7 1987

Informatyka

Prof. Mary M. Shaw o inżynierii oprogramowania

Komunikacja głosowa z komputerem

Struktura systemu PC-DOS

WYDAWNICTWO NOT SIGMA

Cena 150 zł

informatyka

7 1987

Nr 7

Miesięcznik Rok XXII

Lipiec 1987

Organ Komitetu Informatyki MNSZWiT oraz Komitetu Naukowo-Technicznego NOT ds. Informatyki

Materiałów nie zamówionych redakcja nie zwraca

Redakcja: 00-517 Warszawa, ul. Mickiewicza 18 m. 17, tel. 39-14-34

Zakł. Graf. „Tamka". Zam. 0392-1300/87. Obj. 4,0 ark. druk. Nakład 8200 egz. K-82.

ISSN 0542-9951, INDEKS 36124

Cena egzemplarza 150 zł
Prenumerata roczna 1800 zł

NACZELNA ORGANIZACJA TECHNICZNA

WYDAWNICTWO

SIGMA

CZASOPISM I KSIĄŻEK TECHNICZNYCH

00-950 Warszawa
skrytka pocztowa 1004
ul. Biała 4

W NUMERZE: Strona

W NAJBLIŻSZYCH NUMERACH:

- Gerhard Goos o produkcji oprogramowania
- Jan Bielecki o podstawach grafiki w Turbo Pascalu
- Włodzimierz Grudziński o zastosowaniu Prologu w bazach danych
- Jan Bielecki o języku C
- Mariusz Kuc o normie ANSI języka C
- Poglądy Ewy Gurbiel, Heleny Krupickiej i Zdzisława Płoskiego na temat Logo Sinclaira i jego modyfikacji
- Zbigniew Szkaradnik o procesorze Novix NC 400

MARY M. SHAW
Software Engineering Institute
Uniwersytet Carnegie-Mellon
Pittsburgh, stan Pensylwania
USA

Poza programowanie wielkoskalowe — kolejne wyzwania dla inżynierii oprogramowania (I)

Inżynieria oprogramowania zajmuje się poszukiwaniem praktycznych rozwiązań problemów obliczeniowych. W najbliższym pięcioleciu dziedzina ta stanie w obliczu zupełnie nowych zadań takich jak:

- sprostanie wymaganiom społecznym na wytwarzanie oprogramowania dla nieustannie poszerzanych obszarów zastosowań,
- spełnienie coraz większych oczekiwań dotyczących możliwości i jakości oprogramowania,
- przejęcie kontroli intelektualnej nad procesami wytwarzania i pielęgnowania oprogramowania.

Główne wyzwania wynikające z tych potrzeb będą polegały na znacznym poszerzeniu tradycyjnego zakresu inżynierii oprogramowania i na zwiększeniu — o rzędy wielkości — skali konstruowanych systemów. Będzie to wymagało wprowadzenia głębokich zmian w charakterystyce rozwiązywanych problemów i w metodach ich rozwiązywania.

W procesie dojrzewania inżynierii oprogramowania jako dziedziny, niewiarygodnie poszerzył się zakres zadań, które mogą być rozwiązywane przy użyciu komputerów. Stale zwiększa się liczba osób będących użytkownikami komputerów. Niespotykana dotychczas skuteczność technik obliczeniowych spowodowała gwałtowny wzrost potrzeb, w stopniu przekraczającym nasze możliwości wytwarzania oprogramowania mogącego sprostać tym potrzebom. Inżynieria oprogramowania staje więc przed koniecznością poszerzenia zakresu rozwiązywanych problemów jak i skali opracowywanych systemów. Aby to osiągnąć, projektanci oprogramowania muszą stać się bardziej agresywni w upowszechnianiu nowej technologii i w adaptowaniu technik stosowanych w innych dziedzinach.

Niniejszy artykuł rozpoczyna się omówieniem skutków zwiększenia skali rozwiązywanych zagadnień dla inżynierii oprogramowania. Ogólną diagnozę poparto przeglądem problemów, przed którymi obecnie stają twórcy oprogramowania, i faktów wpływających na poszerzenie zakresu i skali potrzeb obliczeniowych. Udowadnia się, że inżynieria oprogramowania musi ulec przeobrażeniu z dziedziny opartej na intensywnym wykorzystaniu pracy (ang. labor-intensive) na dziedzinę opartą na intensywnym użyciu technologii (ang. technology-intensive), korzystającą ze ścisłych modeli i teorii. W zakończeniu zaproponowano wyodrębnienie dwóch nowych zbiorów zagadnień, z którymi przychodzi się stykać, dotyczących umownie programów jako składowych (ang. program-as-component) i programów w roli pełnomocników (ang. program-as-deputy).

WPŁYW SKALI PROBLEMÓW NA INŻYNIERIĘ OPROGRAMOWANIA

Inżynieria oprogramowania dokonała znacznego postępu przechodząc od rozwiązywania małych i prostych zadań do rozwiązywania problemów dużych i dość złożonych. Co więcej, wraz ze wzrostem skali problemów zmieniał się także ich zasadniczy charakter. Zadania, które należało rozwiązać, stawały się bardziej złożone jakościowo, co widać na przykładzie rozwoju od systemów jednostanowiskowych, przez wieloprogramowanie, podział czasu, aż do systemów rozproszonych. Na każdym etapie w historii inżynierii oprogramowania uwaga projektantów oprogramowania skupiała się na określonym zbiorze zagadnień, które można uznać za charakterystykę głównych problemów związanych z wytwarzaniem oprogramowania w tym czasie. Każda nowa generacja systemów stawała się bardziej złożona od poprzedniej, z czym wiązało się powstawanie nowych problemów będące skutkiem tego wzrostu skali. Znaczny wzrost skali systemów i odpowiadające mu zmiany charakteru głównych zagadnień zachodzą w przybliżeniu w cyklu dziesięcioletnim.

Za każdym razem, gdy złożoność systemów oprogramowania wzrasta o rząd wielkości, coraz to inny aspekt za-

Dr Mary Shaw zajmuje stanowisko Chief Scientist w Software Engineering Institute i jest profesorem informatyki w Uniwersytecie Carnegie-Mellon. Pracę w tym Uniwersytecie rozpoczęła po uzyskaniu doktoratu w 1972 roku. Przedtem studiowała w Rice University i pracowała w dziedzinie programowania systemowego i badań systemowych w Research Analysis Corporation i Rice University.

Badania prowadzone przez dr Shaw dotyczą zbyt wysokiego kosztu i zbyt niskiej jakości oprogramowania. Jej podejście do tych problemów jest oparte na metodach inżynierii oprogramowania, ze szczególnym uwzględnieniem języków programowania, technik abstrakcji i metod analitycznych, które wspomagają wytwarzanie niezawodnego oprogramowania.

Jako Chief Scientist w Software Engineering Institute, dr Shaw jest odpowiedzialna za określenie kierunku naukowego tej organizacji. W SEI prowadzi się przede wszystkim prace nad przyspieszeniem tempa wdrażania nowoczesnych technologii w dziedzinie inżynierii oprogramowania. Dr Shaw nawiązuje szerokie kontakty z całą społecznością informatyków zajmujących się badaniami, w celu zidentyfikowania technologii gotowych do wdrożenia i odpowiada za przystosowanie wyników badań zewnętrznych do potrzeb SEI.

Dr Shaw jest autorką lub redaktorką sześciu książek i ponad 70 artykułów lub raportów. Jest starszym członkiem IEEE Computer Society i członkiem Association for Computing Machinery, a także — Society of Sigma Xi, Nowojorskiej Akademii Nauk i Amerykańskiego Stowarzyszenia na rzecz Postępu w Nauce (AAAS). Działa też w Komitecie Technicznym ds. Inżynierii Oprogramowania IEEE, w kolegium redakcyjnym czasopisma IEEE Software i w komitecie egzaminacyjnym na tzw. Graduate Record Examination Computer Science Test. Ponadto jest członkiem licznych ciał doradczych, komitetów programowych i kolegiów redakcyjnych.

czyna odgrywać kluczową rolę w wytwarzaniu systemu stając się wąskim gardłem. W latach pięćdziesiątych, do połowy lat sześćdziesiątych, główny problem polegał na pisaniu zrozumiałych programów, co rozwiązano wprowadzając języki wysokiego poziomu. W latach siedemdziesiątych głównym problemem było zorganizowanie wytwarzania dużych systemów oprogramowania, a jego rozwiązanie osiągnięto przez wprowadzenie narzędzi do programowania wielkoskalowego (ang. programming-in-the-large). Gdy pojawia się nowe wąskie gardło, problemy występujące w mniejszych systemach pozostają nadal aktualne, lecz uwaga projektantów skupia się na nowych zagadnieniach, zasadniczo różnych od występujących dotychczas. Wcześniejsze, mniejsze problemy nie znikają, lecz zwykle stają się podproblemami w obszerniejszych zagadnieniach.

W centrum zainteresowań inżynierii oprogramowania znajdowały się różne zagadnienia. W każdym okresie jednak, największy nacisk był położony na te, które występowały w najbardziej ambitnych przedsięwzięciach programistycznych realizowanych w tym czasie. Często miały one swój początek w systemach wcześniejszych. Jednakże, w systemach wychodzących poza dotychczas zakreślone granice inżynierii oprogramowania, nowe problemy o zasadniczo odmiennym charakterze są najczęściej nie rozpoznawane i nie podejmowane, a twórcy systemów rozwiązują je na zasadzie ad hoc. Dopiero wtedy, gdy nowa problematyka zaczyna regularnie występować w wytwarzaniu oprogramowania, jest wyodrębniana jako oddzielna i warta niezależnych badań.

Najwcześniejsza zmiana w problematyce inżynierii oprogramowania wystąpiła zanim jeszcze przyjęto termin „inżynieria oprogramowania". Pod koniec lat pięćdziesiątych, często niebywałym sukcesem było napisanie programu, który obliczał pożądany wynik. Jeszcze mało upowszechniona była wiedza o organizacji programu i rozumienie działania programów. W połowie lat sześćdziesiątych znaczny wpływ na programowanie wywarło ustalenie, że programy mogą być przedmiotem precyzyjnego, a nawet formalnego wnioskowania. Stwierdziwszy, że algorytmy i struktury danych mogą być projektowane i analizowane niezależnie od ich konkretyzacji w poszczególnych programach, określono je jako cel niezależnych badań [8]. Kolejny krok na drodze ku udoskonaleniu metod programowania wiązał się z uznaniem faktu, że w celu lepszego zrozumienia programów należy je upraszczać [5]. Ostateczne zarzucenie programowania ad hoc na rzecz programowania systematycznego można określić jako przejście do programowania niezorganizowanego (ang. programming-any-which-way) do programowania małoskalowego (ang. programming-in-the-small).

Najbardziej wymownym przykładem zmiany spowodowanej wzrostem skali systemów jest przejście od programowania do zarządzania systemami oprogramowania, co nastąpiło w połowie lat siedemdziesiątych. Wiązało się to ze stwierdzeniem faktu, że konstruowanie dużych i złożonych systemów nie jest tym samym co zadanie pisania małych, pojedynczych programów, nawet jeśli te programy zawierają bardzo dużą liczbę linii kodu. Wytwarzanie dużych systemów wymaga skoordynowania wysiłków wielu osób, pielęgnowania i kontroli wielu wersji i ponownego opracowania nowych wersji w procesie rozwoju systemu.

Problemy związane z tym przejściem wystąpiły w niektórych systemach na wiele lat wcześniej, a wiele z nich zidentyfikowano na tej samej konferencji, na której określono zakres inżynierii oprogramowania [11]. Jednakże, konkretne propozycje rozwiązań stanęły w centrum uwagi dopiero po pewnym czasie. Na początku lat siedemdziesiątych opracowano techniki dekompozycji modułów [12] oraz zbadano metody organizacji pracy zespołów programistycznych [1].

Zadanie opisu struktury dużych systemów i podstawowych różnic między tym zadaniem a opisem programów podjęli DeRemer i Kron [3]. Oni stworzyli terminy, programowanie małoskalowe i programowanie wielkoskalowe, aby wykazać przesunięcie centrum uwagi z problemów rozwiązywanych przez niewiele osób piszących proste programy, na problemy rozwiązywane przez duże grupy osób zajmujących się i zarządzających konstruowaniem dużych zespołów modułów. Znaczenie tego zróżnicowania polega na uznaniu konieczności pojmowania tych dwóch rodzajów problemów w zasadniczo odmienny sposób. Po stwierdzeniu tego faktu, znaczna część społeczności projektantów oprogramowania skierowała swoją uwagę na tę nową problematykę.

W celu wyjaśnienia istoty tych zmian można porównać przesunięcie centrum uwagi dla różnych cech problemów i czynności w programowaniu małoskalowym i wielkoskalowym.

Problemy charakterystyczne. Centrum uwagi przesunęło się z opisu poszczególnych algorytmów na opis sprzężeń (ang. interfaces), struktury systemów i zarządzania grupami osób zaangażowanych w wytwarzanie systemów.

Główne zagadnienia danych. Zainteresowania dotyczące danych przesunęły się ze struktur danych i typów danych na bazy danych, których okresy istnienia (ang. life time) są dłuższe od czasu wykonania poszczególnych programów.

Główne zagadnienia sterowania. Pogląd dotyczący przepływu sterowania (programy są wykonywane jeden raz i kończone) zmienił się na szerszy pogląd dotyczący zespołu modułów obliczeniowych, wykonywanych w sposób ciągły.

Zagadnienia specyfikacji. Zmiana w pojmowaniu zagadnień sterowania doprowadziła do zmian w rozumieniu specyfikacji. O ile programy mające zakończenie można specyfikować jako funkcje matematyczne, to specyfikacja systemu programów wykonywanych w sposób ciągły musi uwzględniać ciąg stanów, przez które przechodzi ten system, i ich efekty uboczne.

Charakter przestrzeni stanów. Przestrzeń stanów fragmentu oprogramowania zmieniła się z małej, łatwo opisywalnej przestrzeni, na dużą przestrzeń stanów o złożonej strukturze.

Zarządzanie. Wzrost skali przedsięwzięć programistycznych spowodował rozszerzenie jednoosobowego kierownictwa na wieloosobowy zespół, poświęcający uwagę wytwarzaniu i pielęgnowaniu dużych systemów.

Narzędzia i metody. O ile w programowaniu małoskalowym używa się struktur danych, kompilatorów, konsolidatorów i programów ładujących, to narzędzia programowania wielkoskalowego obejmują całe środowiska programowe, będące zintegrowanymi zbiorami tych narzędzi, służącymi ponadto do kontroli wersji (ang. version control), zarządzania konfiguracją (ang. configuration management), wytwarzania dokumentacji i generowania raportów.

Jedną z metod uporania się ze wzrostem wielowymiarowości i złożoności jest poszukiwanie sposobów zredukowania tzw. pozornej złożoności problemu. Przykładowo, użyście języka wysokiego poziomu zmniejsza liczbę linii w tekście programu (złożoność pozorna) wymaganą do osiągnięcia określonych właściwości funkcjonalnych (złożoność rzeczywista). Jednakże, nasze możliwości radzenia sobie ze wzrostem złożoności przez uściślanie istniejących metod i rozszerzanie istniejących rozwiązań są dość ograniczone. Niekiedy przychodzi skupić się na rozwiązaniu zupełnie innych zagadnień.

Wzrost wymiarowości problemu o rząd wielkości powoduje, że w procesie produkcyjnym oprogramowania zaczynają odgrywać rolę zupełnie inne jego aspekty, stając się wąskim gardłem. Zjawisko przesuwania środka ciężkości ze wzrostem skali problemu polega ogólnie na podkreśleniu wagi innych czynników w procesie wytwarzania oprogramowania, a nie na odkrywaniu zupełnie nowych zagadnień. Jednakże, zagadnienia o zwiększonym znaczeniu, na ogół, nie są dokładnie zbadane, dlatego konieczne jest poszukiwanie nowych rozwiązań. W tych przypadkach, jak również w przypadkach, gdy komplikują się dobrze znane aspekty problemu, skuteczne jest podejście oparte na wprowadzaniu systematycznych metod, automatyzowaniu rutynowych czynności i redukowaniu pozornej złożoności.

Gdy zostają znalezione rozwiązania tych zagadnień, często okazuje się, że są one również użyteczne w problemach o mniejszej wymiarowości. Przykładowo, wielu programistów używa narzędzi do kontroli wersji w przedsięwzięciach jednoosobowych. Nawet jeśli nowe narzędzia nie są nieodzowne w rozwiązywaniu małych problemów, to mogą być niezwykle użyteczne.

