwróciły, albo zwróciły z dopiskiem „nie dotyczy"! Wśród tych 31 instytutów znalazło się wiele takich, które jedynie zadeklarowały chęć włączenia się w nową falę postępu technologicznego, a niemal wszystkie uzależniły realizację swoich planów informatyzacji dydaktyki od dostarczenia i zainstalowania odpowiedniego sprzętu komputerowego oraz przygotowania kadry w dziedzinie specjalistycznych metod informatycznych.

Spośród instytutów przygotowujących się do wprowadzenia metod informatycznych do przedmiotów kierunkowych wybrano dziesięć, które albo rozpoczęły już realizację swoich planów (zwykle w sposób tradycyjny, bez dostępu do komputera), albo rozpoczną nauczanie według nowych programów w ciągu najbliższych dwóch lat. Wśród nich znalazły się Instytuty: Matematyki, Fizyki Doświadczalnej i Teoretycznej, Chemii, Filologii Polskiej, Nauk Geologicznych, Geograficzny, Botaniki, Mikrobiologii, Bibliotekoznawstwa oraz Nauk Administracyjnych. Wybrane instytuty zostały zaproszone do sporządzenia szczegółowych programów zinformatyzowanych przedmiotów. Spośród dostarczonych programów, 13 zostało przesłanych do ministerstwa. Nawet pobieżna analiza zebranych programów pozwala zorientować się, iż z jednym tylko wyjątkiem (kartografii komputerowej) pozostałe przedmioty kwalifikują się raczej do pierwszej grupy, gdyż ich celem jest przede wszystkim nauka programowania, a z dziedzin studiów czerpią jedynie przykłady. Ma to dość oczywiste uzasadnienie, z którego zdają sobie sprawę prowadzący zajęcia. Jakiekolwiek bowiem istotne wykorzystywanie komputerów w przedmiotach zawodowych powinno być poprzedzone zapoznaniem się z elementami informatyki. Chociaż taka właśnie kolejność upowszechniania informatyki jest sugerowana w szczegółowej części Programu, to jednak w zaleceniach końcowych upowszechnianie narzędzi i metod informatycznych wyprzedza zapoznanie się z nimi przez studentów. Autorzy Programu zapewne kierowali się trudnym do obronienia przekonaniem, że stosowanie komputerów przez wykładowców nie wymaga od słuchaczy znajomości tych narzędzi i metod korzystania z nich.

Harmonogram Programu w swej obecnej postaci ma małe szanse terminowej realizacji, ponieważ, jak się wydaje, jego twórcy nie mają pełnego rozeznania rzeczywistej sytuacji w szkołach wyższych.

Przede wszystkim, jeszcze niewielu nauczycieli akademickich uświadamia sobie, jakie możliwości dla dydaktyki tkwią w komputerach i środkach informatycznych. Z kolei, entuzjaści powszechnej edukacji informatycznej powinni zdawać sobie sprawę z tego, że nie wszystkie zajęcia nadają się w równym stopniu do informatyzacji. Dlatego poszczególne kierunki studiów czeka najpierw identyfikacja tych zajęć (lub ich fragmentów), które z pożytkiem dla rezultatów dydaktycznych należy informatyzować, a następnie określenie skali użycia nowych środków. Pociągnie to za sobą zmiany w programach studiów wielu przedmiotów. Identyfikacja i aktualizacja programów wymagają interdyscyplinarnej wiedzy, w tym także dobrego i nowoczesnego przygotowania informatycznego.

Wiele instytutów, a nawet mniejszych jednostek Uczelni, zaopatrzyło się już w komputery, korzystając najczęściej z funduszy pochodzących z prac zleconych. W wielu jednak wypadkach sprzęt został zakupiony bez uprzedniego uwzględnienia potrzeb dydaktyki i jest wykorzystywany w najlepszym razie w pracach badawczych. Jedynie w dwóch Instytutach Uniwersytetu Wrocławskiego, w Instytucie Chemii i Informatyki, utworzono dydaktyczne laboratoria komputerowe. Ponieważ nie każdy instytut stać na kupno sprzętu i prowadzenie laboratorium, nie każdy też potrzebuje takiego laboratorium, więc decyzje o tworzeniu, wyposażeniu i lokalizacji laboratoriów mają być koordynowane na szczeblu uczelni. Planuje się utworzenie sześciu laboratoriów mikrokomputerowych w najbliższych dwóch latach i dalszych sześciu do końca lat osiemdziesiątych. Podstawowym sprzętem w laboratorium mają być lokalne sieci mikrokomputerowe złożone z kilkunastu terminali każda i oparte na komputerach Elwro 800 Jr (w laboratoriach przygotowujących nauczycieli) oraz IBM PC/AT (w laboratoriach przygotowujących do innych zawodów).

Nauczyciele kierunków eksperymentalnych widzą dużą szansę w wykorzystaniu mikrokomputerów, mikroprocesorów i innych elementów elektronicznych do usprawnienia zajęć laboratoryjnych, np. z fizyki i chemii. Szybkiemu jednak postępowi w tej dziedzinie stoją na przeszkodzie wiek i generacja wielu przyrządów używanych w eksperymentach.

Do braków w sprzęcie dochodzi także prawie całkowity brak oprogramowania dydaktycznego, o które zwykle trudniej niż o oprogramowanie systemowe. Nieliczne programy oferowane przez różne firmy lub osoby prywatne, a nazywane dydaktycznymi, mogą być używane do ilustrowania tylko niektórych partii materiału lub metod. Brak jest natomiast całkowicie programów będących całymi jednostkami dydaktycznymi dla wybranych zagadnień kierunkowych.

Program ministerialny zakłada opracowanie tylko jednego ramowego programu zajęć z Podstaw Informatyki dla wszystkich kierunków uniwersyteckich. Trudno jest znaleźć uzasadnienie dla tego zamiaru, przedmiot ten bowiem powinien mieć ścisły związek z kierunkiem studiów (przynajmniej w zakresie ilustracji i zastosowań) i przygotowywać do odbierania i stosowania bardziej zaawansowanej wiedzy informatycznej w przedmiotach zawodowych. Pod tym względem, na przykład, uniwersyteckie studia na kierunkach ścisłych mają więcej wspólnego ze studiami technicznymi, a studia administracyjne — z ekonomicznymi niż oba typy studiów ze studiami filologicznymi. Ponadto nie istnieje takie pojęcie, jak sylwetka absolwenta uniwersytetu i trudno oczekiwać, by ten nowy przedmiot wiele tutaj zmienił.

Reasumując, na kierunkach nieinformatycznych w Uniwersytecie Wrocławskim, celem większości przedmiotów stosujących elementy informatyki jest wyrobienie umiejętności programowania (najczęściej w Basicu) i rozwiązywania prostych problemów obliczeniowych luźno związanych z kierunkiem studiów. Unowocześnienia sposobu prowadzenia tych zajęć oraz informatyzację dalszych przedmiotów poszczególne kierunki uzależniają od szybkiego wyposażenia w sprzęt mikrokomputerowy i wykształcenia odpowiedniej kadry dydaktycznej. Zwłaszcza trudności w zaopatrzeniu w odpowiedni sprzęt stoją na przeszkodzie realizacji planów poszczególnych kierunków studiów i całej uczelni, a także Programu MNiSzW.