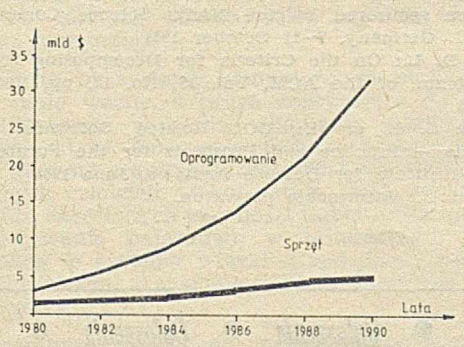
ISTOTA PROBLEMU OPROGRAMOWANIA

W ostatnich kilku latach wzrosło zainteresowanie problemami występującymi w procesie produkcyjnym oprogramowania, prawdopodobnie dlatego, że wskutek wzrostu wymiarowości i złożoności systemów oprogramowania problemy te stały się bardziej widoczne. Obecnie, oprogramo-

wanie jest głównym czynnikiem decydującym o powodzeniu zastosowań systemów zaawansowanych technologicznie, a kluczowym zagadnieniem jest niezawodność oprogramowania. Wynikiem tych zainteresowań było przeprowadzenie licznych badań dotyczących kluczowych problemów oprogramowania. Na ogół, problemy te dotyczą kosztu, zagadnień zarządzania i wydajności oprogramowania. Są one charakterystyczne dla obecnej praktyki, polegającej na intensywnym wykorzystaniu pracy ludzkiej i niedostatecznym wykorzystaniu dostępnej technologii. Stan taki pogłębia się jeszcze wskutek poszerzania się zakresu zastosowań.

Istnieje prawdziwa pokusa, aby wyodrębnić pojedyncze zagadnienie jako „problem oprogramowania" i będąc wyposażonym w odpowiednie środki — przystąpić do jego rozwiązania. Jednakże, taki pogląd byłby zbyt uproszczony. Problemy oprogramowania dotyczą zagadnień ekonomicznych, zarządzania oraz technicznych i są związane z wytwarzaniem, pielęgnowaniem i użyciem systemów. W tym punkcie omówiono niektóre aspekty całego kompleksu problemów oprogramowania.

Oprogramowanie może być kluczowym czynnikiem w produkcji i eksploatacji dużych systemów. Proste defekty oprogramowania mogą powodować kosztowne awarie dużych, złożonych systemów, a przejęcie sterowania systemu przez oprogramowanie może prowadzić do awarii, gdy wystąpią nieoczekiwane warunki lub oddziaływania. Przykładowa awaria pierwszego rodzaju spowodowała, że statek kosmiczny Gemini V wodował około 100 mil od przewidywanego celu [7]. Po prostu, programista pomylił czas gwiezdny z czasem słonecznym i założył, że każdy punkt na kuli ziemskiej powraca do swego położenia względem Słońca po 24 godzinach. Ponieważ takie założenie pomija ruch orbitalny, nagromadzony błąd stał się przyczyną niewłaściwego lądowania. Przykładem awarii drugiego rodzaju był całkowity zanik mocy samolotu Boeing 767, w sierpniu 1983 roku [9]. Komputerowo sterowane schodzenie do lądowania było zoptymalizowane ze względu na wydatek paliwa, lecz mały poziom mocy ułatwił oblodzenie silników, co spowodowało zamknięcie dopływu powietrza niezbędnego do chłodzenia. Wskutek tego silniki przegrzały się i musiały być wyłączone, choć później włączono je ponownie i samolot wylądował bezpiecznie. Inna awaria spowodowana nieoczekiwanymi oddziaływaniami wydarzyła się w elektrowni jądrowej Crystal River, w lutym 1980 roku [10]. Z nieznanych powodów, zwarcie elektryczne doprowadziło do błędnych wskazań niskiej temperatury w reaktorze. Układ sterowania automatycznego odpowiedział na to przyspieszeniem reakcji w rdzeniu. Wskutek tego reaktor przegrzał się, a ciśnienie w rdzeniu wzrosło do niebezpiecznego poziomu i reaktor zaczął samoczynnie się wyłączać. Aby zmniejszyć ciśnienie, komputer otworzył odpowiedni zawór, lecz spadło ono tak szybko, że automatycznie włączył się system uzupełniania ciśnienia, co spowodowało zalanie pętli chłodzenia. Operator zapobiegł dalszym skutkom awarii zamykając ręcznie odpowiednie zawory.

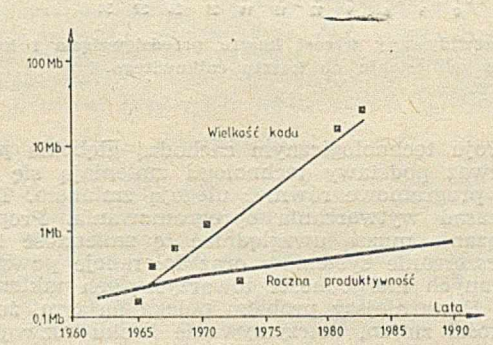


Rys. 1. Przewidywany wzrost kosztów oprogramowania i sprzętu

Oprogramowanie stanowi coraz istotniejszą część zagadnień obliczeniowych. Przeznacza się na nie coraz większy procent ogólnych wydatków na cele komputerowe. Oczekuje się, że w najbliższym czasie nakłady na oprogramowanie gwałtownie wzrosną, zarówno globalnie jak i procentowo. Przykładowo, na rys. 1 przedstawiono przewidywane koszty sprzętu i oprogramowania dla głównych przedsięwzięć w przestrzeni kosmicznej, finansowanych przez Departament Obrony Stanów Zjednoczonych [6].

Technika komputerowa jest tylko jednym wymiarem całego problemu. Wraz ze wzrostem złożoności systemów większego znaczenia nabierają zagadnienia zarządzania i związane z wykonywaniem zawodu. Coraz częściej zauważa się je i uwzględnia, lecz nie przyznano im jeszcze tego samego statusu co zagadnieniom technicznym. Co więcej, wzrostowi wartości oprogramowania — jako własności intelektualnej — towarzyszy wzrost znaczenia czynników ekonomicznych i prawnych.

Do realizacji przedsięwzięć programistycznych brakuje wykwalifikowanego personelu i zbyt wolny jest wzrost jego produktywności. Zarówno koszty wytwarzania, jak i pielęgnacji oprogramowania, są ciągle związane z intensywnym wykorzystaniem pracy ludzkiej, lecz zasoby kadrowe zawodowych informatyków są niewystarczające, aby sprostać tym zapotrzebowaniom. Przyrost tej kadry i produktywności programistów nie jest też dostatecznie szybki. Na rys. 2 porównano zapotrzebowanie na oprogramowanie tylko w jednym obszarze zastosowań [14], z całkowitym wzrostem produktywności programistów [2]. Te braki są szczególnie odczuwalne przy pielęgnacji oprogramowania, co prowadzi do pogorszenia warunków eksploatacyjnych. Za mało jest także wykwalifikowanego personelu, który może pełnić funkcje związane z zarządzaniem.

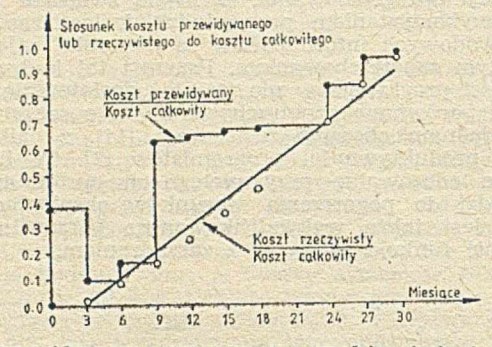


Rys. 2. Wielkość kodu „pokładowego" dla załogowych lotów kosmicznych i roczna produktywność programistów

Obecne technologie mogą okazać się nieprzydatne do rozwiązywania nowych problemów. Podstawowe znaczenie dla systemu oprogramowania ma to, czy system może być w ogóle zaimplementowany, a nie to, jaki będzie koszt implementacji. W inżynierii oprogramowania ciągle opracowuje się nowe narzędzia do wspomagania procesu produkcyjnego oprogramowania. Niektóre z najważniejszych obecnie narzędzi służą do zarządzania dużymi i zmieniającymi się konfiguracjami oprogramowania. Narzędzia te są użyteczne w systemach o dowolnych rozmiarach (w programowaniu małoskalowym), lecz mają kluczowe znaczenie tylko w systemach, których rozmiary osiągają pewien poziom krytyczny (w programowaniu wielkoskalowym). Systemy wielkoskalowe sięgają już granic naszych możliwości uzyskania pożądanych wyników. Ponieważ nasze aspiracje wzrastają szybciej niż nasza produktywność, wciąż stajemy przed zadaniem budowania systemów bardziej złożonych niż te, które powstały dotychczas. Mogą to być bardzo niejednostronne systemy czasu rzeczywistego, w których oprogramowanie jest tylko jednym ze składników, oddziaływującym z elementami elektronicznymi lub elektromechanicznymi. Wskutek tego możemy oczekiwać zapotrzebowania na nowe narzędzia, służące do zmniejszenia pozornej złożoności bardzo dużych systemów. Wydaje się, że te narzędzia będą użyteczne, ale nie kluczowe dla systemów o obecnie spotykanych rozmiarach.

Stosowane dotąd techniki zarządzania przedsięwzięciami programistycznymi są nieodpowiednie. Proces produkcji oprogramowania nie jest tak dobrze poznany jak inne rodzaje produkcji i dlatego jest trudniejszy do planowania, harmonogramowania i zarządzania. Podstawowym problemem jest brak wiarygodnej informacji ilościowej o tym procesie i modeli do jej zinterpretowania. W wielu wypadkach, plany nie są dostatecznie dokładne, aby uwzględnić wszystkie zawiłości procesu produkcyjnego, nie zapewniają solidnych podstaw do podejmowania decyzji lub wręcz są oparte na źle zdefiniowanych wymaganiach. Choć w ostatniej dekadzie poczyniono znaczne postępy w tej dziedzinie, powszechnie używane techniki nie są jeszcze dobrze dostosowane do uwzględniania zmian w wymaganiach i w specyfikacjach, które mają największy wpływ na wzrost kosz-

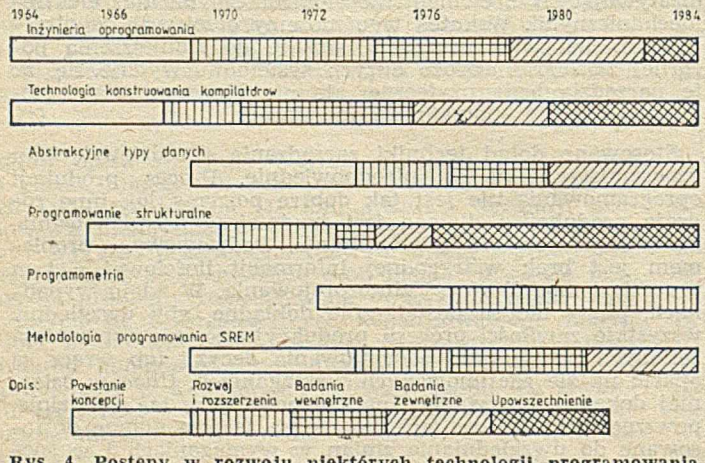
tów i wydłużanie terminów. Na rys. 3 zilustrowano powien problem, często występujący w dotychczasowych strategiach zarządzania. Przedstawione krzywe obrazują przewidywania co do poziomu kosztów (kwadraty) i rzeczywisty przebieg kosztów (kółka) dużego przedsięwzięcia programistycznego. Poszczególne wartości są odniesione do ostatecznego kosztu przedsięwzięcia. Ten przykład wskazuje, że oceny początkowe są bardzo nierealistyczne, a aktualizacja ocen następuje dopiero, gdy wymusza ją praktyka, co zdarzyło się w 9 i 24 miesiącu przedsięwzięcia zilustrowanego na rysunku [4].



Rys. 3. Przewidywany wzrost kosztu przedsięwzięcia i koszt rzeczywisty, w odniesieniu do kosztu całkowitego

W rozwoju technologicznym zachodzą głębokie przemiany. Ponieważ podstawy technologii zmieniają się szybko, narzędzia programowe również ulegają zmianom. To samo dotyczy zasad wytwarzania oprogramowania. Proponowane rozwiązania muszą uwzględniać tę zmienność i umożliwiać rozszerzanie systemów oraz tolerancje powstawania umiarkowanych niezgodności towarzyszących takiemu rozszerzeniu. Poważniejszy problem polega na tym, że wskutek szybkości zmian, dotrzymywanie kroku rozwoju technologii wiąże się z nieustannym nabywaniem coraz to nowej wiedzy o tej technologii. W czasie, gdy przyswaja się obecną technologię i znajduje sposoby jej spożytkowania, powstaje nowsza technologia, a wraz z nią nowe, nieznane problemy.

Nowe technologie są przyswajane dość wolno. Choć rozwój inżynierii oprogramowania jest szybki, to wdrożenie technologii, od powstania koncepcji do upowszechnienia, może trwać dwie dekady [13]. Wskutek użycia technologii powstałej przed dziesięcioma laty, trudne do spełnienia może być osiągnięcie dostatecznej niezawodności i integralności systemu, a sam system może być przestarzały, zanim zostanie zaimplementowany. W wielu instytucjach narzędzia stosowane do użytkowania i wspomagania są niewystarczające, przestarzałe, nieefektywne i często wzajemnie niezgodne. Nie jest praktykowane wielokrotne używanie (ang. reuse) tego samego kodu, a brak normalizacji prowadzi do trudności w zarządzaniu systemu. Choć tempo wdrażania technologii w inżynierii oprogramowania nie jest gorsze niż w niektórych innych dziedzinach, to z pewnością można je znacznie poprawić. Na rys. 4 przedstawiono przebieg wprowadzania do praktyki niektórych znanych technik inżynierii oprogramowania [13].



Rys. 4. Postępy w rozwoju niektórych technologii programowania

Oprogramowanie jest często wiodącym składnikiem zintegrowanego systemu, wbudowanym w jego złożoną, niejednorodną architekturę. Jeśli oprogramowanie jest kluczowym elementem systemu, to wszelkie „poślizgi" w dostawie oprogramowania mają bezpośredni wpływ na opóźnienia i niedociągnięcia we wprowadzeniu całego systemu. Co więcej, mogą się zwiększyć nakłady ponoszone na składniki nieprogramowe; w takim wypadku, bezpośredni koszt spowodowany opóźnieniem dostawy oprogramowania zwiększa się o koszty konserwacji nie używanego sprzętu.

Jednym z głównych problemów jest kontrola intelektualna procesu wytwarzania oprogramowania. Nowoczesne oprogramowanie jest niesłychanie złożone, a jego złożoność wzrasta wraz ze wzrostem oczekiwań dotyczących wydajności systemów. Systemy o wielkości wymaganej w następnej dekadzie będzie można tworzyć z powodzeniem tylko wtedy, gdy struktura oprogramowania, proces produkcyjny i proces pielęgnacji zostaną poznane w sposób precyzyjny i systematyczny.

Podsumowując te rozważania należy stwierdzić, że nie istnieje jeden problem oprogramowania. Są natomiast różnorodne problemy, które łącznie tworzą duży i złożony zbiór zagadnień. Obejmują one przygotowanie narzędzi produkcyjnych i wspomagających, techniki zarządzania, strategie kupowania i sprzedawania oprogramowania i dostępność wykwalifikowanej kadry programistów.

Tłum. i oprac.:
JANUSZ ZALEWSKI

LITERATURA

[1] Baker F. T.: Chief Programmer Team Management of Production Programming. IBM Systems Journal, Vol. 11, No. 1, pp. 56—73, 1972
[2] Boehm B. W.: Software Engineering Economics. Prentice-Hall, Englewood Cliffs (NJ), 1981
[3] DeRemer F., Kron H. H.: Programming-in-the-Large versus Programming-in-the-Small. IEEE Trans. on Software Engineering, Vol. 2, No. 2, pp. 80—86, June 1976
[4] Devenney J.: An Exploratory Study of Software Cost Estimating at the Electronic Systems Division. MSc. dissertation, Air Force Institute of Technology, July 1976
[5] Dijkstra E.: GOTO Statement Considered Harmful. Comm. of the ACM, Vol. 11, No. 3, pp. 147—148, March 1968
[6] EIA: DOD Digital Data Processing Study — a Ten Year Forecast. Electronic Industries Association, 1980
[7] Fox M.: Software and Its Development. Prentice-Hall, Englewood Cliffs (NJ), 1983, pp. 187—188 (cyt. w Software Engineering Notes, Vol. 9, No. 1, January 1984)
[8] Knuth D. E.: Fundamental Algorithms. The Art of Computer Programming, Vol. 1, Addison-Wesley, Reading (MA), 1968
[9] Los Angeles Times, p. 1, 24 August 1983 (cyt. w Software Engineering Notes, Vol, 8, No 5, October 1983)
[10] Marshall E.: NRC Takes a Second Look at Reactor Design. Science, Vol 207, pp. 1445—48, 28 March 1980 (cyt. w Software Engineering Notes, Vol. 10, No. 3, July 1985)
[11] Naur P., Randell B. (eds.): Software Engineering. Report on a conference sponsored by the NATO Science Committee, Garmisch, F. R. Germany, 7—11 October 1968
[12] Parnas D. L.: On the Criteria for Decomposing Systems into Modules. Comm. of the ACM, Vol. 15, No. 12, pp. 1053—1058, December 1972
[13] Redwine S. E. et al.: DOD Related Software Technology Requirements — Practice and Prospects for the Future. IDA Paper P-1788, Institute for Defense Analysis, June 1984
[14] Tomayko J.: Informacja prywatna.

RYSZARD TADEUSIEWICZ
Instytut Automatyki
Akademia Górniczo-Hutnicza
Kraków

Głosowe wprowadzanie informacji do komputera (I)

Komputerowe metody przetwarzania sygnałów są obecnie na tyle rozwinięte i zaawansowane, a tworzone na gruncie sztucznej inteligencji algorytmy rozpoznawania — tak dopracowane od strony praktycznej, że realne i wykonalne jest wykorzystywanie w kontaktach z komputerem dowolnych form sygnałów. Kryterium wyboru może zatem wynikać ze względów psychologicznych, a nie technicznych; wiadomo bowiem, że w złożonych systemach obejmujących zarówno ludzi, jak i maszyny, podstawowe i najtrudniejsze problemy rodzą się „na styku". W komunikacji między człowiekiem a systemem komputerowym najbardziej interesujące wydają się dwie klasy sygnałów: **obrazy i sygnał mowy.**

Rola i znaczenie obrazów przy przekazywaniu informacji, której odbiorcą jest człowiek, są ogólnie znane. Pod każdym względem: szybkości percepcji, pojemności strumienia informacyjnego, możliwości natychmiastowej oceny przyjmowanych informacji, wydobywania strukturalnych właściwości przekazu, a także z punktu widzenia wygody i naturalności — prezentacja informacji w postaci obrazu jest dla człowieka optymalną formą komunikowania. Oczywistym następstwem tego faktu jest rozwój techniki opartej na rozmaitych środkach grafiki komputerowej, wykorzystywanych przy przesyłaniu informacji **od komputera do człowieka.**

Przy przesyłaniu informacji w przeciwną stronę zalety informacji obrazowej nie są już jednak tak oczywiste. Pomimo preferowania wzroku, za pomocą którego odbiera się ponad 90 procent informacji o otoczeniu, człowiek przy komunikacji z innymi ludźmi korzysta z obrazów raczej wyjątkowo (szkice wyjaśniające topografię ulic przy określaniu adresu, rysunki na tablicy podczas wykładu), zasadnicze informacje przekazuje natomiast za pomocą mowy. Człowiek nie posiada bowiem efektywnie działającego efektora graficznego; sporządzanie rysunku jest bez porównania bardziej pracochłonne niż wypowiedzenie nawet bardzo złożonej kwestii. W stosunku do tej niedogodności wszelkie argumenty o zaletach obrazu, jako nośnika określonych treści, tracą znaczenie.

Nie znaczy to, by systemy komputerowej **analizy obrazów** miały być mało ważne, a prace zmierzające do konstrukcji „komputerowego widzenia" stanowiły margines współczesnej informatyki. Znaczenie technik obrazowych wiąże się jednak głównie z automatyzacją różnych stanowisk pracy, na których człowiek wykorzystuje swój wzrok podczas wykonywania określonych czynności (laboratoria, prace montażowe, kierowanie pojazdów), w minimalnym natomiast stopniu można te systemy wiązać z zadaniami komunikacji między człowiekiem a systemem komputerowym (odczytywanie pisma).

W tym ostatnim zagadnieniu niepodzielnie dominują systemy **rozpoznawania mowy.** Wygoda, naturalność, szybkość i niezawodność przekazywania informacji za pomocą mowy stwarzają dla głosowej komunikacji człowieka z komputerem grunt bardzo korzystny i czynią ją z różnych punktów widzenia niezastąpioną. Wprowadzenie informacji za pomocą mowy nie wiąże operatora z żadnym stacjonarnym pulpitem, uwalnia jego ręce, nie wymaga oświetlenia, pozwala wykorzystać istniejące środki telekomunikacji (telefony, radiotelefony), wreszcie może być wykorzystywane w warunkach silnego stresu lub krańcowo niekorzystnych warunków zewnętrznych (przeciążenia lub nieważkość, silne drgania, trudności z orientacją przestrzenną itp.). Nic dziwnego, że głosową komunikacją człowieka z komputerem interesują się ośrodki zajmujące się lotami kosmicznymi i eksploracją dna mórz i oceanów, a także specjaliści wojskowi, opracowujący koncepcje stanowisk pracy dla operatorów sprzętu bojowego (na przykład samolotu lub czołgu). Zainteresowanie tym zagadnieniem wiąże się także z upowszechnianiem techniki komputerowej w zastosowaniach bardziej przyziemnych, przy czym w tym wypadku decydującym argumentem jest brak konieczności specjalnego szkolenia osób korzystających z usług komputera za pomocą dyspozycji wydawanych słownie.

SYGNAŁ MOWY I KOMPUTERY

Oprócz rozważanych dalej zadań automatycznego rozpoznawania treści wypowiadanych informacji istnieją też inne zadania, wiążące temat akustyki mowy z problematyką informatyczną: warto zwrócić na nie teraz uwagę, by oddzielić je od zasadniczego tematu tego artykułu. Często rozważa się zagadnienie przeciwne, to znaczy problem **syntezy mowy,** polegający na tym, że informacja przekazywana przez komputer ma być przedstawiona człowiekowi w formie wypowiedzi, a nie w formie tekstu pisanego czy rysunku. Bez wątpienia systemy atuomatycznej syntezy mowy mogą być użyteczne. Wystarczy wskazać na możliwość przekazywania odpowiedzi komputera przez telefon lub zastosowanie w rozmaitych systemach powiadamiania — masowego, na przykład na dworcu lotniczym, lub indywidualnego, przy dowolnym stanowisku pracy. Można także wspomnieć o możliwości tworzenia nowych stanowisk pracy dla niewidomych oraz o korzyściach, jakie syntezator mowy daje w zastosowaniach komputerów do dydaktyki, zwłaszcza w nauczaniu początkowym. Jednak z punktu widzenia optymalnej wsppółpracy człowieka z komputerem w pracach wymagających szczegółowego i precyzyjnego formułowania wymienianych informacji, syntezator mowy nie ma większych zalet. Może stanowić uzupełniający kanał informacyjny, ale nie może konkurować chociażby z systemami grafiki komputerowej. Z tego powodu automatycznej syntezie mowy poświęca się w informatyce mniej uwagi niż analizie, a szkoda, gdyż synteza jest znacznie łatwiejsza niż rozpoznawanie i możliwe jest przy użyciu dość prostych (i dostępnych handlowo) środków osiąganie dobrych wyników w tej dziedzinie. Można powiedzieć, chociaż jest w tym nieco przesady, że problem automatycznej syntezy mowy przestał już istnieć jako problem badawczy. Otwarta jest natomiast kwestia, jakie urządzenia zastosować, aby uzyskiwać możliwie dobrą jakość procesu syntezy i jego efektów przy angażowaniu możliwie niewielkich kosztów i przy minimalnym obciążeniu komputera, który syntezator mowy ma wykorzystywać jako urządzenie wyjściowe.

W odróżnieniu od **syntezy mowy,** problem jej **analizy** jest nadal otwarty pod względem badawczym. Wynika to z ogromnej złożoności i wieloaspektowości problemu rozpoznawania mowy. Bariery tej złożoności nie udało się dotychczas pokonać dla żadnego języka naturalnego na świecie, chociaż od blisko pół wieku podejmuje się intensywne starania w tej dziedzinie, angażując znaczne środki na badania.

To, co zostało napisane wyżej, wywoła prawdopodobnie sprzeciw u niektórych Czytelników. Wszak nie trzeba być specjalistą, a wystarczy czytać codzienną prasę lub „Młodego Technika", by dowiedzieć się, że istnieją i są sprzedawane różne urządzenia, sterowane sygnałem mowy, a także reklamuje się systemy służące do wprowadzania głoso-

wego informacji do komputera. Są to wprawdzie na ogół systemy działające dla innych języków, niż nasz rodzimy, dla języka japońskiego, angielskiego, niemieckiego, francuskiego...

ROZPOZNAWANIE MOWY

Nieporozumienie polega na tym, że termin **rozpoznawanie mowy** może być rozumiany rozmaicie. Można stwierdzić, że znacznie łatwiej i bardziej jednoznacznie da się określić zadanie rozpoznawania obrazów, niż analogiczne zadanie dla sygnału mowy. Jeśli uznać za rozpoznawanie mowy skuteczne rozróżnianie kilku starannie wybranych i odpowiednio zestawionych haseł lub słów kluczowych, to zadanie takie jest stosunkowo proste. Jeśli dodatkowo narzucić wymagania, by rozpoznawane elementy były wypowiadane w izolacji, szczególnie starannie i przy zachowaniu odpowiednich warunków zewnętrznych (brak zakłócających szumów), to zadanie rozpoznawania jeszcze się upraszcza.

Zupełnie odmiennie wygląda jednak to samo zadanie, gdy zdjąć część lub nawet wszystkie z wymienionych ograniczeń. Zadanie utrudnia zarówno otwarcie go na nieograniczony **zbiór osób mówiących**, jak i przyjęcie nieograniczonego **zbioru rozpoznawanych słów**. Zasadniczym jednak utrudnieniem jest dopuszczenie **niestarannej wymowy**. Ludzka doskonałość w rozpoznawaniu mowy stanowi tu czynnik maskujący skalę rzeczywistych trudności. Skoro człowiek potrafi rozpoznawać słowa wypowiadane niestarannie, z niewłaściwą artykulacją czy też zlewające się w szybkiej mowie ze sobą, więc tego samego oczekuje od systemu automatycznego.

Tak więc doniesienia o stosowanych i oferowanych handlowo systemach rozpoznawania mowy dotyczą — jak dotąd — systemów zdolnych do rozpoznawania oddzielnych słów, starannie wymawianych i tak dobranych, by ich rozróżnienie nie sprawiało kłopotów. W dodatku wymaga się, by **wzorce** tych samych słów, wypowiadanych przez osoby mające pracować z systemem, zostały uprzednio zarejestrowane w celach porównawczych. Przy tak postawionym zadaniu skuteczne rozpoznawanie jest możliwe przy użyciu niezbyt złożonych środków i metod, a efekty mogą być — z punktu widzenia praktyki — zadowalające. Natomiast zadanie rozpoznawania otwartego zbioru dowolnych wypowiedzi w dowolnych warunkach i przy dowolnej liczbie osób mówiących — stanowi na razie problem nie do pokonania dla współczesnej techniki.

CELE PROCESU ROZPOZNAWANIA

Przed przystąpieniem do bardziej szczegółowych rozważań warto uzupełnić dotychczas podane fakty o jeszcze jedno stwierdzenie. Otóż jedną z podstawowych trudności, którą trzeba przezwyciężyć przy pisaniu programów rozpoznających mowę, jest kwestia **różnic osobniczych** głosów poszczególnych osób. W istocie, sygnał mowy, odbierany i wprowadzany do komputera, niesie różne informacje. Należy wyróżnić wśród nich informacje semantyczne, czyli treść wypowiadanych słów, ale oprócz tej zasadniczej informacji jest jeszcze **informacja osobnicza**, określająca kto wypowiedział te słowa, informacja **emocjonalna** (w jakim nastroju był mówiący), informacja **socjologiczna** (z jakiej grupy społecznej pochodzi, jakie ma wykształcenie), informacja **medyczna** (dla wielu chorób charakterystyczne cechy, umożliwiające diagnozę, są zawarte w sygnale mowy, zwłaszcza gdy chodzi o krtań lub deformację jamy ustnej), oraz wiele innych, na przykład informacja o regionie kraju, w którym prawdopodobnie mieszka osoba mówiąca. Te i inne źródła zmienności sygnału mowy powodują, że do rozumienia treści wypowiedzi trzeba skutecznie odfiltrować z niej inne, nieistotne w tym wypadku, informacje. Jest to jedno z najtrudniejszych zadań w procesie rozpoznawania mowy.

Czasem jednak celem rozpoznawania jest jedna z wymienionych cech „zakłócających", a sens i treść wypowiedzi mogą w tym rozpoznawaniu pomagać lub przeszkadzać. Najczęściej rozważa się zadanie określania indywidualnych cech głosu osoby mówiącej w celu jej identyfikacji. Można tu zresztą wyróżnić przynajmniej dwa oddzielne zadania: problem **rozpoznania** osoby mówiącej, kiedy znane są wzorce różnych głosów i trzeba z maksymalnym prawdopodobieństwem ustalić, który z nich jest aktualnie rejestrowany, oraz problem weryfikacji osoby mówiącej, kiedy z góry podane jest, kim powinna być osoba mówiąca, a trzeba upewnić się na podstawie analizy jej głosu, że jest to istotnie właściwa osoba (głos jako czynnik umożliwiający dostęp do określonych pomieszczeń, urządzeń lub informacji). W prezentowanym artykule pominięto jednak zagadnienia identyfikacji i weryfikacji głosów, jak również problemy rozpoznawania mowy dla potrzeb medycznych i badania korzystające z komputerowej analizy mowy osoby badanej w celu ustalenia jej stanu emocjonalnego (na przykład stopnia znużenia operatora na stanowisku roboczym wymagającym ciągłego skupienia uwagi). Pozostaje czysty, najczęściej stawiany i najtrudniejszy do osiągnięcia cel: znalezienie takiej metody analizy sygnału mowy, by możliwe było bezbłędne określenie treści wypowiedzi.

ETAPY PROCESU ROZPOZNAWANIA

Rozpoznawanie mowy jest procesem wieloetapowym, a jego struktura, pomimo częstych różnic zdań co do szczegółów realizacji poszczególnych etapów, jest w zasadzie ustalona. Wyróżnia się więc zwykle:

- wprowadzanie sygnału mowy do komputera (wraz z ewentualnym wstępnym przetwarzaniem);
- opisywanie sygnału za pomocą cech i parametrów nadających się do wykorzystania przy rozpoznawaniu;
- segmentację ciągłego sygnału mowy na odcinki odpowiadające rozpoznawanym elementom;
- rozpoznawanie elementów mowy;
- łączenie rozpoznanych elementów w jednostki wyższego rzędu i kontekstowe korygowanie błędów powstałych na wcześniejszym etapie;
- analizę całych wypowiedzi w celu wydobycia ich sensu w kontekście zastosowania, któremu proces rozpoznawania służy.

Niektórzy badacze dodają nowe etapy do tego schematu; na przykład rozważa się często oddzielnie moduł analizy syntaktycznej, wyodrębniając go z analizy sensu wypowiedzi — głównie na bazie sukcesów w automatycznym rozbiorze gramatycznym języków formalnych, przy konstrukcji translatorów języków programowania. Inni badacze eliminują pewne etapy (na przykład podział na segmenty) lub łączą je w większe całości (na przykład wydobywanie cech z rozpoznawaniem lub rozpoznawanie elementów z rozpoznawaniem całych wypowiedzi). W sumie jednak podany schemat analizy jest zwykle zachowany i może stanowić wygodną postawę do systematycznego omówienia procesu rozpoznawania mowy, a także może być wygodnym punktem wyjścia do wyjaśnienia przyczyn niepowodzeń (lub lepiej niepełnego powodzenia usiłowań budowy systemu rozpoznawania mowy).

* * *

W drugiej części artykułu zostaną zaprezentowane szczegóły naszkicowanego wyżej schematu systemu rozpoznawania mowy. Prezentacja ta będzie oparta na wynikach uzyskanych w Instytucie Automatyki Akademii Górniczo-Hutniczej. Według podobnych zasad i z podobnymi wynikami pracują także inne zespoły zajmujące się w Polsce problematyką rozpoznawania mowy: Instytut Podstawowych Problemów Techniki PAN (w Warszawie i w Poznaniu), Instytut Akustyki i Telekomunikacji Politechniki Wrocławskiej i zespół Instytutu Informatyki Uniwersytetu Warszawskiego. W wymienionych pracowniach powstało już kilka udanych systemów, które rozpoznają mowę polską na skalę laboratoryjną. Problematykę automatycznego rozpoznawania mowy integruje Polskie Towarzystwo Fonetyczne.