Doświadczenia ośrodków zagranicznych

Aby przybliżyć krajowemu środowisku informatycznemu, a zwłaszcza dydaktykom i nauczycielom akademickim, doświadczenia ośrodków zagranicznych, bardziej zaawansowanych we wdrażaniu kształcenia informatycznego, omówiono poniżej najciekawsze fragmenty materiałów z konferencji zorganizowanej w 1983 roku na temat powszechnej edukacji informatycznej studentów uniwersytetów [1]) (uniwersytet jest tutaj synonimem wyższej szkoły i obejmuje swoim zasięgiem wszystkie kierunki kształcenia, porozdzielane u nas w kraju między uniwersytety, politechniki, akademie medyczne, ekonomiczne oraz rolnicze i inne wyższe szkoły). W materiałach zamieszczono tekst referatu wprowadzającego oraz referaty dotyczące powszechnego kształcenia informatycznego na wszystkich kierunkach studiów (6 referatów), zajęć z informatyki na kierunkach ścisłych i przyrodniczych (3), technicznych (5), ekonomicznych (3), medycznych (2) i prawniczych (1). Warto uzupełnić to wyliczenie geografią pierwszej grupy wystąpień. Dotyczyły one

[1]) Tekst szczegółowej recenzji materiałów można otrzymać pisząc bezpośrednio do autora tego artykułu pod adres: M. M. Sysło, Instytut Informatyki, Uniwersytet Wrocławski, Przesmyckiego 20, 51-151 Wrocław.

doświadczeń zebranych w uczelniach Kanady, Japonii (dwa), Izraela, Bułgarii i Włoch (warto zwrócić uwagę, że materiały pochodzą z 1983 r.).

Wszyscy autorzy materiałów przypisują komputerom olbrzymią rolę w obecnej, a zwłaszcza w przyszłej działalności człowieka i społeczeństw. Według Ch. K. Knappera i B. L. Willsa z Uniwersytetu w Waterloo (Kanada), we współczesnych społeczeństwach coraz większą rolę zaczyna odgrywać prawo „kto ma informację, ten ma władzę". Aby zaś w pełni kontrolować procesy związane ze zbieraniem, przechowywaniem, przetwarzaniem i wykorzystywaniem informacji, należy posiąść technologię informacji, której głównym elementem są obecnie mikrokomputery. W sferze dydaktyki, według C. Davida i N. Rafi (Uniwersytet w Tel-Awiwie), komputer jest pierwszym od chwili odkrycia druku w XV w. elementem technologii, mającym szansę zrewolucjonizować proces kształcenia. Dzięki komputerom będzie można wkrótce zrealizować zasadę nauczania, że WSZYSCY studenci (uczniowie) uczą się WSZYSTKIEGO dobrze. Student nie będzie mógł przejść do następnej partii materiału, jeśli nie potrafi wykazać się znajomością poprzedniej partii na odpowiednim poziomie. Ponieważ nie wszyscy uczą się w takim samym tempie, komputery pozwolą porcjować materiał w zależności od uczącego się.

Prezentowane w materiałach poglądy wskazują potrzebę dalszych głębokich studiów nad programami powszechnych zajęć informatycznych dla wszystkich studentów. Nie ulega wątpliwości, że właściwe przygotowanie studentów do przyszłej działalności w społeczeństwie, komputeryzującym obecnie niemal każdą dziedzinę swojej działalności, powinno obejmować także naukę korzystania z możliwości, jakie stwarzają komputery i metody informatyczne. W materiałach nie ma wyraźnego podziału na zajęcia z podstaw informatyki i na komputerowe wspomaganie nauczania przedmiotów kierunkowych. Silny nacisk jest natomiast położony na związek zajęć informatycznych z kierunkiem studiów i specjalizacją.

Większość autorów zgadza się, że zajęcia informatyczne, którymi mają być objęci wszyscy studenci, powinny wyrabiać umiejętności korzystania z procesorów tekstu, dużych baz (banków) danych i ze standardowego oprogramowania specjalistycznego. Przykładowo, zajęcia informatyczne ze studentami matematyki powinny przede wszystkim uzupełniać ich analityczne i algebraiczne rozważania myśleniem algorytmicznym. Matematycy są zadowoleni, gdy potrafią udowodnić istnienie rozwiązania, a jeszcze bardziej, gdy mogą wykazać jego jednoznaczność. Dla informatyka jest to dopiero punkt startowy na drodze do otrzymania rozwiązania w sposób konstruktywny. W zajęciach z fizyki powinno się wykorzystać języki symulacyjne i uwzględnić elementy odpowiednich obliczeń numerycznych. Niektórzy autorzy uważają, że najlepszym źródłem praktycznej wiedzy informatycznej w naukach eksperymentalnych (np. w fizyce i chemii) są problemy świata rzeczywistego, których rozwiązywanie technikami komputerowymi dostarcza najcenniejszej wiedzy metodologicznej i merytorycznej.

W przedmiotach przyrodniczych główny nacisk powinien być położony nie na słownictwo i fakty, jak to się teraz czyni, ale na głębsze zrozumienie istoty nauk przyrodniczych, a tym także teorii naukowych (na przykładach prawdziwych i fałszywych teorii), procesu odkryć i bazy empirycznej. Odpowiednie moduły programowania powinny wyrabiać w słuchaczach podejście określane mianem naukowego, a polegające na zbieraniu faktów, formułowaniu hipotez, ich testowaniu i modyfikowaniu tworzonych teorii. Odpowiednimi środkami informatycznymi dla takiego podejścia mogą być systemy ekspertowe.

W zajęciach ze studentami prawa i administracji należy wyróżnić dwie kategorie powiązań: po pierwsze — technologia dostarcza narzędzi i metodologi na potrzeby legalnej działalności społeczeństw, m.in. w badaniach prawniczych i w administracji państwowej, a po drugie — technologia oraz społeczne skutki jej ekspansji są przedmiotem aktów prawnych. Dodatkowo, studentów administracji, podobnie jak studentów kierunków ekonomicznych, należy uczyć korzystania z arkuszy kalkulacyjnych (ang. spreadsheet). Studenci nauk filologicznych i historycznych powinni zapoznawać się z wyspecjalizowanymi procesorami tekstów oraz bazami danych i środkami do ich tworzenia (np. z systemem dBase).

Nowoczesne środki informatyczne (sprzętowe i programowe) powiny być wykorzystywane na kierunkach inżynierskich, m.in. do nauczania komputerowo wspomaganego projektowania i wytwarzania (CAD/CAM).

Na kierunkach medycznych studenci winni zapoznawać się z obsługą i użyciem tych komputerów, które są wykorzystywane do wspomagania urządzeń medycznych oraz z systemami wspomagającymi proces stawiania diagnozy. W tym drugim wypadku, znajomość języków autorskich i systemów ekspertowych może być pomocna przy budowie własnego wyspecjalizowanego oprogramowania.

W materiałach milcząco przyjęto założenie, że przyszli specjaliści nieinformatycy będą mogli korzystać z oprogramowania bez potrzeby tworzenia go. Założenie to może nie być słuszne w naszych warunkach nawet przez wiele lat, a wtedy, gdy będzie już dostępne potrzebne oprogramowanie, umiejętność programowania może ułatwić posługiwanie się gotowymi pakietami i umożliwić ich niewielkie modyfikacje. Autorzy materiałów nie są jednak zgodni co do roli, jaką należy przyznać umiejętnościom programowania w programach powszechnej edukacji informatycznej. Przykładowo, dwaj autorzy z Uniwersytetu w Waterloo przypisują umiejętnościom programowania w skomputeryzowanym społeczeństwie rolę, jaką znajomość języków klasycznych greki i łaciny odgrywała w wykształceniu człowieka, a w szczególności w nauczaniu języków obcych. Z kolei, J. Hebenstreit z Francji w swoim artykule wprowadzającym przytacza bardzo ciekawe argumenty przeciwko nauczaniu programowania wszystkich studentów. Otóż najczęściej nauka programowania jest uzasadniana tym, że w przyszłości komputery będą wszędzie i studenci powinni nauczyć się z nich korzystać, czyli je programować, oraz tym, że umiejętność programowania wnosi ze sobą pewne ogólne wartości intelektualne, takie jak logiczne myślenie, dobrą organizację pracy i rozumienie algorytmów, czyli efektywnych procedur osiągania określonego celu. Ten pierwszy argument autor nazywa patrzeniem wstecz idąc naprzód, gdyż już teraz mało który użytkownik komputerów (nieinformatyk) sam programuje. O drugim argumencie mówi, że jest w zasadzie kontrprzykładem, gdyż logika języków programowania jest bliższa logice i organizacji komputerów niż logice człowieka. (Czy nie powierdza tego także japoński projekt komputerów piątej generacji, będący kolejną próbą zbliżenia do siebie logik człowieka i maszyny?). Duża różnorodność metod dobrego programowania także świadczy na niekorzyść programowania jako dyscypliny uczącej logicznego myślenia. W tej argumentacji autorowi chodzi przede wszystkim o to, że logika i metodologia programowania są stworzone na pożytek programowania, a nie myślenia w ogólności. Jeśli celem kształcenia nie ma być nauka myślenia na wzór komputera (np. krok po kroku, iteracyjnie lub rekurencyjnie), to należy wyrabiać w studentach raczej intuicję, dociekliwość, wyobraźnię, zdolność oceny i syntezy i inne tego rodzaju umiejętności.

Istotnym aspektem programów zajęć informatycznych poruszanym w materiałach przez wielu autorów jest ich aktualność. Tempo zmian sprzętu, oprogramowania i wyspecjalizowanych zastosowań komputerów zmusza do uwzględnienia w programach zajęć (niemal na bieżąco) efektów tych zmian, które występują wówczas, gdy obecni studenci będą opuszczać mury uczelni i przystępować do pracy zawodowej. W Uniwersytecie Kelo w Jokohamie studenci mogą wybrać sobie przedmiot „Wstęp do Nowej Generacji Komputerów", którego głównym celem jest uświadomienie słuchaczom, jak bardzo nowe systemy komputerowe będą w stanie zmienić w najbliższych latach nasze życie i stosunki społeczne. W szczególności omawiane są różne organizacje komputerów, od vonneumannowskiej, przez prologowe do rozproszonych, oraz ich wykorzystanie w takich dziedzinach, jak: nauki polityczne, socjologia, psychologia, literatura, historia, estetyka i prawo.

W zdecydowanej większości wypadków opisanych w referatach, wyposażenie w sprzęt komputerowy niezbędny do realizacji założonych programów nie przedstawia żadnych trudności. Powszechnie przyjętym modelem laboratorium komputerowego jest lokalna sieć mikrokomputerów będących inteligentymi terminalami dla minikomputera, który pełni także nadzorczą rolę nad wszystkimi użytkownikami oraz prowadzi pełne rozliczenie studenckich kont. Liczba terminali waha się od 30—40 (na 1600 studentów w Uniwersytecie Tsukuba) do ponad 200 (w Uniwersytecie Tokijskim). Jedynym wyjątkiem jest Uniwersytet w Rzymie, którego kłopoty biorą się stąd, że studiuje w nim 50 tys. studentów, a więc około 15 tys. powinno być objętych powszechnym nauczaniem informatyki każdego roku. Chociaż powszechnie uważa się, że idealnym rozwiązaniem jest je-

den student przy terminalu, wielu autorów, obawiając się dehumanizacji procesu nauczania, skłania się ku koncepcji 2—3-osobowych zespołów przy jednym terminalu.

Zdecydowanie więcej miejsca w swoich referatach poświęcają autorzy materiałów oprogramowaniu, które ma wspomagać procesy dydaktyczne. Pierwsze programy dydaktyczne powstawały już w latach pięćdziesiątych i głównym ich zadaniem było zastąpienie nauczyciela, tak jak celem wyprowadzenia komputerów było wtedy wyeliminowanie człowieka. Testy komputerowe, sprawdzające i oceniające opanowanie materiału przez studentów, działały rzeczywiście efektywniej niż człowiek, a jednocześnie były potwierdzeniem teorii Skinnera, według której kształcenie jest niczym innym jak przekazywaniem wiadomości w procesie stawiania pytań i uzyskiwania odpowiedzi.

Dzisiaj takie metody stosowania komputerów w nauczaniu są możliwe jedynie w dziedzinach, które są zbiorami luźno powiązanych ze sobą wiadomości lub procedur postępowania. W połowie lat siedemdziesiątych, gdy koszty mikrokomputerów zaczęły drastycznie maleć, rozpoczął się kolejny okres trwający aż do dzisiaj. Jego główną cechą jest zmiana roli komputerów w dydaktyce — z urządzeń mających zastąpić nauczyciela, na urządzenia asystujące nauczycielom i uczniom w ich pracy. Trzeba sobie jednak zdawać sprawę, że komputerowe wspomaganie nauczania nie jest kolejną odpowiedzią na pytanie, w jaki sposób kształcić, a jedynie nowoczesnym narzędziem dydaktycznym pozostawionym do dyspozycji nauczyciela, który powinien umieć określić właściwy moment, miejsce i zakres stosowania tych nowych możliwości w swojej działalności. Potwierdziły to badania ankietowe wśród uczniów szkół francuskich, którzy nie chcą spędzać przy komputerach więcej niż 10—20% swojego czasu. Z drugiej strony, do procesu kształcenia powinna być włączona cała osobowość uczącego się wraz z jego fantazją i uczuciami, które są całkowicie obce komputerom.

Według J. Hebenstreita, na przygotowanie godzinnego programu dydaktycznego potrzeba 100—300 osobogodzin pracy. Dla usprawnienia swojej pracy programista (projektant) ma do swojej dyspozycji tzw. języki autorskie, np. COURSEWRITER, TUTOR czy PILOT, które ze względu na swoją złożoność sprawiają jednak wiele kłopotów użytkownikom. Ten sam autor ocenia, że 95% programów zwanych dydaktycznymi nie powinno znaleźć się na rynku, a tym bardziej w szkołach.

Program dydaktyczny oznacza tu pakiet oprogramowania umożliwiający realizację całej jednostki zajęć, np. wykładu, ćwiczenia, ćwiczenia laboratoryjnego. Poza takim wykorzystaniem, komputery mogą być pomocne w mniejszym zakresie, np. w uzyskiwaniu dostępu do dużych baz danych (informacji), ilustrowaniu omawianych pojęć, zjawisk i metod lub w symulacji działania algorytmów i obiektów fizycznych. W tym zakresie mieszczą się, na przykład, programy ilustrujące przebieg funkcji lub rozwiązujące zadania rachunkowe. Ładunek dydaktyczny w tych programach jest zdecydowanie niższy niż w programach realizujących całe jednostki zajęć.

Oprogramowanie dydaktyczne jest najczęściej przeznaczone albo dla uczących się, albo dla nauczycieli. Program dla uczących się jest grą dwuosobową między uczniem a komputerem, jego zadaniem jest więc wyeliminowanie nauczyciela. Ten rodzaj oprogramowania jest bardzo skomplikowany, gdyż każda możliwa sytuacja musi być przewidziana i uwzględniona w programie. Oprogramowanie dla uczących się jest najczęściej stosowane poza regularnymi zajęciami, gdyż nie wymaga ingerencji nauczyciela, a więc — na przykład — w domu. Rola nauczyciela w tym wypadku sprowadza się do testowania i oceny dydaktycznego poziomu programów.