Stan badań nad automatycznym rozpoznawaniem mowy polskiej można przedstawić następująco. Stworzono bardzo solidne podstawy teoretyczne i doświadczalne do budowy systemu rozpoznawania obszernych zbiorów słów języka polskiego (prof. Jassem, IPPT Poznań), a także zbudowano praktycznie działające systemy rozpoznawania (dla potrzeb automatyki) ograniczonego zbioru słów (dr Kot, AGH Kraków, dr Grocholewski, Politechnika Poznańska). Przeprowadzono bardzo zaawansowane badania nad wykorzystaniem sygnału mowy do celów diagnostyki medycznej (prof. Kacprowski, IPPT Warszawa) oraz nad identyfikacją i weryfikacją głosu osoby mówiącej (prof. Majewski, Politechnika Wrocławska). We wszystkich ośrodkach zajmujących się problematyką sygnału mowy podejmowano na ogół udane próby konstrukcji syntezatorów sygnału mowy (prof. Kacprowski — IPPT Warszawa, dr Myślecki — Politechnika Poznańska, prof. Jassem — IPPT Poznań, dr Kiełczewski — Uniwersytet Warszawski). Udane próby budowy systemów automatycznej syntezy mowy polskiej podejmowano także w ośrodkach nie mających tradycji w

dziedzinie badań nad sygnałem mowy, dysponujących natomiast kadrą wytrawnych informatyków (np. prace Instytutu Maszyn Matematycznych).

Przed prezentacją wyników badań warto podać orientacyjne oszacowania wymagań, jakie zadania rozpoznawania mowy stawiają systemowi komputerowemu. Oszacowania te mają charakter prowizoryczny, gdyż rozwój metod przetwarzania sygnału mowy doprowadzi zapewne do powstania specjalizowanych rozwiązań sprzętowych, w których znaczną część operacji przetwarzania i rozpoznawania będzie można realizować znacznie mniejszym kosztem sprzętowym (pamięć) i programowym (czas obliczeń). Zanim to jednak nastąpi można przyjąć jako punkt wyjścia następujące wymagania:

- pojemność pamięci systemu 1—2 MB (przy mniejszej pojemności pamięci operacyjnej możliwe jest wykorzystywanie dysków, ale wtedy nie ma praktycznie mowy o uzyskaniu rozpoznawania w czasie rzeczywistym);
- liczba obliczeń około miliona operacji arytmetycznych i logicznych na jedną sekundę rozpoznawanego sygnału (liczba ta zależy oczywiście od stopnia złożoności rozpoznawanego sygnału).

Wymagania te wyznaczają orientacyjną moc obliczeniową komputera, którą trzeba zaangażować w proces rozpoznawania, by uzyskać zadowalające wyniki. W warunkach krajowych badacze muszą zadowalać się sprzętem o znacznie skromniejszych możliwościach. Przykładowo, zespół IPPT z Poznania prowadzi badania o bardzo wysokim poziomie naukowym, dysponując komputerem Mera 300 z pamięcią o pojemności 8 KB. W przyszłości można oczekiwać, że moc obliczeniowa wymagana do realizacji zadań związanych z rozpoznawaniem mowy może być zasadniczo ograniczona przez zastosowanie specjalizowanych preprocesorów sygnałowych, gdyż znakomita większość wymagań pamięciowych i związanych z dużą liczbą operacji arytmetycznych procesów przetwarzania jest związana z sygnałem w jego najbardziej pierwotnej, nie przetworzonej postaci. W dalszych etapach procesu przetwarzania rośnie wprawdzie złożoność wykonywanych operacji, lecz towarzyszy temu tak radykalna redukcja liczby danych (wynikająca z redukcji i agregacji informacji na kolejnych szczeblach), że łączny efekt prowadzi do zmniejszenia wymagań.

LITERATURA

[1] Borodziewicz W.: Cyfrowe przetwarzanie sygnałów z wykorzystaniem teorii systemów rozmytych. Rozprawa doktorska, AGH, Kraków, 1986
[2] Fant G., Tatham M. A. A.: Auditory Analysis and Perception of Speech. Academic Press, New York, 1976
[3] Flanagan J. L.: Speech Analysis, Synthesis and Perception. Springer Verlag, Berlin, 1965
[4] Izworski A.: Globalna metoda segmentacji zredukowanego widma sygnału mowy. Rozprawa doktorska, AGH, Kraków, 1986
[5] Jassem W.: Podstawy fonetyki akustycznej. PWN, Warszawa, 1973
[6] Jassem W. (ed.): Speech analysis and synthesis. PWN, Warszawa, 1968
[7] Kacprowski J.: Teoretyczne podstawy metod automatycznego rozpoznawania samogłosek. Archiwum Akustyki, nr 2, 1967
[8] Kot L.: Ocena przydatności analizy pasmowej do rozpoznawania prostych elementów mowy polskiej przez maszynę cyfrową. Rozprawa doktorska, AGH, Kraków, 1980
[9] Oppenheim A. V.: Sygnały cyfrowe — przetwarzanie i zastosowania. WNT, Warszawa, 1982
[10] Peciak J.: O utajnianiu mowy bez tajemnic. Wydawnictwo MON, Warszawa, 1980
[11] Sapożkow M. A.: Sygnał mowy w telekomunikacji i cybernetyce. WNT, Warszawa, 1966
[12] Tadeusiewicz R.: Sygnał mowy. WKiŁ, Warszawa (w druku).

JAN MADEY
Instytut Informatyki
Uniwersytet Warszawski

POLSKIE TOWARZYSTWO INFORMATYCZNE

Problematyka systemów operacyjnych na przykładzie systemu Unix (2)

W drugiej części artykułu omówiono wybrane zagadnienia zarządzania zasobami.

ZARZĄDZANIE ZASOBAMI

W systemach komputerowych występują różnego rodzaju zasoby, którymi zarządza system operacyjny. Zarządzanie to polega na takim rozdzielaniu zasobów pomiędzy użytkowników, aby każdy z nich miał wrażenie, że pracuje na własnym komputerze. Taki zindywidualizowany komputer nazywa się często **wirtualnym.** Jest on dostosowany do bieżących potrzeb użytkownika, a różni się od rzeczywistego komputera przede wszystkim wolniejszą pracą (bo procesor jest dzielony pomiędzy wiele osób). Zasoby komputera wirtualnego nazywa się **zasobami wirtualnymi.** Komputer wirtualny daje użytkownikowi niekiedy większe możliwości od rzeczywistego. Na przykład, **pamięć wirtualna** jest dużo większa od rzeczywistej pamięci operacyjnej. Efekt ten uzyskuje się dzięki temu, że odpowiedni moduł systemu operacyjnego nie tylko bezkolizyjnie dzieli pamięć operacyjną pomiędzy użytkowników, ale również zapewnia automatyczne przesyłanie informacji między pamięcią operacyjną a pomocniczą. Stwarza to każdemu użytkownikowi złudzenie posiadania do własnej dyspozycji prawie nieograniczonej pamięci operacyjnej (limitowanej wielkością przydzielonej pamięci pomocniczej).

Zasoby komputerowe są to obiekty dzielone (współużywane) przez użytkowników oraz przez sam system operacyjny. Tradycyjnie wyróżnia się przy tym **zasoby sprzętowe** oraz **zasoby programowe.** Do pierwszej grupy zalicza się podstawowe składowe sprzętowe komputera, tj. **procesory** (procesor centralny, kanały, urządzenia wejścia-wyjścia) i **pamięci** (obszary na różnego rodzaju nośnikach informacji). Drugą grupę stanowią programy usługowe oraz struktury danych tworzone przez użytkownika lub przez system **(pliki).** Ponadto, w niektórych systemach do zasobów programowych zalicza się jeszcze pewne bufory, służące do komunikacji między procesami. Niezależnie od tej klasyfikacji, zasoby dzielą się na **trwałe** (niezużywalne) i **tymczasowe** (zużywalne). Przykładem trwałego zasobu jest urządzenie fizyczne, przykładem tymczasowego — komunikat.

Obsługując użytkowników, system operacyjny dla każdego z nich tworzy, utrzymuje, a następnie likwiduje komputer wirtualny. Przy jednoczesnej obsłudze wielu klientów system musi więc bezkolizyjnie i bezpiecznie dzielić zasoby. W tym celu określone fragmenty systemu **(tzw. zarządcy zasobów)** prowadzą i realizują politykę przydziału zasobów oraz śledzą ich stan i stopień wykorzystania. Do zadań zarządcy zasobów należy zatem:

- śledzenie na bieżąco stanu danego zasobu bądź jego części („wolny", „zajęty") oraz ewentualne śledzenie stopnia wykorzystania tego zasobu w określonym przedziale czasu,
- ustalanie strategii przydziału danego zasobu (kto, kiedy, jak często, na jak długo i ile otrzyma),
- przydzielanie zasobu według przyjętej strategii,
- odzyskiwanie zasobu.

W dwóch następnych punktach przedstawiono zwięźle wybrane fakty o zarządzaniu dwoma ważnymi zasobami: **procesorem centralnym** oraz **pamięcią operacyjną.** Omówienie choćby skrótowe pozostałych rodzajów zasobów przekracza zakres niniejszego artykułu, jednak, pewne uwagi na temat systemu plików zawarto w punkcie poświęconym Unixowi.

Procesor

Zarządzanie procesorem jest realizowane w jądrze systemu. Każdy z procesów, który ubiega się w danej chwili o procesor jest umieszczany **w kolejce procesów gotowych.** Procesy systemowe mają przy tym zawsze wyższy priorytet od procesów użytkowników, tzn. przede wszystkim im jest przydzialany procesor (w kolejności ustalonej zwykle dla danego systemu). Natomiast konkurujące między sobą procesy użytkowników są obsługiwane zgodnie z pewną strategią.

Strategie przydziału procesora dzielą się na dwie podstawowe grupy: **z wywłaszczaniem** (ang. preemptive scheduling) oraz **bez wywłaszczania.** W pierwszym wypadku procesor może być odebrany procesowi, a w drugim — nie. Przykładem strategii bez wywłaszczania jest obsługa procesów według kolejności ich pojawiania się w kolejce procesów gotowych. Strategia ta jest często nazywana **FIFO** (ang. first-in-first-out). Przykładem strategii z wywłaszczaniem jest tzw. **karuzelowa obsługa procesów**, zwana **RR** (ang. round-robin). Do jej realizacji jest niezbędne urządzenie fizyczne zwane czasomierzem, który powoduje przerwania co określony kwant czasu. Po każdym takim przerwaniu procesor jest przydzialany kolejnemu gotowemu procesowi, a proces wywłaszczony jest umieszczany na końcu kolejki. Wielkość kwantu czasu jest zwykle ustalona dla danej instalacji, ale są też systemy ze zmiennym kwantem, którego wielkość jest uzależniona, na przykład, od obciążenia systemu.

Pamięć operacyjna

Metody przydziału pamięci operacyjnej są przedmiotem wszechstronnych badań od ponad 20 lat i uzyskano już wiele ciekawych wyników teoretycznych oraz praktycznych. Organizacja pamięci, jej parametry (pojemność i szybkość dostępu) oraz środki techniczne realizacji uległy bardzo istotnej ewolucji. Podobnie, ewolucji uległy strategie przydziału pamięci. Należy zauważyć jednak, że niezmienna pozostała zasada wiążąca ze sobą pamięć z procesorem: **nie ma sensu przechowywać w pamięci operacyjnej procesów, które nie są bezpośrednimi kandydatami do przydziału procesora, i odwrotnie, nie ma sensu przydzielanie procesora takiemu procesowi, którego przestrzeń adresowa nie jest w pamięci operacyjnej.**

Początkowo rozwijano metody, które zakładały obecność w spójnym obszarze pamięci operacyjnej całej przestrzeni adresowej wykonywanego procesu. Ograniczając się do systemów wieloprogramowych, należy wspomnieć o następujących ważnych strategiach:

— statyczny podział na strefy,

— dynamiczny podział na strefy,

— wymiana (ang. swapping).

W pierwszym wypadku pamięć operacyjna jest podzielona w czasie inicjowania systemu operacyjnego na stałe strefy, tzn. rozłączne i spójne obszary. Procesowi jest przydzielana jedna strefa, w ramach której są wykonywane wszystkie obliczenia tego procesu (liczba stref ogranicza więc stopień wieloprogramowości). Łatwo jest przy tym zagwarantować **ochronę pamięci** — wystarczą dwa rejestry graniczne, których wartości określają zakres dopuszczalnej zmienności adresów dla danego procesu. Strategia ta, prosta w realizacji, ma istotną wadę: często duże obszary pamięci są niewykorzystane. To niekorzystne zjawisko nazywa się **fragmentacją pamięci.** Wyróżnia się przy tym

fragmentację zewnętrzną (pozostają niewykorzystane wolne strefy, gdyż są za małe na potrzeby procesów gotowych) oraz **fragmentację wewnętrzną** (część strefy nie jest wykorzystywana przez realizowany w niej proces).

W celu wyeliminowania fragmentacji wewnętrznej opracowano drugą z wymienionych strategii. Strefy są w niej tworzone dynamicznie, w chwili pojawienia się nowego procesu. Otrzymuje on wówczas tyle pamięci, ile potrzebuje (stąd przyjęło się mówić w tym wypadku o **zmiennych strefach**). Pojawiają się jednak dwa nowe problemy. Po pierwsze, trzeba znaleźć w pamięci odpowiedni obszar, a po drugie, pamięć po pewnym czasie jest bardzo „poszatkowana" — są w niej przemieszane obszary zajęte z obszarami wolnymi, przy czym te ostatnie są często nieduże i przez to pozostają niewykorzystane. Jest kilka metod wyszukiwania wolnego obszaru. Najczęściej stosuje się algorytmy o nazwach: **„pierwszy zdatny"** (ang. first-fit) i **„najlepszy zdatny"** (ang. best-fit). W pierwszym wypadku przydziela się procesowi pierwszy znaleziony wolny obszar, w którym proces się zmieści. W drugim wypadku przydziela się najmniejszy z takich obszarów (w celu zminimalizowania wielkości odrzuconego kawałka wolnego obszaru). Jeżeli natomiast chodzi o „poszatkowanie" pamięci, to stosuje się co pewien czas tzw. **scalanie** (ang. compaction), czyli przemieszcza się tak przestrzenie adresowe procesów w pamięci operacyjnej, aby wszystkie wolne obszary połączyły się w jeden. Jest to operacja bardzo kosztowna.

Trzecia z wymienionych strategii, **wymiana** (ang. swapping), polega na czasowym usuwaniu całej przestrzeni adresowej pewnego procesu z pamięci operacyjnej i sprowadzaniu w to miejsce innego procesu. Strategia ta bywa stosowana zarówno samodzielnie, jak i w połączeniu z podziałem na strefy. Jest ona szczególnie przydatna w wielodostępnych systemach operacyjnych, opartych na podziale czasu (ang. time-sharing).

Gdy zauważono w latach sześćdziesiątych, że nie ma potrzeby przechowywania w pamięci operacyjnej przez cały czas całej przestrzeni adresowej wykonywanego procesu, zaczęły się rozwijać nowe metody alokacji pamięci. Doprowadziło to (po rozwiązaniu także pewnych problemów sprzętowych) do wypracowania techniki opartej na pamięci wirtualnej. Idea polega na zautomatyzowaniu przesyłania fragmentów przestrzeni adresowej procesów między pamięcią pomocniczą a pamięcią operacyjną i ukryciu przed użytkownikiem faktu istnienia dwóch poziomów pamięci. Najczęściej stosowaną metodą realizacji tej techniki jest **stronicowanie na żądanie** (ang. demand paging). Zarówno pamięć operacyjna, jak i pomocnicza są podzielone na jednakowej wielkości obszary zwane **ramkami stron**. Przestrzeń adresowa procesu użytkownika jest umieszczana w pamięci pomocniczej i dzielona **na strony** o rozmiarach odpowiadających ramkom. Następnie do pamięci operacyjnej sprowadzane są tylko te strony, które są rzeczywiście potrzebne w danej chwili. Każda strona może być przy tym umieszczona w dowolnej ramce. Problem pojawia się wtedy, gdy brakuje wolnych ramek — moduł zarządzający pamięcią musi wówczas usunąć jedną ze stron z pamięci operacyjnej do pamięci pomocniczej. Wybór takiej „ofiary" do usunięcia odbywa się według przyjętej w systemie metody. Najczęściej stosuje się algorytm zwany **LRU** (ang. least-recently-used) — usuwa się stronę najdłużej nie używaną.