Program dydaktyczny dla nauczyciela można uznać za grę trzyosobową między studentami, komputerem i nauczycielem. Oprogramowanie takie ma pomagać nauczycielowi w prowadzeniu zajęć. W szczególności komputer może być użyty przy zgłębianiu trudniejszych partii materiału i do usprawniania prezentacji bardziej skomplikowanych technicznie fragmentów, a także do ożywienia zajęć nowymi sposobami prezentacji treści. Chociaż tworzenie oprogramowania dla nauczycieli jest łatwiejszym zadaniem (program nie musi uwzględniać wszystkich możliwych sytuacji), oprogramowanie to nie było dotychczas zbyt intensywnie rozwijane, gdyż:

- główna uwaga skupiona była przede wszystkim na sposobach zastąpienia nauczyciela komputerem,
- oprogramowanie takie powinno być w dużym stopniu dostosowane do indywidualnych cech nauczyciela, zatem może być mało użyteczne dla innych nauczycieli,
- tylko nieliczni nauczyciele są zdolni sami napisać odpowiednie dla siebie programy.

Powszechność zajęć informatycznych propagowana przez niemal wszystkich autorów materiałów sprawia, że podstawy informatyki stały się nagle największym blokiem zajęć specjalistycznych. Niestety, prawie żadna z uczelni reprezentowanych w materiałach nie dysponuje liczbą specjalistów-informatyków wystarczającą do obsadzenia wszystkich zajęć. Z dyskusji wynika jednak, że ze względu na konieczność uwzględniania w zajęciach informatycznych specyficznych potrzeb kierunków, prowadzącymi powinni być nauczyciele przedmiotów zawodowych gruntownie przygotowani w zakresie informatyki. Niestety, większość kierunków studiów nie jest jeszcze gotowa do samodzielnego prowadzenia tych zajęć. Jako przykład niewłaściwego wyboru podano specjalistę w swojej dziedzinie, który uczy podstaw informatyki, a jego cała znajomość przedmiotu bierze się z doświadczeń zebranych w trakcie programowania na Fortranie. Naturalnym (w wypadku 15 tys. nowych studentów każdego roku w Uniwersytecie Rzymskim — jedynym) podejściem w tej sytuacji jest szybkie przygotowanie kadry nauczycieli kierunkowych — tak właśnie postąpiła większość z reprezentowanych uczelni.

Szkoła Pedagogiczna Uniwersytetu w Tel-Awiwie poszła w swojej działalności o krok dalej, organizując także specjalne kursy z podstaw informatyki dla osób podejmujących decyzje dotyczące wyposażenia placówek nauczania w odpowiedni sprzęt informatyczny i oprogramowania oraz szkolenia kadry dydaktycznej.

Wielu autorów materiałów dzieli się jednak swoimi pesymistycznymi doświadczeniami z kontaktów z nauczycielami przedmiotów kierunkowych. Faktem jest, że nie wszystkie przedmioty w równej mierze mogą być wspomagane komputerowo, należy więc zachować umiar w skali upowszechniania technik komputerowych. Z drugiej jednak strony wielu nauczycieli (ich liczba zwykle rośnie z wiekiem) bardzo niechętnie zmienia formę swoich zajęć, prowadzonych niezmiennie od lat. Siła, z jaką starsi nauczyciele tkwią przy swoich klasycznych metodach nauczania, doprowadziła wielu autorów do przekonania, że w tych wypadkach należy już tylko czekać, aż odejdą oni na emeryturę. Wprowadzanie nowych przedmiotów do programów studiów napotyka także całkiem obiektywne przeszkody, gdyż programy studiów są przeładowane i nie ma w nich miejsca na nowy przedmiot. Zatem wprowadzenie podstaw informatyki nawet w najmniejszym wymiarze będzie związane z zmianami w całej siatce zajęć.

* * *

Na nasz krajowy użytek warto może dodać na zakończenie, że w żadnej z uczelni opisanych w materiałach nie istnieją instytuty czy zakłady (poza obsługą techniczną) specjalnie utworzone do upowszechniania informatyki i jej metod. Upowszechnianie informatyki bowiem nie ma być celem działalności tylko wydzielonych jednostek, ale powinno stać się nieodłącznym elementem naukowego i dydaktycznego warsztatu wszystkich nauczycieli.

Komputerowe nauczanie Ady

Komputerowy kurs języka Ada dla mikrokomputerów IBM PC został opracowany przez firmę Alsys we współpracy z firmą Cassie. Kurs zawiera 27 lekcji. W jedenastu pierwszych wprowadza się podstawowe pojęcia języka, a 16 pozostałych przygotowuje do pełnego wykorzystania jego możliwości. Uczestnik kursu może w każdej chwili wybrać lekcję, w której chce uczestniczyć.

Każda lekcja składa się z części przedstawiającej temat za pomocą przykładów oraz z zestawu ćwiczeń wykonywanych przez użytkownika pod kontrolą programu. Minimalna konfiguracja sprzętowa wymagana przez program zawiera komputer osobisty IBM, 160 KB pamięci RAM, monitor monochromatyczny i kolorowy oraz duże stacje dysków 320 KB. Na ekranie monitora kolorowego wyświetlane są w jednolitej konwencji barw przykłady i ćwiczenia.

Konieczność nauczania Ady

Ada weszła do użycia w końcu 1983 roku. Obecnie dziesiątki kompilatorów tego języka są opracowywane na całym świecie. Wybór administracji amerykańskiej narzucił jej kontrahentom Adę jako język do realizacji licznych projektów wojskowych. Powszechna już dostępność kompilatorów, cechy charakterystyczne języka pozwalające na opracowywanie dużych programów oraz istniejące środowiska programowe prowadzą do coraz szerszego wykorzystania Ady również do realizacji projektów cywilnych.

Ocenia się, że począwszy od 1984 roku w Stanach Zjednoczonych trzeba w tym celu kształcić 10—20 tysięcy informatyków rocznie. Ada jest w tej chwili wykładana na uniwersytetach całego świata, zarówno studentom informatyki, jak i wszystkim innym, których studia i praca zawodowa będą związane z programowaniem.

Zainteresowanie Adą pojawiło się na świecie jeszcze przed ukończeniem prac nad definicją języka. Seminaria, konferencje i wykłady na temat języka Ada prowadzono już od roku 1979 i do chwili obecnej nabrały one dużego znaczenia. W tym samym roku Jean Ichbiah, John Barnes i Rubert Firth przygotowali seminarium, którego zadaniem było przedstawienie Ady do oceny administracji amerykańskiej. To seminarium, nagrane na taśmie wideo, wkrótce stało się produktem handlowym. W ciągu następnych czterech lat zostało przedstawione w całym świecie w języku francuskim i angielskim ok. 30 razy.

Równie szybko pojawiły się książki na temat Ady. Pod koniec 1987 roku było ich (łącznie z tłumaczeniami) kilkadziesiąt (większość w jęz. angielskim, ponad 10 po francusku, kilka po niemiecku i japońsku). Kilkaset artykułów na ten temat ukazało się w prasie zarówno specjalistycznej, jak i przeznaczonej dla szerszego grona odbiorców. Przy wielu towarzystwach informatycznych istnieją grupy szkoleniowe języka Ada.

Metody nauczania coraz częściej polegają na użyciu najnowocześniejszych środków: magnetowidu i komputera. Wykłady i seminaria są rejestrowane na kasetach wideo od 1981 roku. Zaczynają się też pojawiać materiały dydaktyczne do nauczania języka Ada za pomocą komputera: pod koniec 1983 roku opracowano zbiór ćwiczeń i kurs informacyjny dla decydentów.

Elementy dydaktyki

Firmy Alsys i Cassie proponują interakcyjny kurs mikrokomputerowy obejmujący całą problematykę języka Ada na mikrokomputerze. Mikrokomputerowy kurs Ady jest przewidziany dla programistów posiadających już pewne doświadczenie, przy czym nie uczyniono żadnych założeń co do rodzaju języka, jaki znają.

Cechy Ady są przedstawiane stopniowo, za pomocą przykładów odwołujących się do codziennego życia i podstawowych wiadomości każdego człowieka. Typ wyliczeniowy, np.:

```
type
  DZIEŃ is
  (PONIEDZIAŁEK, WTOREK, ŚRODA, CZWARTEK, PIĄTEK, SOBOTA, NIEDZIELA);
type KIERUNEK is (PÓŁNOC, POŁUDNIE, WSCHÓD, ZACHÓD);
```

jest wprowadzany najwcześniej i na nim głównie opierane są przedstawiane przykłady i ćwiczenia.

Typy liczb całkowitych i rzeczywistych są używane tylko wówczas, gdy jest to konieczne. Kiedy możliwy jest wybór, preferowany jest zawsze typ wyliczeniowy. Wszędzie używa się identyfikatorów znaczących (sugerujących znaczenie). Ta sama zasada obowiązuje przy bardziej szczegółowym omawianiu konstrukcji języka: wychodzi się od wyrażeń prostych, zrozumiałych dla wszystkich, aby ukazać oryginalne formy języka, właściwe dla postawionego problemu.

W celu prezentacji niektórych, szczególnie złożonych właściwości języka, w wielu punktach świadomie odstąpiono od wiernego wzorowania się na podręczniku wzorcowym. Przykładowo, wprowadzenie typu tablicowego odbywa się przez wprowadzenie najpierw tablic ograniczonych i ukazanie podstawowych cech tablic, a dopiero później — uogólnienie pojęcia typu tablicowego.

Użycie komputera pozwala konstruować i wprowadzać przykłady stopniowo, a następnie modyfikować je dostosowując do kolejnych etapów nauczania.

Przedstawianie cech języka jest często przerywane prostymi pytaniami kontrolnymi, które pozwalają użytkownikowi ocenić własne postępy. Jeśli pojawią się kłopoty ze zrozumieniem lub przyswojeniem pewnej części materiału, to można uzyskać dodatkowe wyjaśnienia. Użytkownik może także powrócić do początku danego tematu lub przerwać lekcję na jakiś czas, aby samemu uzupełnić wiadomości. Przy końcu każdego tematu i każdej lekcji przedstawione są bardziej skomplikowane ćwiczenia dotyczące jednego lub kilku tematów.

Rola komputera w nauczaniu

Trzeba sobie zdawać sprawę z korzyści, jakie daje użycie komputera w nauczaniu. Szczególnie istotnym czynnikiem jest ilość i forma przekazywanych informacji. Na specjalne życzenie użytkownika tekst jest przedstawiany akapit po akapicie w sposób szczególnie przejrzysty. Każdy akapit składa się z trzech lub czterech wierszy tekstu. Użycie edytora pozwala otrzymać tekst wyrównany prawo- i lewostronnie.

W ustalonym miejscu ekranu stale widoczna jest informacja o aktualnie wyświetlanym tekście: tytuł lekcji i aktualnie omawianego tematu. Dwie inne linie są przeznaczone do wyświetlania komunikatów i poleceń, takich jak proponowane działanie, ocena odpowiedzi użytkownika za zadane pytanie, opis błędów itp.

Program korzysta z dwóch monitorów: monochromatycznego i kolorowego. Wykorzystanie wyświetlania wielobarwnego doprowadziło do ustalenia konwencji dla obu monitorów. Możli-

Konwencje wyświetlania informacji

Rodzaj informacji	Monitor monochromatyczny	Monitor kolorowy
Tekst zwykły	Zielony na czarnym tle	Ciemnoniebieski na tle jasnoniebieskim
Informacje szczególnie ważne	Zwiększona jasność lub inwersja	Zielone tło lub żółte znaki
Błąd	Zwiększona jasność lub inwersja	Czerwone znaki lub czerwone tło
Słowa kluczowe	Podkreślenie	Czarne znaki
Odpowiedzi użytkownika	Inwersja	Białe tło
Kursor	Miganie	Żółty

we sposoby wyświetlania tekstu i dostępne kolory są następujące:
- **na ekranie monitora monochromatycznego:**

wyświetlanie normalne — zielone znaki na czarnym tle,
zwiększona jasność, znaki podkreślone, inwersja — czarne znaki na zielonym tle,
miganie,
- **na ekranie monitora kolorowego:** biały, czarny, jasnoniebieski, ciemnoniebieski, czerwony, żółty, zielony, fioletowy, zwiększona jasność, miganie.

Na monitorze kolorowym kolory znaków i tła są niezależne.

W pierwotnej wersji programu przyjęto konwencję wyświetlania informacji przedstawioną w tabeli. Miganie zostało użyte tylko do oznaczenia miejsca kursora. Migotanie tekstu szybko stałoby się męczące i zmniejszyłoby jego czytelność. Przy wyświetlaniu schematów wykorzystano możliwości semigraficzne, co pozwoliło urozmaicić i uatrakcyjnić prezentację materiału.

Istotną cechą systemu dydaktycznego jest sposób reagowania na odpowiedzi użytkownika. Zastosowany w opisywanym tu programie system analizy odpowiedzi pozwala na sukcesywne naprowadzanie użytkownika na właściwą odpowiedź, nawet gdy jego początkowa znajomość tematu lekcji jest ograniczona. Istnieją więc dwa sposoby uczenia użytkownika: przez wstępne objaśnienia oraz przez korekcję odpowiedzi.

Menu główne zawiera listę wszystkich lekcji podzielonych na trzy grupy: lekcje aktualnie zalecane, lekcje przerobione i lekcje przewidziane do przerobienia w dalszej kolejności. Klasyfikacja zmienia się w miarę przerabiania przez studenta kolejnych lekcji. Pierwsze 11 lekcji następuje po sobie w ustalonej kolejności, graf wyboru następnych lekcji ma postać drzewa. Menu wyszczególnia również lekcje, które zostały przerwane, a powinny być przerobione przed rozpoczęciem aktualnie wybranego tematu. Użytkownik ma jednak pełną swobodę wyboru lekcji, niezależnie od sugestii systemu.

Opisany system dydaktyczny był opracowywany na komputerze osobistym IBM i dla niego pierwotnie przeznaczony. Z powodu braku kompilatora Ady oprogramowanie stworzono w Pascalu pod nadzorem systemu operacyjnego MS-DOS, wykorzystując edytor ekranowy.

Lekcje napisano w języku zdefiniowanym specjalnie dla potrzeb tego systemu. Język ten zawiera dyrektywy określające:
- monitor i miejsce na ekranie, gdzie będzie wyświetlona informacja,
- akcje możliwe do wykonania,
- sposób analizy odpowiedzi.

Tekst lekcji jest tłumaczony na kod pośredni, a następnie interpretowany. Interpreter jest jedyną częścią oprogramowania dostępną dla użytkownika.

W procesie tworzenia systemu wykorzystano doświadczenie firmy Alsys w pracy z językiem Ada oraz osiągnięcia firmy Cassie w zakresie nauczania przy użyciu komputera. Projekt był opracowywany przez siedem osób w ciągu 15 miesięcy.

Opracowała DOROTA INKIELMAN
na podstawie BIGRE, 1983

Terminologia języka Smalltalk-80

Trudności związane z dobrym tłumaczeniem terminologii języka programowania, a szczególnie systemu Smalltalk, mają przynajmniej dwa źródła. Jedno, to niestety już tradycyjny w języku polskim brak wzorców — publikacji dotyczących bardziej zaawansowanych działów informatyki. Drugie, istotniejsze, to potknięcia, niekonsekwencje i udziwnienia samych autorów systemu; być może przyczyną tego jest idea, by system Smalltalk był łatwo zrozumiały dla niespecjalistów (a nawet dla dzieci, według początkowych założeń) i stąd nazewnictwo, w mniemaniu autorów, potoczne i nie nawiązujące do stanu sztuki w dziedzinie języków programowania.