Realizacja pamięci wirtualnej nastręcza wiele problemów, zarówno natury technicznej, jak i systemowej. Zły dobór algorytmu usuwania stron, nieodpowiednia wielkość strony, niewłaściwe urządzenie dla pamięci pomocniczej, brak wspomagania sprzętowego do efektywnego obliczania adresów, wszystko to może znacznie zmniejszyć korzyści z wirtualizacji pamięci. W przypadkach patologicznych może nawet dojść do zjawiska zwanego **migotaniem stron** (ang. thrashing), kiedy to system nie robi zasadniczo nic innego, tylko zajmuje się przesyłaniem stron pomiędzy obydwoma poziomami pamięci. Jednak, pamięć wirtualna jest powszechnie stosowana, co świadczy o rozwiązaniu podstawowych problemów.

Moduł zarządzający pamięcią, w zależności od tego jaką realizuje strategię, musi współpracować z różnymi innymi modułami systemu operacyjnego. Jak wspomniano, jest ścisły związek między zarządzaniem pamięcią a zarządzaniem procesorem. Ponadto, moduł ten musi współpracować z modułem zarządzającym pamięcią zewnętrzną. Jest on więc często umieszczany w jądrze systemu operacyjnego, choć nie jest to niezbędne.

Zarządzanie zasobami w Unixie

System Unix ma wiele wersji, edycji i mutacji. Oczywiste jest więc, że różnice istnieją także w modułach zarządzających zasobami, zwłaszcza przy istotnie różnych konfiguracjach sprzętowych. Niezmienny pozostaje jednak **system plików** (przynajmniej z punktu widzenia użytkownika), który przyczynił się w dużym stopniu do popularności Unixa. Po podaniu elementarnych uwag o zarządzaniu procesorem i pamięcią operacyjną w Unixie, warto więc, choćby skrótowo, przedstawić podstawowe cechy jego systemu plików.

Zarządzanie procesorem w Unixie jest ukierunkowane na faworyzowanie procesów interakcyjnych (konwersacyjnych). Każdy proces ma przyporządkowany przez system priorytet, który jest uaktualniany dynamicznie (co **T1** sekund). Ponadto jest ustalony kwant czasu **T2** (zwykle mniejszy o rząd wielkości od **T1**), na który przydziela się procesowi procesor. Im więcej czasu procesora wykorzystał proces, tym ma mniejszy priorytet. Dla procesów obliczeniowych (ang. cpu-bound) przyjęta metoda redukuje się do karuzelowej obsługi procesów **(RR)** z kwantem **T2**. Dla przykładu w wersji **Unix 4.2BSD** przyjęto: **T1**=1 s, **T2**= =0,1 s.

Zarządzanie pamięcią ulegało istotnym zmianom w miarę powstawania nowych wersji systemu. Początkowo stosowana była metoda wymiany w połączeniu ze strefami, a następnie stronicowanie na żądanie. Nie były to „czyste" strategie, ale uwzględniały wiele specyficznych cech i rozwiązań przyjętych w Unixie, a w szczególności strukturę przestrzeni adresowej procesu (której nie omówiono tu z powodu braku miejsca).

Przy prezentacji głównych cech systemu plików należy przede wszystkim wspomnieć, że autorzy Unixa podjęli pewne strategiczne decyzje, które zdeterminowały charakter systemu plików:

— **plik** jest ciągiem bajtów, bez żadnej struktury wewnętrznej,
— pliki są grupowane w **katalogi** (ang. directory), które same są także traktowane jako pliki,
— oprócz plików tekstowych i katalogów są także pliki specjalne, reprezentujące urządzenia zewnętrzne,
— nie ma wstępnej alokacji miejsca na dysku dla pliku, więc może on być dowolnej długości,
— biblioteka plików może być instalowana na kilku pakietach dyskowych,
— system ochrony plików jest prosty i elastyczny, pozwalający wyróżniać poziomy w prawach dostępu.

System plików tworzy drzewiastą strukturę hierarchiczną. Jest mianowicie wyróżniony katalog nadrzędny, zwany **korzeniem** (lub **katalogiem pierwotnym**, ang. root), który zawiera informacje o swoich bezpośrednich podkatalogach. Z kolei, każdy z tych katalogów jest początkiem poddrzewa, składającego się znowu z katalogów i (lub) innych plików. W celu obsługi tej struktury użytkownik ma do dyspozycji bogaty zestaw prostych narzędzi. Można łatwo tworzyć nowe pliki (i dołączać je do dowolnego katalogu), usuwać, kopiować lub przesuwać. Podobnie prosto można operować katalogami, a więc manipulować poddrzewami.

Plik jest identyfikowany przez **ścieżkę** prowadzącą do niego albo od katalogu pierwotnego (mówi się wtedy o **ścieżce bezwzględnej**), albo od bieżącego, którym operuje w danej chwili użytkownik (mamy wówczas **ścieżkę względną**). Syntaktycznie jest to ciąg nazw plików-katalogów rozdzielanych kreską ukośną „/". Na przykład:

JM/Referaty/R12/roz4

identyfikuje plik **roz4** będący elementem katalogu **R12**, który jest elementem katalogu **Referaty**, a ten z kolei jest podkatalogiem katalogu **JM**. Ścieżka ta jest względna, bo zaczyna się od nazwy **JM**. Natomiast zapis:

/usr/JM/poczta

oznacza ścieżkę bezwzględną, identyfikującą plik **poczta** w katalogu **JM**, który jest podkatalogiem katalogu **usr**. Początkowa kreska ukośna nie jest rozdzielaczem, ale nazwą katalogu pierwotnego.

W typowej instalacji Unixa istnieje zwykle dosyć rozbudowana biblioteka plików, o pewnej liczbie standardowych katalogów (np. **usr, lib, bin, dev**) i o dużej liczbie katalogów oraz wychodzących z nich poddrzew, tworzonych i dy-

namicznie zmienianych przez użytkowników. W szczególności jeden plik może być znany pod różnymi nazwami, w jednym lub kilku katalogach, dzięki czemu można uniknąć kosztownego kopiowania oraz ułatwić współpracę między użytkownikami.

Powyższe zwięzłe omówienie zrobiono z perspektywy użytkownika. Operacje dostępne na tym poziomie są przekształcane na wywołania jądra systemu, które dysponuje odpowiednim zestawem bardziej prymitywnych narzędzi. Należą do nich takie operacje, jak **open, close, read, write, seek, create, link, unlink.**

* * *

Problematyka systemów operacyjnych jest bardzo bogata. Przedmiot ten jest prezentowany na uczelniach w ramach całorocznego wykładu z ćwiczeniami. Niektóre podręczniki z systemów operacyjnych mają po kilkaset stron. W niniejszym artykule można więc było zaledwie naszkicować wybrane aspekty tej problematyki. Nie poruszano w ogóle wielu istotnych kwestii.

Podobnie, system Unix ma wiele wersji i każdej z nich można poświęcić wiele godzin wykładowych oraz wiele stron tekstu. Należało więc z konieczności ograniczyć się do pobieżnego przedstawienia tylko niektórych faktów o tym systemie. Trzeba przy tym pamiętać, że na Unix nadal nie ma licencji eksportowej do Polski, stąd i rozważania o tym systemie mają akademicki charakter.

Bardzo trudno jest również wskazać literaturę. Zarówno na temat systemów operacyjnych, jak i na temat Unixa ukazało się na świecie wiele podręczników, prac, raportów, dokumentacji. Kilka wartościowych pozycji (ale niestety już przestarzałych) z zakresu systemów operacyjnych zostało przetłumaczonych na język polski. Prawdę mówiąc jednak, autor artykułu od wielu lat poszukuje „idealnego" podręcznika i „idealnej" metodyki nauczania tego przedmiotu — jak dotąd, bezskutecznie. Dlatego też w niniejszym artykule zrezygnowano z bibliografii. Przy jego opracowywaniu nie korzystano z żadnego konkretnego zestawu materiałów, poza własnym tekstem autora pn. „Systemy operacyjne dla mikrokomputerów", z poprzedniej Jesiennej Szkoły PTI.

JAN BIELECKI
Instytut Informatyki
Politechnika Warszawska

Turbo Pascal

Jednym z najpopularniejszych języków programowania, choć nie tak modnym jak C, jest Turbo Pascal. Mimo iż jest to w istocie dialekt języka wzorcowego, duże rozpowszechnienie uczyniło go jednym z najważniejszych współczesnych języków programowania, szczególnie dobrze nadającym się do programowania zadań graficznych.

W niniejszym artykule zostaną omówione różnice między Turbo Pascalem i językiem wzorcowym, a następnie zostanie przedstawiony przykład programu graficznego umożliwiającego przeniesienie wykresu z monitora na drukarkę wierszową.

OGRANICZENIA A ROZSZERZENIA

Turbo Pascal wykazuje następujące ograniczenia względem języka wzorcowego:

- argumentem procedury New nie może być nazwa zmiennej wskazującej na rekord z wariantami,
- nie implementowano procedur Get i Put, zamiast nich należy posługiwać się procedurami Read i Write,
- wykluczono możliwość posługiwania się instrukcjami goto w celu zakończenia wykonywania bloku,
- nie implementowano procedury Page,
- nie implementowano parametrów reprezentujących podprogramy.

Jak wynika z przytoczonego zestawienia, ograniczenia języka Turbo Pascal są nieliczne i z wyjątkiem parametrów proceduralnych, niekłopotliwe.

Język zawiera natomiast wiele rozszerzeń, znacznie ułatwiających programowanie i skracających programy wynikowe. Omówienie ich wszystkich wykracza niestety poza zakres niniejszego artykułu. Zostały one szczegółowe przedstawione w książkach [1—4]. Poniżej zostaną omówione tylko te rozszerzenia, z których skorzystano w artykule. Dodatkowymi słowami zastrzeżonymi są w Turbo Pascalu: **absolute, external, forwad, inline, overlay, shl, shr, string, xor.** Nie jest natomiast zastrzeżone słowo packed.

IDENTYFIKATORY

Wszystkie znaki identyfikatora są istotne. Ich liczba nie może przekroczyć 127. Identyfikator musi zaczynać się od litery albo znaku podkreślenia. Jego następnymi znakami mogą być podkreślenia, litery i cyfry. Pojęcie „litera" dotyczy dowolnej małej albo dużej litery alfabetu angielskiego.

Wiele identyfikatorów ma znaczenie zdefiniowane pierwotnie. Do tej grupy należy m.in. identyfikator Pi reprezentujący daną o wartości π.

Przykład

```
program Area;
var
  Radius,Area : real;
begin
  Readln(Radius);
  Write ('Area = ',Pi * Radius * Radius)
end.
```

Wykonanie programu powoduje wyprowadzenie pola okręgu o zadanym promieniu.

LITERAŁY CAŁKOWITE, ZNAKOWE I NAPISOWE [1])

Literały całkowite mogą być przedstawiane za pomocą liczb szesnastkowych. Liczba szesnastkowa zaczyna się od znaku $ (dolar), po którym następują cyfry szesnastkowe.

Przykład

```
program jb;
begin
  Write($ffff)
end.
```

Wykonanie programu powoduje wyprowadzenie liczby —1.

Literałem znakowym jest literał napisowy składający się z jednego znaku. Literał napisowy składa się w ogólnym przypadku z sekwencji następujących bezpośrednio po sobie i występujących w dowolnym porządku: podstawowych literałów napisowych, znaków zakodowanych i znaków sterujących.

Podstawowy literał napisowy składa się z najdłuższego ciągu znaków zawartego między parą apostrofów. Znak zakodowany składa się ze znaku # (ang. hash), po którym bezpośrednio występuje liczba dziesiętna albo szesnastkowa określająca kod znaku. Znak sterujący składa się ze znaku ^ (ang. caret), po którym bezpośrednio występuje znak widoczny. Reprezentacja tak przedstawionego znaku sterującego jest identyczna z reprezentacją znaku uzyskanego przez jednoczesne naciśnięcie klawisza CTRL i obranego znaku widocznego.

Przykład

```
program JanBielecki;
begin
  Writeln('J'#97'n'#$42'ielecki')
end.
```

Wykonanie programu powoduje wyprowadzenie napisu JanBielecki.

KOMENTARZE I DYREKTYWY

Komentarzem jest nie tylko napis rozpoczynający się od nawiasu klamrowego otwierającego i zakończony nawiasem klamrowym zamykającym, ale również każdy napis, w którym nawiasy takie zastąpiono symbolami (* i *). Komentarze ograniczone znakami { i } mogą być zagnieżdżone w komentarzach ograniczonych symbolami (* i *), a komentarze ograniczone symbolami (* i *) mogą być zagnieżdżone w komentarzach ograniczonych znakami { i }.

Jeśli bezpośrednio po znaku albo symbolu rozpoczynającym komentarz (nie zagnieżdżony w innym komentarzu) występuje znak $ (dolar), to komentarz taki zostaje uznany za dyrektywę kompilatora.

Przykład

```
program Circle;
{ Just a circle }
{ $i Graph.p }
begin
  GraphColorMode;
  Circle(160,100,80,2); (* Red circle *)
  repeat until KeyPressed;
  TextMode
end.
```

W trzecim wierszu programu występuje dyrektywa kompilatora. Jej zinterpretowanie powoduje włączenie w miejscu jej wystąpienia zawartości zbioru o nazwie Graph.p.

NAGŁÓWEK PROGRAMU

Nagłówek programu może być pominięty. Nie zmieni to sensu programu.

Przykład

```
program Negate(Input,Output);
var
  Num : integer;
begin
  Read(Input,Num);
  Write(Output, -Num)
end.
```

Nagłówek programu mógłby zostać przedstawiony na przykład w postaci

```
program Negate;
```

albo zostać pominięty.

STANDARDOWE TYPY SKALARNE

Dodatkowym standardowym typem skalarnym jest typ byte. Jest on związany ze zbiorem danych całkowitych o wartościach z przedziału 0..255. W każdym miejscu programu, w którym występuje odwołanie do danej typu integer może wystąpić odwołanie do danej typu byte. Wyjątek od tej zasady dotyczy jedynie skojarzeń parametrów i argumentów podprogramów. W tym przypadku wymaga się pełnej zgodności typów danych.

Przykład

```
program Mix;
var
  first : integer;
  second : byte;
begin
  first := 8;
  second := first + 5;
  Write(second)
end.
```

Wykonanie programu powoduje wyprowadzenie liczby 13.

PREDEFINIOWANY TYP STRING

Typ string jest predefiniowanym typem napisowym. Opis tego typu składa się ze słowa zastrzeżonego **string**, po którym bezpośrednio następuje, zawarta w nawiasach kwadratowych, maksymalna liczba znaków napisu danego typu. Liczba ta może być wyrażona za pomocą literału typu integer albo za pomocą równoważnej takiemu literałowi nazwy literału. Zmiennym typu **string[n]** można przypisywać dane napisowe reprezentujące ciągi do **n** znaków.

Przykład

```
program Name;
var
  FirstName, LastName : string[8];
begin
  FirstName := 'Jan';
  LastName := 'Bielecki';
  Write(FirstName,LastName)
end.
```

Wykonanie programu powoduje wyprowadzenie napisu:

```
Jan Bielecki
```

Po wykonaniu pierwszego przypisania, zmienna FirstName reprezentuje ciąg 4-znakowy.

Zmienne typu string mogą być indeksowane w taki sam sposób jak zmienne typu **array [...] of char**. Jeśli zmiennej

---

[1]) Wychodząc z założenia, że napisem jest wszystko to, co może być napisane, a więc także dowolny fragment programu, napisy takie jak np. „jb" nazwano literałami napisowymi. W tym ujęciu literał napisowy reprezentuje pewną stałą, składającą się z ciągu znaków.

typu **string[n]** zostanie przypisana dana napisowa reprezentująca ciąg więcej niż **n** znaków, to zostanie potraktowana tak, jakby reprezentowała ciąg **n** pierwszych znaków. Nic nie stoi na przeszkodzie, aby operacje na danych napisowych, takie jak np. konkatenacja, dotyczyły zarówno tablic znakowych, zmiennych typu **char** jak i zmiennych typu **string**.

Przykład

```
program JanB;
var
  FirstName : string[3];
  LastName  : array[..8] of char;
begin
  FirstName := 'Janek';
  LastName := 'Bielecki';
  Write(FirstName + ' ' + LastName + ' = ',
        FirstName[1] + LastName[1]
end.
```

Wykonanie programu powoduje wyprowadzenie napisu:

```
Jan Bielecki = JB
```

PREDEFINIOWANE TABLICE

Predefiniowano dwie tablice o elementach typu byte i dwie tablice o elementach typu integer. Umożliwiają one dostęp do pomięci operacyjnej oraz do portów wejścia-wyjścia.

Tablica Mem zapewnia dostęp do bajtów pamięci operacyjnej. Adres bajtu składa się z adresu segmentu i przemieszczenia w segmencie. Indeksem w odwołaniu do tablicy Mem jest para wyrażeń typu integer. Wyrażenia są oddzielone znakiem „:" (dwukropek) i reprezentują odpowiednio wspomniany adres i przemieszczenie. W analogiczny sposób tablica MemW zapewnia dostęp do słów pamięci operacyjnej.