Aby ułatwić rozwiązanie problemów terminologicznych, w artykule nt. Smalltalku (str. 4), odszedłem w wielu miejscach (szczególnie w gramatyce) od terminologii oryginalnej wykorzystując pojęcia dobrze znane w językach programowania (a semantycznie równoważne pierwowzorom smalltalkowym).

Podstawowe pojęcia języka Smalltalk, to obiekt (ang. object), programowanie obiektowe (ang. object-oriented programming), komunikat (ang. message), metoda (ang. method), klasa (ang. class), reprezentant (ang. instance). Oto krótka dyskusja tych terminów (wszystkie definicje lingwistyczne pochodzą z [1] i [2]).

Obiekt jest centralnym pojęciem w Smalltalku. Termin ten oznacza w języku angielskim „coś, co można zobaczyć lub dotknąć: przedmiot" (something that can be seen or touched; material thing) lub „osobę lub rzecz, do której można skierować działanie, uczucie lub myśl" (person or thing to which action or feeling or thought is directed). Projektanci języka Smalltalk wykorzystali to drugie znaczenie i idąc dalej — wprowadzili komunikat — „wiadomość lub żądanie skierowane do kogoś" (piece of news, or a request, sent to somebody) oraz metodę w znaczeniu „sposób robienia czegoś" (way of doing something).

Polskie odpowiedniki **komunikat** i **metoda** są jeszcze odleglejsze od intuicji niż terminy angielskie. Komunikat („oficjalna wiadomość, informacja podana do powszechnej wiadomości, tekst składający się na taką wiadomość") nie ma zupełnie charakteru lokalnego — żądania, polecenia skierowanego do obiektu. Metoda, to „świadomie i konsekwentnie stosowany sposób postępowania dla osiągnięcia określonego celu", lub zupełnie nieadekwatnie, a najbliżej greckiemu praźródłu „sposób badania rzeczy i zjawisk; droga dochodzenia do prawdy".

Te dwa terminy wydają się dużym udziwnieniem użytkownikowi konwencjonalnych języków programowania i są całkowicie niezrozumiałe bez ich wyprowadzenia od centralnego pojęcia obiektu. Komunikat ma na dodatek już utarte znaczenie w programowaniu równoległym i systemach operacyjnych, z czym — uwaga! — Smalltalk nie ma nic wspólnego. Jednak ze względu na powszechność tych terminów w literaturze smalltalkowej, pozostaniemy przy nich (choć metoda to po prostu funkcja lub operacja, a komunikat — to operator lub określenie operacji).

Wracając do centralnego pojęcia obiektu — w języku polskim zwykle rozumie się je bardziej statycznie i biernie („przedmiot, rzecz; przedmiot poznania i działalności człowieka"), często przeciwstawiając podmiotowi („osoba poznająca, przeżywająca, działająca"). Stąd paradoksalnie znacznie lepszym terminem od programowania obiektowego byłoby programowanie podmiotowe — obiekty nie są przedmiotem obliczeń, lecz ich podmiotem, gdyż wyznaczają proces obliczeń, interpretując wysyłane do nich komunikaty. Tak całkowite przeciwstawienie się terminowi oryginalnemu mogłoby jednak prowadzić do nieporozumień, pozostaniemy więc przy programowaniu obiektowym.

Termin klasa w języku angielskim — „grupa mająca cechy tego samego rodzaju; rodzaj, gatunek lub kategoria" (group having qualities of the same kind; kind, sort or division) i polskim — „kategoria przedmiotów, zjawisk wyróżnionych na podstawie wspólnych cech, kategoria przedmiotów jednakowej jakości, gatunek" — są całkowicie zbliżone.

Najtrudniej, jak się okazuje, znaleźć odpowiednik angielskiego terminu instance — „przykład; fakt itp. wspierający ogólną prawdę; dać na przykład" (example; fact, etc. supporting a general truth; give as an example). Dodatkowym wymaganiem powinna być łatwość tworzenia przymiotnika (na przykład, ang. instance variable).

Tłumaczenie instancja (zbliżone z pisownią oryginału) należy odrzucić, gdyż semantycznie jest równoważne jedynie bardzo wyspecjalizowanemu terminowi sądowemu; jest więc całkowicie nieadekwatne.

Porównajmy następujące odpowiedniki: wcielenie, egzemplarz oraz reprezentant (termin przykład wydaje się zbyt potoczny i wieloznaczny). W tym celu przytoczę jeszcze znaczenie klasy jako terminu logicznego — „zbiór przedmiotów wyodrębnionych ze względu na posiadaną przez nią wspólną cechę".

Termin wcielenie („uosobienie, ucieleśnienie czegoś; postać wcielająca jakąś ideę, urzeczywistniająca wyobrażenie pewnych cech lub pojęć") jest adekwatny, choć może zbyt abstrakcyjny (sugeruje coś więcej niż jeden z elementów klasy); trudno utworzyć od niego przymiotnik.

Termin egzemplarz („jedna sztuka z grupy jednakowych przedmiotów; zwykle o przedmiocie, roślinie lub zwierzęciu, radziej o człowieku; okaz") ma właściwie tylko jedną wadę — trudno utworzyć od niego przymiotnik oraz brak pochodnego czasownika.

Jak się można domyślać, przyjąłem tłumaczenie reprezentant jako odpowiednik angielskiego terminu instance. Reprezentant („osoba wchodząca w skład reprezentacji, działająca w czyimś imieniu, reprezentująca kogo lub co; przedstawiciel") jest terminem bliskoznacznym egzemplarzowi, przykładowi; nie jest tak abstrakcyjny jak wcielenie (reprezentanci mając wiele wspólnego ze sobą — bo reprezentują tę samą klasę — nie muszą być identyczni).
Istnieje przymiotnik — reprezentatywny „dobrze reprezentujący kogoś, coś, mający charakterystyczne cechy reprezentanta jakiejś zbiorowości; typowy" oraz czasownik reprezentować („występować, działać w czyimś imieniu; być przedstawicielem, reprezentantem kogoś lub czegoś; być wyrazicielem czegoś, wyrażać coś").

Może dla pełnego obrazu warto zauważyć, że jest świetny odpowiednik łaciński — exemplum; oznacza on jednocześnie: odpis, kopia (brzmienie dosłowne, naśladowanie, odbicie), próba (wzór, modła), przykład, precedens.

Pozostałe terminy języka Smalltalk nie stwarzają już tak użych trudności, bo albo są pochodnymi już przyjętych terminów podstawowych, albo zostały zastąpione utartymi równoważnymi pojęciami. Do tej pierwszej kategorii należą przykładowo następujące zwroty:

- wysyłanie komunikatu (ang. mesage send) — żądanie wykonania metody wysłane do obiektu;
- zmienna reprezentatywna (ang. instance variable) — dobrze reprezentująca, typowa dla klasy, wyrażająca charakterystyczne cechy reprezentanta tej klasy, jej wartości są wyróżnikami poszczególnych reprezentantów tej samej klasy;
- zmienna klasowa (ang. class variable) — zadeklarowana w klasie i dostępna wszystkim jej reprezentantom;
- nadawca-odbiorca (ang. sender/receiver) — odpowiednio obiekt nadający lub odbierający komunikat.

Przykłady terminów drugiej kategorii — równoważników oryginalnych smalltalkowych, to stała (ang. literal), przecią-

żenie (ang. polymorphism), przedefiniować metodę (ang. override a method). W języku Smalltalk nie ma możliwości definiowania stałych — nie ma więc potrzeby rozróżniania stałej i literału (ang. literal). Zamiast terminu wielopostaciowy (ang. polymorphic) można użyć znanego — przeciążenie (selektorów komunikatu) na określenie faktu, że ten sam selektor komunikatu spowodować może wykonanie różnych metod (w zależności od jakiego odbiorcy zostanie nadany ten komunikat). Przez przeciążenie rozumiem właściwość języków programowania polegającą na oznaczaniu cech różnych semantycznie w identyczny składniowo sposób (np. przeciążenie operatorów). Przedefiniowanie metody następuje wówczas, gdy w podklasie definiuje się metodę o nazwie już użytej w nadklasie (do oznaczenia metody, którą w ten sposób przedefiniujemy). Podobnie termin pseudozmienna (ang. pseudovariable) — na określenie zmiennej, której nie wolno przypisać wartości — okazał się niepotrzebny; zamiast niego wprowadziłem stałe i zmienne standardowe.

Na zakończenie wyjaśnienie, dlaczego w przykładach programów zawartych w artykule nt. Smalltalka użyto nazewnictwa angielskiego. Jest to powierzchowna oznaka znacznie głębszego problemu. W systemie Smalltalk występuje wiele nazw, np. w Smalltalku V ponad 2 tysiące! Są to m.in. nazwy klas i selektory metod. Odgrywają one rolę słów kluczowych i operatorów w zwykłych językach programowania, np. Pascalu, i muszą być wykorzystywane w oryginalnej pisowni w programach w Smalltalku. Nazwy te są oczywiście angielskie, bo pomysłodawca i producent systemu pochodzi z USA. Wprowadzając miejscami nazwy polskie, otrzymuje się trudno strawną estetycznie mieszankę językową angielsko-polską. W tym momencie również niełatwe problemy czysto techniczne (brak w alfabetach komputerowych znaków charakterystycznych dla języka polskiego, nieakceptowanie alfabetu polskiego przez analizator leksykalny języka Smalltalk) wydają się drugorzędne.

Zdecydowałem więc, że wszystkie nazwy w przykładach będą po angielsku, a jedynie komentarze po polsku.

ARTUR KRĘPSKI

Instytut Podstaw Informatyki PAN

LITERATURA

[1] Oxford Advanced Learner's Dictionary of Current English, A. S. Hornby, Ed., Oxford University Press, 1981
[2] Słownik Języka Polskiego pod red. M. Szymczaka, PWN, Warszawa, 1978.

Struktura systemu operacyjnego PC-DOS (6)

dokończenie ze s. 20

Funkcja 4EH — szukanie pierwszego wystąpienia danego pliku

Powyższa funkcja przeszukuje skorowidz plików szukając pierwszego wystąpienia danej nazwy (w nazwie mogą być zawarte gwiazdki i znaki zapytania). W momencie znalezienia tej nazwy, w aktualnym buforze dyskowym (DTA) zostaje założony obszar danych, którego budowę przedstawiono w tabeli 3. W procesie przeszukiwania można podać maskę atrybutów umożliwiającą znalezienie, na przykład, plików ukrytych. Należy zauważyć, że zadając atrybut znajdzie się również pliki mające atrybuty o mniejszym ograniczeniu dostępu.

Parametry wejściowe:
AH — 4EH
CX — maska atrybutu używanego w procesie przeszukiwania
DS:DX — adres nazwy pliku zakończonej przez 00H
Komunikaty błędów (przeniesienie ustawione):
AX — 02H, jeśli została podana niewłaściwa ścieżka
AX — 12H, jeśli szukany plik nie został znaleziony.

Funkcja 4FH — kolejne wystąpienie danego pliku

Powyższa funkcja przeszukuje skorowidz szukając kolejnego wystąpienia danego pliku. Konieczne jest uprzednie wywołanie funkcji 4EH. W momencie znalezienia danej nazwy obszar danych założony przy użyciu funkcji 4EH w aktualnym obszarze dyskowym zostaje zmodyfikowany.

Parametr wejściowy:
AH — 4FH

Komunikaty błędów (przeniesienie ustawione):
AX — 02H, jeśli została podana niepoprawna ścieżka
AX — 12H, jeśli szukany plik nie został znaleziony.

Funkcja 56H — zmiana nazwy pliku

Powyższa funkcja dokonuje przemianowania pliku. Należy zauważyć, że zarówno stara, jak i nowa nazwa musi zawierać tę samą specyfikację dysku (explicite lub implicite); jeśli nie zostanie podana, to system przyjmuje ścieżkę najbardziej sensowną. Możliwe jest przemianowanie pliku umieszczonego w jednym skorowidzu na plik umieszczony w innym skorowidzu, na tym samym nośniku danych, przez podanie różnych ścieżek.

Parametry wejściowe:
AH — 56H
DS:DX — adres pierwotnej nazwy pliku ograniczonej przez 00H; może zawierać specyfikację dysku i ścieżkę
ES:DI — adres nowej nazwy pliku zakończonej przez 00H; może zawierać specyfikację dysku i ścieżkę

Komunikaty błędów (przeniesienie ustawione):
AX — 02H, jeśli plik pierwotny nie został znaleziony lub wyspecyfikowana ścieżka nie jest poprawna
AX — 05H, jeśli pierwsza nazwa pliku nie jest poprawną nazwą, lecz jedynie specyfikacją skorowidza lub drugi plik już istnieje albo nie daje się utworzyć
AX — 11H, jeśli usiłowano zmienić nazwę pliku na plik na innym nośniku danych.

* * *

Na załączonym wydruku przedstawiono użycie niektórych funkcji typu Xenix oraz zademonstrowano możliwości przeadresowania logicznych kanałów we-wy (ang. I/O redirection) przy użyciu tych funkcji.

Sysło M. M.: Algorytmy kombinatoryczne i ich efektywność (1). Problem najkrótszego drzewa rozpinającego

INFORMATYKA 1988, nr 3, s. 1

Pierwsza część charakterystyki wybranych metod rozwiązywania problemów kombinatorycznych, zawierająca omówienie sposobu rozwiązania problemu najkrótszego drzewa rozpinającego w sieci.

Sysło M. M.: Combinatorial algorithms and their effectiveness (1)

INFORMATYKA 1988, No. 3, p. 1

First part of selected methods for combinatorial problem solving characteristics, which contains discussion of solution method for minimum spanning tree problem in network.

Sysło M. M.: Kombinatorische Algorithmen und ihre Effektivität (1). Problem des kürzesten Spannbaumes

INFORMATYKA 1988, Nr. 3, S. 1

Erster Teil einer Charakteristik von ausgewählten Lösungsmethoden für kombinatorische Algorithmen, der eine Besprechung von Lösungsmethode des kürzesten Spannbaumes umfasst.

Krępski A.: Język Smalltalk-80 (1)

INFORMATYKA 1988, nr 3, s. 4

Pierwsza część charakterystyki systemu programowania Smalltalk-80, zawierająca omówienie języka Smalltalk-80.

Krępski A.: Smalltalk-80 language (1)

INFORMATYKA 1988, No. 3, p. 4

First part of the Smalltalk-80 programming system characteristics, which contains discussion of the Smalltalk-80 language.

Krępski A.: Smalltalk-80-Sprache (1)

INFORMATYKA 1988, Nr. 3, S. 4

Erster Teil einer Charakteristik von Smalltalk-80-Programmiersystem, der eine Besprechung der Smalltalk-80-Sprache umfasst.

Michalski R. S.: O naturze uczenia się — problemy i kierunki badawcze (2)

INFORMATYKA 1988, nr 3, s. 8

Druga część omówienia zadań i kierunków badań naukowych w dziedzinie uczenia się wspomaganego komputerem.

Michalski R. S.: On nature of learning — problems and research directions (2)

INFORMATYKA 1988, No. 3, p. 8

Second part of discussion about tasks and research directions in computer assisted learning.