Tablica Port zapewnia dostęp do bajtowych portów wejścia-wyjścia. Indeksem w odwołaniu do tablicy Port jest wyrażenie typu integer reprezentujące adres portu. W analogiczny sposób tablica PortW zapewnia dostęp do portów słowowych.

Przykład

```
program jb;
const
  Screen = $B800;
  Blink = $80;
begin
  TextMode(BW40);
  Write('jb');
  Mem[Screen : 1] := Mem[Screen : 1] or Blink;
  Mem[Screen : 3] := Mem[Screen : 3] or Blink;
  repeat until KeyPressed;
  TextMode(BW80)
end.
```

Wykonanie programu powoduje wyświetlenie migoczącego napisu **jb**. Posłużenie się tablicą Mem umożliwia bezpośrednie sięgnięcie do atrybutów znaków umieszczonych w pamięci ekranu.

OPERATORY

Zbiór dwuargumentowych operatorów typu mnożenia rozszerzono o operatory **shl** i **shr**. Operacje **shl** i **shr** mogą dotyczyć jedynie danych typu integer. Wynikiem każdej z tych operacji jest także dana typu integer. Ma ona taką reprezentację, jaka powstaje z reprezentacji pierwszego argumentu po przesunięciu go w lewo **(shl)** albo w prawo **(shr)** o taką liczbę pozycji, jaką określa wartość drugiego argumentu. Przyjmuje się, że przesuwanie ma charakter logiczny, a nie arytmetyczny.

Zbiór dwuargumentowych operatorów typu dodawania rozszerzono o operator różnicy symetrycznej **xor**. Jeśli argumentami są dane typu boolean, to jej wynikiem jest dana typu boolean wyznaczona w następujący sposób: **a xor b** ma wartość true wtedy i tylko wtedy, gdy wartości obu argumentów są różne, w pozostałych wypadkach ma wartość false.

Istotnym rozszerzeniem języka jest dopuszczenie wykonywania operacji logicznych na danych typu integer. W każdym takim wypadku, dotyczącym operacji **not**, **and**, **or** i **xor**, wynikiem jest dana typu integer. Operacje **and**, **or** i **xor** są wykonywane na parach odpowiadających sobie bitów argumentów. Wykonanie operacji **not** powoduje zanegowanie bitów.

Przykład

```
program Invert;
{ $i...itd.Graph. }
const
  Scr = $B800;
var
  Hor,Ver,Adr : integer;
begin
  HiRes;
  Circle(320,100,80,1);
  for Ver := 0 to 199 do begin
    for Hor := 0 to 79 do begin
      Adr := 80 * (Ver shr 1) + Hor;
      if Ver and 1 <> 0 then
        Adr := Adr + 8192;
      Mem[Scr : Adr] := not Mem[Scr : Adr]
    end
  end;
  repeat until KeyPressed;
  TextMode
end.
```

Wykonanie programu powoduje wykreślenie elipsy, a następnie wykonanie inwersji ekranu w kierunku od góry do dołu. Inwersja jest wykonywana bezpośrednio w buforze ekranu. Linie o numerach parzystych są w nim zapamiętane począwszy od początku tego bufora, a linie o adresach nieparzystych — o 8192 bajty dalej.

KONWERSJE TYPÓW

Rozwinięto koncepcję funkcji ord i chr wykonujących konwersję typu. Wprowadzono rodzinę operatorów jednoparametrowych, o nazwach identycznych z nazwami typów porządkowych, umożliwiających wykonywanie konwersji między dowolnymi typami porządkowymi.

Operator konwersji ma postać opr(arg), gdzie opr jest identyfikatorem typu porządkowego, a zaś arg jest dowolnym wyrażeniem porządkowym. Rezultatem takiej operacji jest dana typu opr tak dobrana, że prawdziwa jest relacja:

```
ord (a) = ord (opr (a))
```

Przykład

```
program Convert;
type
  Days = (Mon,Tue,Wed,Thu,Fri,Sat,Sun);
begin
  if Days(integer(^M) — byte(^J) = Thu then
    Write(boolean(Tue))
end.
```

Wykonanie programu powoduje wyprowadzenie napisu TRUE.

JAWNE PRZYDZIELANIE MIEJSCA ZMIENNYM PROGRAMU

Użycie w deklaracji zmiennej słowa zastrzeżonego **absolute** umożliwia zarezerwowanie miejsca dla całej tej zmiennej w miejscu przydzielonym innej zmiennej albo w dowolnie obranym miejscu pamięci operacyjnej.

Deklaracja zawierająca słowo zastrzeżone absolute składa się z nazwy zmiennej deklarowanej, po której następuje znak „:" (dwukropek), określenie typu zmiennej, słowo zastrzeżone **absolute**, określenie miejsca, które ma zostać przydzielone zmiennej i znak „;" (średnik). Określenie miejsca może mieć postać nazwy uprzednio zadeklarowanej albo postać adresu pamięci operacyjnej. Adres pamięci operacyjnej składa się z numeru segmentu i przemieszczenia w segmencie, oddzielonych znakiem „:" (dwukropek). Numer i przemieszczenie mogą być literałami typu integer, nazwami literałów takiego typu, albo nazwami bezargumentowych funkcji Cseg, Dseg i Sseg, udostępniających odpowiednio adresy segmentu kodu, segmentu danych i segmentu stosu.

Przykład

```
program jb;
var
  Jan : byte absolute $B800 : 0;
  Bielecki : byte absolute $B800 : 2;
begin
  TextMode(BW40);
  Jan := byte ('j');
  Bielecki := byte('b');
  repeat until KeyPressed;
  TextMode(BW80)
end.
```

Wykonanie programu powoduje wyświetlenie napisu **jb.** Operacje są wykonywane bezpośrednio w buforze ekranu.

PRZYPISYWANIE DANYCH POCZĄTKOWYCH

Jedną z ważnych właściwości języka jest możliwość przypisywania zmiennych danych początkowych. Jeśli podczas wykonywania programu zmiennym, którym przypisano dane początkowe, nie zostaną przypisane inne dane, to można uznać, że takie zmienne zachowują się jak stałe. Jeśli natomiast zmiennej, której przypisano daną początkową, zostanie przypisana inna dana, przypisanie to zachowuje swą ważność podczas ponownego aktywowania programu znajdującego się w pamięci operacyjnej. Ponieważ taki program z reguły traci właściwości powtarzalności, zaleca się, aby przypisywanie danych początkowych było ograniczone do zmiennych zachowujących się jak stałe. Z tej to właśnie przyczyny, deklaracje zmiennych, którym należy przypisać dane początkowe, występują w programie po słowie zastrzeżonym **const,** a nie po słowie zastrzeżonym **var.**

W ogólnym przypadku deklaracja zmiennej, której przypisano daną początkową składa się z następujących kolejno po sobie: nazwy zmiennej, znaku „:" (dwukropek), określenia typu zmiennej, znaku „=" (równość) i literału albo nazwy literału reprezentującego daną początkową.

Przykład

```
program Print_13;
const
  _13 : byte = 13;
begin
  Write_13
end.
```

Wykonanie programu powoduje wyprowadzenie liczby 13.

INSTRUKCJA INLINE

Instrukcja **inline** składa się ze słowa zastrzeżonego **inline,** po którym bezpośrednio następuje ujęty w nawiasy okrągłe ciąg elementów kodu. Elementy kodu są oddzielone kreskami ukośnymi, a każdy z nich składa się elementów danych oddzielonych od siebie znakami „+" (plus) albo „—" (minus).

Elementem danych jest literał typu integer, nazwa takiego literału, identyfikator zmiennej, identyfikator podprogramu oraz wyrażone za pomocą znaku „*" (gwiazdka) oznaczenie licznika instrukcji, np.:

```
inline(20/$40Fun—2/*+3)
```

Każdy element danych generuje jeden bajt albo jedno słowo kodu. Przyjmuje się, że identyfikator zmiennej albo podprogramu reprezentuje adres tej zmiennej albo podprogramu. Analogicznie przyjmuje się, że oznaczenie licznika instrukcji reprezentuje adres tego najbliższego bajtu pamięci operacyjnej, w którym zostanie umieszczony wygenerowany kod.

Jeśli element kodu składa się wyłącznie z literałów, nazw literałów i separatorów, a jego wartość mieści się w zakresie 0..255, to zostanie wygenerowany jeden bajt kodu. Jeśli wartość wykracza poza ten zakres, jak również wtedy, gdy element kodu zawiera nazwy zmiennych lub podprogramów, albo odwołania do licznika instrukcji, to jest generowane jedno słowo. W słowie tym bajt mniej znaczący jest umieszczany przed bajtem bardziej znaczącym.

Przytoczone domniemania liczby generowanych bajtów kodu mogą zostać zmienione, jeśli przed elementem kodu zostanie umieszczony znak „<" (mniejszość) lub „>" (większość). W pierwszym wypadku zostanie wygenerowany tylko mniej znaczący z tych bajtów, w których jest reprezentowana wartość elementu kodu, a w drugim dwa bajty i to nawet wtedy, gdy drugi z nich (bardziej znaczący) reprezentuje daną o wartości 0.

Przykład

```
program Invert;
{ $Si Graph.p }
procedure InvertScreen;
const
  Size = 8192;
var
  ScrAdr = $B800;
begin
  inline
    $1E                 (*      PUSH DS                    *)
    /$B8/ ScrAdr        (*      MOV AX,ScrAdr              *)
    /$8E/$D8            (*      MOV DS,AX                  *)
    /$B9/ Size          (*      MOV CX,Size                *)
    /$31/$DB            (*      XOR BX,BX                  *)
    /$F7/$17            (* LAB: NOT WORD PTR [BX]          *)
    /$43                (*      INC BX                     *)
    /$43                (*      INC BX                     *)
    /$E2/$FA            (*      LOOP LAB                   *)
    /$1F                (*      POP DS                     *)
end;
begin (* Main *)
  HiRes;
  Circle(320,100,80,1):
  InvertScreen;
  repeat until KeyPressed;
  TextMode
end.
```

Wykonanie programu powoduje wykreślenie okręgu, a następnie wykonanie inwersji ekranu. Większa część ciała procedury InvertScreen została zakodowana za pomocą instrukcji inline.

REPREZENTOWANIE DANYCH

Standard języka Pascal nie określa sposobu reprezentowania danych. Poniżej podano ustalenia dotyczące reprezentowania danych przyjęte w implementacji języka Turbo Pascal dla IBM PC.

DANE TYPU PORZĄDKOWEGO

Dane typu porządkowego są reprezentowane w jednym albo w dwóch bajtach pamięci operacyjnej.

W jednym bajcie są reprezentowane dane typu **char,** dane typu wyliczeniowego związanego ze zbiorem liczącym nie więcej niż 256 elementów oraz dane typu okrojonego **min..max,** takiego, że zarówno **ord(min)** jak i **ord(max)** należy do przedziału 0..255. W szczególności w jednym bajcie są więc reprezentowane dane typu boolean i byte.

W dwóch bajtach są reprezentowane dane typu integer, dane typu okrojonego, które mogą być reprezentowane w jednym bajcie oraz dane typu wyliczeniowego o więcej niż 256 elementach.

W przypadku danych zajmujących dwa bajty, mniej znaczącym bajtem danych jest pierwszy z nich.

Przykład

```
type
  Color = (Red,Green,Blue,Yellow,Orange);
  Hue = Green..Yellow;
  Selector = (enum,card,bool);
  union = record
            case Selector of
              enum : (Rng : Hue);
              card : (Int : integer);
              bool : (Log : boolean)
            end;
  const
    HueVar : union = (Rng : Green);
  begin
    with HueVar do
    Write(Int,Log :5)
end.
```

Ponieważ ord(Green)≤255 oraz ord(Yellow)≤255, pole Rng typu Hue jest reprezentowane w jednym bajcie. Mniej zna-

czący bajt Int pokrywa się z bajtem pola Rng i bajtem pola Log. Wykonanie instrukcji Write powoduje wyprowadzenie napisu:

1 TRUE

DANE TYPU RZECZYWISTEGO

Dane typu real są reprezentowane w sześciu bajtach pamięci operacyjnej. Dane te składają się z jednobajtowego wykładnika i pięciobajtowej mantysy. Pod najniższym adresem pamięci występuje wykładnik, a za nim mantysa, w kolejności od bajtu najmniej znaczącego do najbardziej znaczącego. Wartość wykładnika jest reprezentowana z przesunięciem $80, a mantysa jest znormalizowana i pozbawiona najbardziej znaczącego bajtu. Bit ten jest wykorzystany do reprezentowania jej znaku. Dana rzeczywista typu real o wartości 0.0 jest reprezentowana przez mantysę i wykładnik składające się z samych bitów 0.

Przykład

```
type
  union = record
            case boolean of
              false: (Float : real);
              true: (Arr : array[0..5] of byte)
          end;
const
  RealVar : union = (Arr : ($83,0,0,0,0,$80) );
begin
  with RealVar do
    Write(Float)
end.
```

Pierwszy bajt zmiennej RealVar jest daną bajtową o wartości $83, a więc reprezentuje wykładnik 3. Bajt o wartości $80 jest najbardziej znaczącym bajtem mantysy. Reprezentuje on znak „—" (minus) mantysy oraz mantysę:

10000000 00000000 00000000 00000000

Najbardziej znaczący bit tego bajtu reprezentuje znak mantysy, tu „—" (minus). Ponieważ mantysa ma wartość —0.5, a wykładnik ma wartość 3, dana początkowa przypisana zmiennej RealVar ma wartość $-0{,}5 * 2^3 = -4.0$. Wykonanie instrukcji Write powoduje wyprowadzenie liczby —4.0.

DANE TYPU NAPISOWEGO

Dane typu **string** są reprezentowane w **n+1** bajtach pamięci. Pierwszy bajt danej napisowej określa liczbę znaków tej danej.

Przykład

```
const
  StrVar : string[5] = 'Janek';
procedure CutOff(var Par);
var
  Len : byte absolute Par;
begin
  Len := Len — 2
end;
begin
  Writeln(StrVar);
  CutOff(StrVar);
  Write(StrVar)
end.
```

Operacja na zmiennej Len jest w istocie operacją na najbardziej znaczącym bajcie zmiennej StrVar. Wykonanie przytoczonego programu powoduje wyprowadzenie napisu:

Janek
Jan

DANE TYPU MNOGOŚCIOWEGO

Dane typu mnogościowego, bazowane na typie porządkowym o najmniejszym elemencie min i największym elemencie max zajmują:

ord(max) div 8 — ord(min) div 8 +1

bajtów pamięci. Dane mnogościowe są reprezentowane w taki sposób, jakby bazowy typ porządkowy składał się z 256 elementów. Ponieważ wymagałoby to zarezerwowania dla danej 32 bajtów pamięci, z których wiele nie byłoby w większości wypadków używanych, z których 32 bajtów odrzuca się te bajty skrajne, których wartość nigdy nie ulega zmianie. W szczególności, dana typu set of 10..16 nie jest reprezentowana w postaci 32-bajtowej (literami x oznaczono aktywne pozycje danej :
(00000000 ... 00000000 0000000x xxxxxx00 00000000),

lecz w postaci 2-bajtowej:

0000000x xxxxxx00

w kolejności od bajtu najmniej znaczącego do najbardziej znaczącego.

Przykład

```
type
  union = record
            case boolean of
              false : (aSet : set of 10..16);
              true:   (anInt : integer)
          end;
const
  SetVar : union = (aSet : [16]);
begin
  with SetVar do
    Write (anInt)
end.
```

Pole aSet zajmuje dwa bajty i jest reprezentowane w pamięci operacyjnej w postaci 00000000 00000001. Wykonanie instrukcji Writeln powoduje wyprowadzenie liczby 256.

DANE WSKAZUJĄCE

Dane wskazujące są reprezentowane w czterech bajtach pamięci. Każda z nich jest reprezentowana tak jak dana typu integer i określa adres pamięci operacyjnej. Dana reprezentowana przez **nil** składa się z samych bitów 0.

Przykład

```
type
  union = record
            case boolean of
              false: (Ref : byte);
              true: (Int : integer)
          end;
const
  PtrVar : union = (Int : 0);
begin
  Write(PtrVar = nil)
end.
```

Zmienna PtrVar zajmuje dwa bajty. Wykonanie instrukcji Write powoduje wyprowadzenie napisu TRUE.

TABLICE

Elementy tablic są rozmieszczone w pamięci operacyjnej wierszami.