Michalski R. S.: Über Natur des Lernens — Probleme und Forschungsrichtungen (2)

INFORMATYKA 198, Nr. 3, S. 8

Zweiter Teil einer Besprechung von Aufgaben und Forschungsrichtungen im Bereich des rechnerunterstützten

Kirpluk M., Sobolewski P.: Implementacja dedukcyjnej bazy danych Holmes (2)

INFORMATYKA 1988, nr 3, s. 12

Druga część charakterystyki dedukcyjnej bazy danych Holmes, zawierająca szczegółowe informacje na temat jej implementacji.

Kirpluk M., Sobolewski P.: Implementation of the Holmes deductive data base system (2)

INFORMATYKA 1988, No. 3, p. 12

Second part of the Holmes deductive data base system characteristics, which contains detailed information about implementation of the system.

Kirpluk M., Sobolewski P.: Implementierung des Holmes-Deduktionsdatenbanksystems (2)

INFORMATYKA 1988, Nr. 3, S. 12

Zweiter Teil einer Charakteristik von Holmes-Deduktionsdatenbanksystem, der detaillierte Informationen über Implementierung des Systems umfasst.

Bielecki J.: Język C. Coraz bliżej normy (1)

INFORMATYKA 1988, nr 3, s. 16

Omówienie wybranych elementów opracowanego przez amerykański instytut normalizacji ANSI projektu normy języka C.

Bielecki J.: C language. Nearer and nearer to standard (1)

INFORMATYKA 1988, No. 3, p. 16

Discussion on selected elements of the C language standard project, which is elaborated by ANSI.

Bielecki J.: C-Sprache. Immer näher zur Norme (1)

INFORMATYKA 1988, Nr. 3, S. 16

Eine Besprechung von ausgewählten Elemente des von ANSI erarbeiteten Entwurfes einer C-Sprache-Norm.

Syfert A., Raznowiecki A.: Struktura systemu operacyjnego PC-DOS (6)

INFORMATYKA 1988, nr 3, s. 18

Ostatnia, szósta część charakterystyki systemu operacyjnego PC-DOS, zawierająca omówienie funkcji zarządzania pamięcią i programem oraz pozostałe funkcje dotyczące plików.

Syfert A., Raznowiecki A.: Structure of the PC-DOS operating system (6)

INFORMATYKA 1988, No. 3, p. 18

Sixth and last part of the PC-DOS operating system characteristics, which contains discussion of memory and program management function, as well as remaining file functions.

Syfert A., Raznowiecki A.: Struktur des PC-DOS-Betriebssystems (6)

INFORMATYKA 1988, Nr. 3, S. 18

Sechster und letzter Teil einer Charakteristik von PC-DOS-Betriebssystem, der eine Besprechung von Funktionen der Speicher- und Programmverwaltung, sowie von restlichen Dateinfunktionen, umfasst.

Warda A.: Pamięci dyskowe Mazovii 1016 (1)

INFORMATKA 1988, nr 3, s. 21

Pierwsza część charakterystyki pamięci dyskowych mikrokomputera Mazovia 1616, zawierająca omówienie napędu i sterownika pamięci dyskietkowej, a także znajdującego się na tym samym pakiecie — sterownika drukarki.

Warda A.: Disk storages of Mazovia 1016 (1)

INFORMATYKA 1988, No. 3, p. 21

First part of the Mazovia 1016 microcomputer disk storage characteristics, which contains discussion of the diskette storage drive and controller, as well as printer controller, fixed on the same plate.

Warda A.: Plattenspeicher von Mazovia 1016 (1)

INFORMATYKA 1988, Nr. 3, S. 21

Erster Teil einer Charakteristik von Plattenspeichern des Mazovia 1016-Mikrorechners, der eine Besprechung von Diskettenspeicherlaufwerk und -Steuerung, sowie von auf dieselber Karte realisierten Druckwerksteuerung, umfasst.

Sysło M.: Uwagi o powszechnym kształceniu informatycznym studentów wyższych uczelni

INFORMATYKA 1988, nr 3, s. 23

Doświadczenia Uniwersytetu Wrocławskiego oraz uczelni zagranicznych w dziedzinie informatycznego kształcenia studentów wyższych uczelni.

Sysło M.: Considerations on general informatics education of university students

INFORMATYKA 1988, No. 3, p. 23

Wrocław University and foreign universities experience in student's informatics education.

Sysło M.: Bemerkungen zur allgemeinen Informatikausbildung der Hochschulstudenten

INFORMATYKA 1988, Nr. 3, S. 23

Erfahrungen der Universität in Wrocław und der ausländischen Universitäten im Bereich der Informatikausbildung von Studenten.

6-letnie doświadczenie softwarowe sprawdzone w ponad 1000 zakładach pracy

TYLKO SYSTEM CSK/32 jeśli zarządzanie Twoim przedsiębiorstwem wymaga minikomputera o mocy obliczeniowej Odry oraz kilkunastu skomputeryzowanych stanowisk pracy

System CSK/32 to:

MIKROKOMPUTERY QUATRO CSK/32
— przystosowane do pracy z kilkunastoma terminalami, pracujące w dowolnym trybie graficznym

OPROGRAMOWANIE
— systemowe WDOS, MIKROLAN
— narzędziowe — MEGA BLOK, TABPLAN, PL-TEKST, BGRAF, TRYS
— aplikacyjne — FK K&K, EM K&K, KADRY, PŁACE

WDROŻENIA, SZKOLENIA
— zespoły ds. wdrożeń i szkoleń pracują w:
Gdyni tel. 248 016, tlx 054792 ● Krakowie tel. 373 573 ● Poznaniu tel. 676 271
Wrocławiu tel. 431 679 ● Warszawie tel. 258 528, tlx 816711
Sopocie — w wyspecjalizowanym Salonie Sprzedaży, ul. Kombatantów 1

KAJKOWSCY™

computer studio kajkowscy

EO/11/88

MIĘDZYWOJEWÓDZKA SPÓŁDZIELNIA PRACY „SIÓDEMKA"

ŁODZ, AL. KOŚCIUSZKI 93,

ZAKŁAD INFORMATYKI I SYSTEMÓW KOMPUTEROWYCH

poleca po **najniższych cenach** w kraju:

— mikrokomputery 8-, 16-, 32-bitowe najwyższej jakości renomowanych firm z całego świata
— urządzenia peryferyjne:
- drukarki — również 24-igłowe i laserowe
- streamery
- napędy dyskowe 3" i 5.25"
- monitory monochromatyczne, kolorowe, EGA, HEGA, VGA
- karty rozszerzenia pamięci
- kontrolery
- dyski twarde typu Winchester 20 MB, 40 MB, 60 MB, 80 MB

— systemy wielodostępne i lokalne sieci mikrokomputerowe:
- MULTI-LINK
- D-LINK
- XENIX

— materiały eksploatacyjne:
- dyskietki 5.25" MD2-D
- dyskietki 5.25" MD2-HD
- dyskietki 3" CF2
- dyskietki 3.5" MF 2DD
- taśmy barwiące do wszystkich typów drukarek STAR i NEC

Termin realizacji zamówień **natychmiast** po złożeniu zamówienia. **Bezpłatnie:** szkolenia, kursy, zestawy oprogramowania narzędziowego i użytkowego — przy dostarczeniu kompletnych systemów.

Proponujemy również na wszelkiego rodzaju mikrokomputery programy wspomagające zarządzanie przedsiębiorstwem:
— system finansowo-księgowo-kosztowy
— system zbytu i zaopatrzenia
— system technicznego przygotowania produkcji
— system materiałowy
— system kadrowy i kadrowo-płacowy (również dla pracowników akordowych)
Wymienione systemy pracują w wersjach sieciowych i wielodostępnych.

Wszelkich informacji udzielamy codziennie (oprócz sobót) w siedzibie Zakładu w Łodzi przy Al. Kościuszki 101, tel. 36-51-00, w godzinach 8—16.

EO/1605/87