Przykład

```
type
  union = record
            case boolean of
              false: (Arr : array [boolean,
                                   boolean] of byte);
              true: (Vec : array [1..4] of byte)
          end;
const
  ArrVar : union = Arr: ((11,12) , (21,22) ));
var
  Index : 1..4;
begin
  with ArrVar do
    for Index := 1 to 4 do
      Write(Vec [Index] :3)
end.
```

Wykonanie instrukcji Write powoduje wyprowadzenie liczb: 11, 12, 21, 22.

DANE REKORDOWE

Pola rekordów są rozmieszczone w pamięci operacyjnej w kolejności ich wyszczególnienia w deklaracji. Jeśli re-

kord nie zawiera wariantów, to jego rozmiar jest równy sumie rozmiarów pól. W przeciwnym razie rozmiar rekordu jest równy sumie rozmiaru tego z wariantów, który wymaga najwięcej miejsca. Zarówno rekordy bez wariantów, jak i rekordy z wariantami są stałego rozmiaru. Informacja ta jest istotna z punktu widzenia operacji wejścia-wyjścia.

Przykład

```
var
  RecVar : record
             case Selector : integer of
               6: (Arr : array [1..3,1..2] of byte);
               8: (aSet : set of 'A'..'z')
           end;
begin
  with RecVar do
    Write(SizeOf(RecVar),
          SizeOf(Arr) :2,
          SizeOf(aSet) :2)
end.
```

Część stała rekordu RecVar ma rozmiar 2 bajty. Pierwszy wariant tego rekordu ma rozmiar 6 bajtów. Drugi wariant ma rozmiar 8 bajtów. Rekord ma rozmiar 10 bajtów. Wykonanie instrukcji Write powoduje wyprowadzenie liczb: 10, 6 i 8.

PRZYKŁAD WYKREŚLANIA PUNKTÓW I ODCINKÓW

Na wydruku 1 przedstawiono program, którego wykonanie powoduje wyprowadzenie na ekran obrazu przedstawionego na rys. 1 oraz wyprowadzenie na drukarkę graficzną obrazu przedstawionego na rys. 2. W rozpatrywanym programie włączono dwa pliki: plik PlotDraw — przedstawiony na wydruku 2 — zawierający definicję procedur do wykreślania punktów i odcinków oraz plik HardCopy — przedstawiony na wydruku 3 — zawierający definicję procedury do wykonania zrzutu ekranu. Wszystkie przytoczone programy dotyczą IBM PC z kartą kolorową. Do wykreślania odcinka wykorzystano algorytm Bresenbama, a w procedurze Plot wykorzystano fakt, że punkt o współrzędnych (x,y) jest reprezentowany w bajcie o adresie względnym

(y and 1) * 8192 + (y shr 1) * 80

równoważnym adresowi

(y and 1) shl 13 +
(y and —2) shl 3 +
(y and —2) shl 5

```
program Cardioids;

{$i PlotDraw.jb }
{$i HardCopy.jb }

var
  xCoord,yCoord : integer;
  xCor,yCor,Radius : real;
  Count,Phi : integer;
  Ph : real;

const
  Color = 1;

procedure DrawSquare(xMin,yMin,
                     xMax,yMax : integer);
begin

  Draw(xMin,yMin,xMax,yMin,Color);
  Draw(xMax,yMin,xMax,yMax,Color);
  Draw(xMax,yMax,xMin,yMax,Color);
  Draw(xMin,yMax,XMin,yMin,Color)

end;

begin

  HiRes;

  DrawSquare(0,0,639,199);

  GotoXY(15,13);
  Write('Cardioids');


  xCoord := 420;
  yCoord := 100;
  Radius := 70;
  Count  := 200;

  for Phi := 0 to Count do begin
    Ph := 2 * Pi / Count * Phi;
    xCor :=  Radius * (1 + cos(Ph)) * cos(Ph);
    yCor :=  Radius * (1 + cos(Ph)) * sin(Ph);
    Draw(xCoord + trunc(xCor),yCoord + trunc(yCor),
         xCoord,yCoord,Color);
    Draw(xCoord - trunc(xCor),yCoord - trunc(yCor),
         xCoord,yCoord,Color);
  end;

  PrintScreen('D','L');

  repeat until KeyPressed;

  TextMode

end.
```

Wydruk 1. Program Cardioids

```
{ PlotDraw.jb }

procedure Plot(x,y : integer);

const
  ScreenBase = $B800;

var
  Offset : integer;

begin

  Offset := (y and 1)  shl 13 +
            (y and -2) shl  3 +
            (y and -2) shl  5 + x shr 3;

  Mem[ScreenBase : Offset] := Mem[ScreenBase : Offset] or
                              (128 shr (x and 7))
end;

procedure Draw(xMin,yMin,xMax,yMax,Color : integer);

var
  dx,dy,s1,s2,i,e,x,y : integer;
  Flag : boolean;

function Sign(Val : integer) : integer;
begin
  if Val > 0 then
    Sign := 1
  else if Val < 0 then
    Sign := -1
  else
    Sign := 0
end;

procedure Swap(var x,y : integer);
var
  Temp : integer;
begin
  Temp := x;
  x := y;
  y := Temp
end;


begin { Draw }

  y := yMin;
  x := xMin;
  dy := Abs(yMax - yMin);
  s2 := Sign(yMax - yMin);
  dx := Abs(xMax - xMin);
  s1 := Sign(xMax - xMin);

  if dx + dy = 0 then
    Plot(x,y)
  else begin
    Flag := dy > dx;
    if Flag then
      Swap(dx,dy);
    e := dy + dy - dx;
    for i := 1 to dx do begin
      Plot(x,y);
      while e >= 0 do begin
        if Flag then
          x := x + s1
        else
          y := y + s2;
        e := e - dx - dx
      end;
      if Flag then
        y := y + s2
      else
        x := x + s1;
      e := e + dy + dy
    end
  end
end;
```

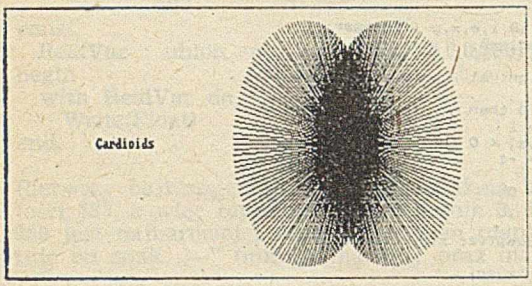
Wydruk 2. Plik PlotDraw. jb

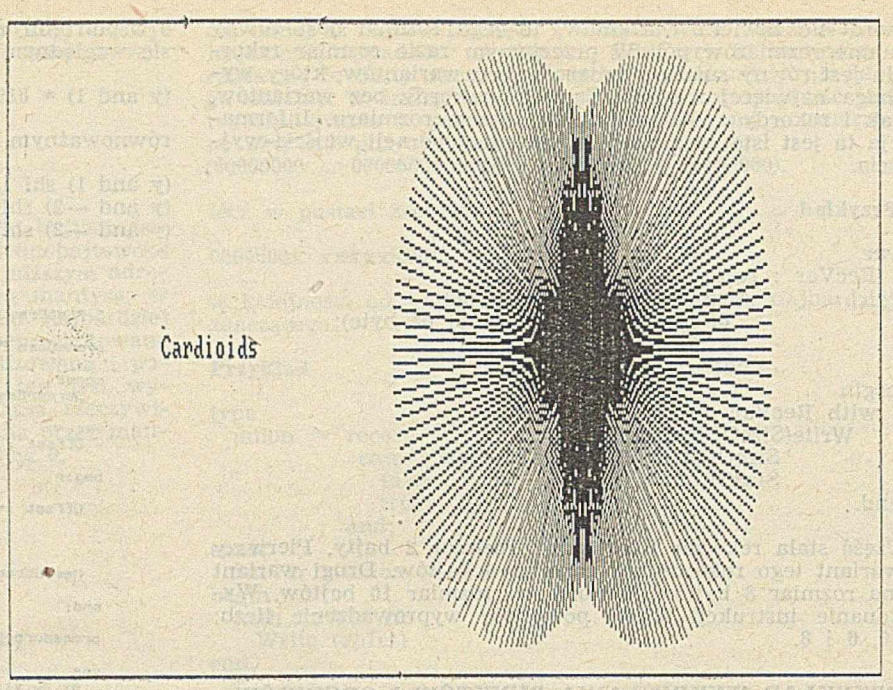
```
type
  ModeString = string[255];

procedure PrintScreen(Mode : ModeString);

  const ScreenBase = $B800;

  var
    x,yL : integer;
    Break : boolean;

  procedure OneLine;

    function PixelOn(x,y : integer) : boolean;
    begin
      PixelOn := (Mem[ScreenBase :
                      (y and  1) shl 13 +
                      (y and -2) shl  5 +
                      (y and -2) shl  3 + x shr 3] and
                      (128 shr (x and 7)) <> 0)
    end;
```

```
function OneChar(x,yL : integer) : byte;

const
  Bits : array [0..7] of byte =
         (128,64,32,16,8,4,2,1);
var
  OneByte,i : byte;

begin
  if KeyPressed then break := true;
  yL := yL shl 3;
  OneByte := 0;
  for i := 0 to 7 do
    if PixelOn(x,yL + i) then
      OneByte := OneByte or Bits[i];
  OneChar := OneByte
end;

begin  { OneLine }
  if not break then begin
    Write(Lst,Mode);
    Write(Lst,chr(Lo(640)),chr(Hi(640)));
    for x := 0 to 639 do
      Write(Lst,chr(OneChar(x,yL)));
    Writeln(Lst)
  end
end;

begin  { HardCopy }
  break := false;
  Write(Lst,^[^@'^[^3'#24);
  for yL := 0 to 24 do OneLine;
  Writeln(Lst,^[^@');
end;
```

Wydruk 3. Plik HardCopy. jb

Rys. 2. Wykres na drukarce





Rys. 1. Wykres na monitorze

LITERATURA

[1] Bielecki J.: Turbo Pascal, wersja 3.0, WNT 1987
[2] Bielecki J.: Turbo Pascal z grafiką dla IBM PC, WNT 1987
[3] Bielecki J.: Turbo Pascal dla Elwro 800 Jr, WNT 1987
[4] Turbo Pascal, wersja 3.0. Borland International, 1985.

ANNA SYFERT
ANDRZEJ RAZNOWIECKI
JANUSZ ZALEWSKI

Struktura systemu operacyjnego PC-DOS (I)

W artykule przedstawiono strukturę systemu operacyjnego PC-DOS rozumianą jako budowę na poziomie widzianym przez programistę, tj. nieco głębiej niż na poziomie zewnętrznym, widzianym przez zwykłego użytkownika (ten poziom opisano w poprzednich numerach Informatyki [2].

Do istotnych elementów tej struktury należą:

— rozmieszczenie poszczególnych części systemu w pamięci operacyjnej (mapa pamięci),
— budowa plików zawierających programy użytkowe,
— organizacja wewnętrzna plików,
— wywołania systemowe.

W kilku kolejnych odcinkach omówimy wymienione tematy, poświęcając najwięcej miejsca wywołaniom systemowym.

MAPA PAMIĘCI

System operacyjny PC-DOS znajduje się w zasadzie stale na tym samym miejscu, mianowicie w dolnym obszarze pamięci, dzięki czemu na przykład, można rozbudować pamięć bez przesuwania samego systemu. W tabeli 1 przedstawiono mapę pamięci (ang. memory map) typowego systemu PC-DOS. Przedstawienie adresu w postaci XXXX:YYYY uwzględnia podział pamięci w procesorach 8086/8088 na 64-kilobajtowe segmenty, przy czym XXXX przedstawia adres segmentu, zaś YYYY wyrównanie (ang. offset) wewnątrz segmentu. Zarówno XXXX jak i YYYY są zapisywane szesnastkowo.

Adres fizyczny może zostać utworzony według następującego wzoru:

ZZZZZ = XXXX*10H + YYYY

Należy zaznaczyć, że przyjęty sposób zapisu nie jest jednoznaczny, ponieważ adres fizyczny 23457H można zapisać jako 2345H:0007H, 2340H:0057H, 2000H:3457H itd.

Najniżej położonym obszarem w pamięci operacyjnej jest tablica wektorów przerwań [1]. Wektor przerwania służy do zaadresowania programu obsługi w odpowiedzi na żądanie przerwania. Na każde przerwanie przypadają cztery bajty odpowiedniego wektora. Dlatego adresy wektorów przerwań otrzymuje się mnożąc numery przerwań [3] przez 4. Jeżeli do komputera dołączamy nowe urządzenie (np. przyrząd pomiarowy), które nie ma typowego programu obsługi, to należy ten program napisać samodzielnie, a jego adres umieścić w odpowiednim miejscu tablicy wektorów przerwań. Podobnie, w wypadku napisania nowego programu obsługi dla standardowego urządzenia należy zmienić odpowiedni wektor przerwania.

Dwa bardzo ważne obszary komunikacyjne przeznaczone dla pamięci ROM i systemu PC-DOS omówiono oddzielnie [4]. Ich zawartość jest ładowana podczas rozpoczęcia sesji (ang. start up).

Obszar pamięci określony na mapie pamięci jako IBMBIO jest pośrednikiem między systemem operacyjnym a specyficznym sprzętem. Jest on z reguły dostosowany przez producenta do użytego sprzętu IBMBIO.COM jest ładowany do pamięci RAM przez program inicjujący (ang. bootstrap loader).

Obszar przeznaczony na plik IBMDOS.COM zawiera kod wynikowy obsługi przerwań systemowych, które zostaną omówione dokładnie w następnych częściach artykułu. Rozróżnienie między plikami IBMBIO.COM i IBMDOS.COM nie jest zbyt skomplikowane, a służy jedynie do oddzielenia części systemu operacyjnego specyficznej dla określonej konfiguracji komputera od części wspólnej dla wszystkich komputerów.

Obszar, przeznaczony na interpreter poleceń PC-DOS COMMAND.COM, jest podzielony na dwie części odpowiednio do podziału samego interpretera i zawiera: jego część rezydentną, umieszczoną obok programów obsługi urządzeń oraz część nierezydentną na końcu dostępnej przestrzeni RAM. Część rezydentna zajmuje się obsługą przerwań wspólnie z IBMBIO.DOS i wykonuje obsługę błędów (ang. error handling). Część nierezydentna przetwarza informacje wprowadzane z klawiatury, rozpoznaje je i dokonuje odpowiednich działań, np. wykonuje polecenia wewnętrzne, jak DIR, lub zewnętrzne, jak CHKDSK przez załadowanie i wykonanie odpowiednich programów.

Tabela 1. Mapa pamięci systemu operacyjnego PC-DOS

Początek obszaru	Zawartość	Uwagi
0000:0000	Tablica wektorów przerwań	Zależna od zainstalowanego sprzętu
0000:0400	Obszar komunikacyjny ROM	Obszar kontrolny
0000:0500	Obszar komunikacyjny DOS	
XXXX:0000	IBMBIO.COM — obsługa operacji wejścia-wyjścia	
XXXX:0000	IBMDOS.COM — obsługa przerwań DOS	INT 21H
XXXX:0000	Bufor PC-DOS	Specyficzny obszar użytkowy
XXXX:0000	Programy obsługi urządzeń (ang. device drivers)	
XXXX:0000	COMMAND.COM — część rezydentna	
XXXX:0000	Programy użytkowe typu COM i EXE	
XXXX:0000	Stos użytkownika — 256 bajtów	
XXXX:0000	COMMAND.COM — część nierezydentna	Koniec obszaru RAM

Jeśli część nierezydentna interpretera poleceń COMMAND.COM nie jest potrzebna, to odpowiedni obszar pamięci może zostać wykorzystany przez inne programy. W zależności od tego czy część nierezydentna znajduje się jeszcze w pamięci, czy musi zostać załadowana, takie działanie interpretera ma wpływ na wyjaśnienie niespodziewanego powstania błędu INSUFFICIENT MEMORY (niedostateczna pamięć). Jeśli, na przykład, w programie użytkowym wykonuje się polecenie DIR, to powstanie ten błąd, gdy nierezydentna część pliku COMMAND.COM nie może zostać załadowana. Ponieważ DIR jest poleceniem wewnętrznym na próżno szukalibyśmy zbyt dużego programu DIR.COM (lub DIR.EXE).

Obszary pamięci przeznaczone na bufor, programy obsługi urządzeń oraz plik COMMAND.COM są nazywane specyficznym obszarem użytkowym. Chodzi o to, że użytkownik może wpływać na zawartość tego obszaru. Plik CONFIG.SYS odczytywany podczas rozpoczęcia sesji umożliwia zmianę obszaru pamięci przeznaczonego na bufor, instalację nowych programów obsługi urządzeń (np. dla dysku) oraz użytkowanie alternatywnego interpretera poleceń.

PROGRAMY UŻYTKOWE

Do obszaru przeznaczonego na programy użytkowe można ładować pliki wykonywalne typu COM lub EXE. Pliki typu COM zawierają ostateczny kod maszynowy i są ładowane

wane pod stały adres 100H w segmencie kodu, zaś pliki kodu EXE są relokowalne. Oprócz relokowalnego kodu wynikowego w pliku typu EXE podana jest długość programu, wielkość stosu itp. Ostateczne przyporządkowanie pamięci dla plików typu EXE wykonuje się dopiero podczas ładowania, w zależności od wolnego obszaru.

Pliki typu COM są ładowane szybciej, ale pliki typu EXE są elastyczniejsze dzięki nie sztywnemu przyporządkowaniu pamięci. Przy użyciu polecenia systemowego EXE2BIN można przekształcać pliki typu EXE na pliki typu COM.

Właściwe ładowanie programu wykonuje interpreter poleceń, który po wyprowadzeniu nazwy pliku wyszukuje go i ładuje pod adresem względnym 100H. Równocześnie, w tym samym segmencie zostaje zbudowany (pod adresem względnym 0000H) tzw. prefiks segmentu programu (ang. program segment prefix, PSP), zawierający m.in. adresy skoków i obszary dla bloku kontroli pliku (ang. file control block, FCB).

Po zakończeniu procesu ładowania sterowanie zostaje przekazane do wywołanego programu. Z programu użytkowego powraca się do systemu operacyjnego za pomocą skoku do adresu względnego 0 w prefiksie segmentu programu. W razie potrzeby, interpreter poleceń zostaje załadowany na nowo i sterowanie zostaje mu przekazane ponownie.

Tabela 2. Budowa prefiksu segmentu programu PSP

Wyrównanie	Opis
00H—01H	Zawiera INT 20H; długi skok do tego miejsca powoduje zakończenie programu
02H—03H	Wskaźnik podający pierwszy wolny segment pamięci; program nie może nim dysponować, jednakże pamięć od tego adresu może zostać zarezerwowana przy użyciu funkcji 48H
04H	Zarezerwowane dla PC—DOS
05H—09H	Dalekie wywołanie (międzysegmentowe) dla funkcji (00H..., 24H); w tym celu numer funkcji musi być umieszczony w rejestrze CL; nie stosowane jako przestarzałe
0AH—0DH	Adres, do którego wykonuje się skok po zakończeniu programu (kopia wektora przerwań 22H)
0EH—11H	Adres, do którego wykonuje się skok po wciśnięciu CTRL—BREAK (kopia wektora przerwań 23H)
12H—15H	Adres, do którego wykonuje się skok przy obsłudze krytycznych błędów dysku (kopia wektora przerwań 24H)
16H—5BH	Zarezerwowane dla PC—DOS
5CH—6BH 6CH—7BH	Trzon bloków FCB (blok kontrolny pliku) z nazwami plików w poleceniu
7CH—7FH	Zarezerwowane
80H—FFH	Standardowy obszar bufora dla dyskowego wejścia-wyjścia

W tabeli 2 przedstawiono strukturę prefiksu segmentu programu, umieszczonego przed każdym programem w odpowiednim segmencie pamięci, tzn. poniżej adresu względnego 100H. Prefiks stanowi praktycznie pomost między programem a systemem PC-DOS.

PLIKI Z ROZSZERZENIEM COM

Nazwa pliku COM jest w istocie pewnym zabytkiem pochodzącym od systemu CP/M. Ponieważ w okresie rozkwitu CP/M-80 mikroprocesory mogły adresować w zasadzie tylko pamięć 64 KB, wielkość programu była ograniczona do takiego obszaru (pod warunkiem, że nie uwzględni się obszaru potrzebnego dla systemu operacyjnego), a najbardziej rozpowszechnione kompilatory i asemblery nie stosowały podziału pamięci na tzw. banki.

Ograniczenie to zostało wprawdzie usunięte po wprowadzeniu nowych procesorów, lecz usuwanie naleciałości systemu CP/M nie zostało jeszcze w pełni zakończone. Ponieważ firma Intel miała na uwadze zachowanie kompatybilności procesorów 8088 i 8086 w stosunku do 8080 oraz stworzono narzędzia przetwarzające programy dla 8080 na pragramy dla 8088/8086, można było oprzeć się na licznym i wypróbowanym oprogramowaniu pomocniczym CP/M i używać gotowych programów (np. Turbo Pascal).

Maksymalna wielkość programu z rozszerzeniem COM wynosi 64 KB; na co składa się 100H bajtów PSP, kod, dane i stos (100H bajtów na końcu 64-kilobajtowego obszaru pamięci, do którego jest ładowany program). Wszystkie cztery rejestry segmentowe (CS, DS, ES, SS) mają po załadowaniu programu tę samą zawartość, zaś licznik rozkazów (instruction pointer, IP) jest ustawiony na 100H, aby pominąć prefiks PSP leżący na początku segmentu programu. Wprawdzie możliwe jest adresowanie większego obszaru przy użyciu umiejętnych manipulacji rejestrami segmentowymi, jednakże utworzone w ten sposób oprogramowanie nie jest łatwe do uruchomienia.

Mikroprocesory 8086/8088 tworzą bezwzględny adres pamięci przez zsumowanie adresu segmentu i wyrównania (według wyrażenia: segment*10H+offset). Część adresu zwana wyrównaniem może adresować dokładnie 64 KB pamięci. Zatem przesuwanie 64-kilobajtowego obszaru pamięci po całej przestrzeni adresowej, procesora jest możliwe jedynie przez zmianę adresu segmentu w krokach co 16 bajtów. Jeśli dodatkowo, w programie z rozszerzeniem COM używa się tylko bliskiego 16-bitowego adresowania (adresowanie wewnątrzsegmentowe) to otrzymuje się program ładowalny, pod dowolny adres podzielny bez reszty przez 16 (dwójkowo xxxx xxxx xxxx xxxx 0000, gdzie x jest równe 1 lub 0).

W celu zapewnienia elastyczności systemu operacyjnego adres początkowy programu nie może zostać określony jako bezwzględny (na wielkość obszaru pamięci między IBMDOS.COM a COMMAND.COM użytkownik może mieć bezpośredni wpływ). Dzięki temu możliwe jest ładowanie programów mniejszych od 64 KB jako relokowalnych.

Przy tworzeniu programu, który powinien być wykonywany w systemie PC-DOS jako program z rozszerzeniem COM, należy pamiętać o tym, że:

- program zaczyna się zawsze od adresu 100H rozkazem wykonywalnym (licznik rozkazów ustawia się po załadowaniu programu z rozszerzeniem COM zawsze na 100H), a dane, które nie modyfikują w sposób celowy PSP, nie mogą zostać umieszczone w żadnym wypadku poniżej 100H, aby ochronić odpowiedni obszar pamięci dla PSP,
- nie można zakładać segmentu stosu, ponieważ PC-DOS przeznacza automatycznie najwyższe 100H bajtów segmentu 64 KB jako obszar stosu programu.

Mimo ograniczenia do 64 KB obszaru na program z rozszerzeniem COM, programy te mają nieprzyjemną właściwość blokowania całej wolnej pamięci. Dlatego należy zwrócić uwagę na to, że kolejne programy mogą zostać załadowane dopiero wtedy (wywołane przez funkcje 4BH), gdy program wywołujący odblokował odpowiedni obszar pamięci (wywołanie przez funkcje 4AH).

Po załadowaniu programu z rozszerzeniem COM, PC-DOS odkłada 0000H na początek stosu jako adres skoku powrotnego (wyrównanie FFFEH-FFFFH), w celu umożliwienia wykonania rozkazu RET po zakończeniu programu. Skok wykonuje się tylko wtedy, jeśli program nie zniszczył adresu skoku powrotnego, jeśli wskaźnik stosu (SP) ma wartość FFFEH bezpośrednio przed rozkazem RET i jeśli rejestr segmentowy stosu (SS) ma tę samą zawartość, co przy ładowaniu programu. Warunek ten dotyczy rejestru CS, ponieważ dzięki takiej metodzie wykona się przerwanie INT 20H w PSP (adres 00H i 01H), przy czym w trakcie kończenia wykonywania programu występują jeszcze odwołania do adresów od 0AH do 0DH w PSP. Jest oczywiste, że obszar PSP nie może być naruszony. Aby proces zakończenia programu był uwieńczony powodzeniem, należy spełnić wiele warunków i dlatego zaleca się używanie tej metody z największą ostrożnością, tym bardziej, że przy użyciu wywołania funkcji 4CH można zakończyć program omijając takie ograniczenia.

PLIKI Z ROZSZERZENIEM EXE

W przeciwieństwie do plików z rozszerzeniem COM programy z rozszerzeniem EXE są ograniczone w swoich rozmiarach jedynie przez fizyczne granice systemu (maksymalne rozszerzenie pamięci, pojemność użytych nośników itp.). Ponieważ tego rodzaju programy nie podlegają ograniczeniom segmentowym charakterystycznym dla COM, składają się często z kilku modułów, które mogą mieć własne niespójne obszary danych i stosu, dlatego używają najczęściej tzw. dalekiego adresowania (adres efektywny jest 20-bitowy). Jest więc oczywiste, że nie mogą być one ładowane do dowolnego miejsca pamięci bez podjęcia spo-

Tabela 3. Budowa nagłówka programu dla pliku z rozszerzeniem EXE

Wyrównanie	Opis
00H—01H	Zawiera znaki „MZ" jako znaki rozpoznawcze dla plików z rozszerzeniem EXE (sygnatura)
02H—03H	Liczba bajtów na ostatniej stronie użytej przez program (1 strona = 512 bajtów)
04H—05H	Długość pliku EXE w stronach (włącznie z nagłówkiem)
06H—07H	Liczba wpisów w nagłówku koniecznych do przystosowania programu do ostatecznego obszaru pamięci (ang. number of relocation entries)
08H—09H	Długość nagłówka programu jako wielokrotność 16 bajtów
0AH—0BH	Minimalna liczba bloków 16-bajtowych, które po załadowaniu wykraczają poza koniec programu (ang. minimum allocation of memory, MINALLOC)
0CH—0DH	Maksymalna liczba bloków 16-bajtowych, które po załadowaniu wykraczają poza koniec programu (ang. maximum allocation of memory, MAXALLOC). Na wartość MAXALLOC można wpływać podczas łączenia za pomocą opcji CPARMAXALLOC: n. Jeśli wartość MAXALLOC nie jest explicite ustawiona podczas łączenia, to przyporządkowuje się jej FFFFH (program zajmuje cały wolny obszar pamięci), a jeśli MINALLOC i MAXALLOC są równe 0, to program zostanie załadowany pod możliwie największym adresem pamięci
0EH—0FH	Zawartość początkowa rejestru segmentu stosu (SS) po załadowaniu
10H—11H	Wartość początkowa wskaźnika stosu (SP) po załadowaniu
12H—13H	Ujemna suma kontrolna wszystkich słów 16-bitowych pliku (ang. load error checksum)
14H—15H	Zawartość początkowa licznika rozkazów (IP) po załadowaniu
16H—17H	Zawartość początkowa rejestru segmentu kodu (CS) po załadowaniu
18H—19H	Wyrównanie tablicy relokacji od początku pliku
1AH—1BH	Numer wygenerowanej przez linker nakładki; 0 charakteryzuje część rezydentną programu
1CH ...	Tablica wpisów relokacyjnych

cjalnych środków. Ponieważ relokowalność programów jest w systemie PC-DOS niezbędna, pliki z rozszerzeniem EXE zawierają przed właściwym programem dodatkowo tzw. nagłówek programu (ang. program header), mieszczący wszystkie informacje potrzebne do załadowania programu w dowolnym miejscu pamięci (w 16-bajtowych krokach) oraz do jego wykonania pod danym adresem. Budowę nagłówka przedstawiono w tabeli 3.

RELOKOWANIE PROGRAMU

Podczas relokacji program należy zmienić tak, aby poprawne były rozkazy zawierające adresy segmentów.

Po zakończeniu konsolidacji programu, linker[1]) generuje kod, który może zostać wykonany, jeśli zostanie załadowany pod adresem 0000:0000. Aby ten program mógł być wykonywany w innym miejscu, należy we wszystkich rozkazach zawierających adresy segmentowe zmodyfikować je przez dodanie wyrównania segmentu (część segmentowa adresu wygenerowanego dla danego programu PSP). Dzięki tej modyfikacji program może być wykonywany pod dowolnym adresem pamięci podzielonym bez reszty przez 10H.

Ładowanie programu z rozszerzeniem EXE jest wykonywane w czterech krokach.

1. Najpierw następuje wczytanie nagłówka do obszaru o wyrównaniu od 00H do 1BH, przy czym z wielkości programu (zmienna SIZE) oraz wartości zmiennych MINALLOC i MAXALLOC (por. tabela 3) zostaje określona wielkość pamięci potrzebna do ładowania:

a) jeśli wolna pamięć jest mniejsza od SIZE+MINALLOC, to proces ładowania zostanie przerwany z podaniem komunikatu o niedostatecznej pamięci **(insufficient memory)**,

b) jeśli wolna pamięć jest większa od SIZE+MAXALLOC, to system PC-DOS rezerwuje obszar SIZE+MAXALLOC,

c) jeśli wolna pamięć ma rozmiar między granicami z a i b, to PC-DOS rezerwuje całą pozostałą pamięć (przy łączeniu przez linker za MAXALLOC przyjmuje się zawsze FFFFH, z wyjątkiem wypadku, kiedy wartość MAXALLOC zostaje zmodyfikowana explicite przez opcję linkera CPARMAXALLOC:n).

2. Na podstawie informacji zawartych w nagłówku i z bieżącego stanu zajętości pamięci, PC-DOS określa pierwszy segment, który może być użyty w procesie ładowania, generuje w nim PSP oraz ładuje program do pamięci.

3. Następuje wczytanie tablicy relokacji i na jej podstawie wykonywane są niezbędne modyfikacje w programie. Tablica relokacji zawiera adresy informacji segmentowych, które należy zmodyfikować względem adresu pamięci 0000:0000, pod którym program mógłby być wykonany. Wpisy w tablicy relokacji składają się zawsze z czterech bajtów (dwa bajty wyrównania, dwa bajty segmentu), dzięki czemu można łatwo znaleźć miejsca w programie, które należy zmienić.

4. Segment kodu oraz segment stosu zostają ustawione przy uwzględnieniu wyrównania segmentowego oraz informacji w nagłówku. Wskaźniki SP i IP zostają załadowane bezpośrednio, odpowiednio do informacji zawartej w nagłówku, a rejestry DS i ES zostają załadowane wyrównaniem segmentowym i w ten sposób adresują PSP.

```
        name    exe_test
stack           segment  stack
                dw       20 dup (?)
stack           ends
main            segment  byte     'code'
assume          cs:main,ds:nothing,ss:stack,es:nothing
start:          call     far ptr prompt  ;wyprowadzenie "hallo"
                mov      ah,4ch
                int      21h
main            ends
in_out  segment byte     'code'
assume          cs:in_out,ds:in_out
text            db       'hallo$'
prompt          proc     far
                 mov     ax,in_out       ;ustawienie rejestru ds:
                 mov     ds,ax
                 mov     dx,offset text  ;ust.offset text do wydruku
                 mov     ah,9            ;wybranie funkcji
                 int     21h
                 ret
prompt          endp
in_out          ends
                end      start
```

Wydruk 1. Program drukujący na monitorze ciąg znaków „Hallo"

Na wydruku 1 przedstawiono krótki program w języku asemblera wypisujący na ekranie monitora napis „Hallo". Program jest tak „niezdarnie" zbudowany, że dwa rozkazy musiały zawierać adresy segmentowe. Ponieważ podczas tłumaczenia ostateczne adresy pamięci są nieznane, asembler zaznacza je symbolem "------R" (R oznacza relokowalny).

```
File size=2H 512 byte pages, 44H bytes in last page
#of base fixup=2
Header size is 200H
Free memory needed=00H (MINALLOC)
STACK address is 0000:0028
Checksum is 1675H
Starting address is 0002:0008
Fixup Table Offset 1E:
0002:000B    0003:0008
```

Wydruk 2. Odczytanie informacji z nagłówka programu z wydruku 1

Na wydruku 2 pokazano zawartość odnośnego nagłówka w łatwo czytelnej postaci, a na wydruku 3 — fragment programu w kodzie szesnastkowym. Z wydruku 2 można odczytać, że nagłówek ma wielkość 200H bajtów, tzn. że program zaczyna się od 200H. W tabeli Fixup przyjmuje się ten adres jako punkt odniesienia 0000:0000. Przez porównanie wydruków 1, 2 i 3 stwierdzamy, że oba adresy zawarte w tabeli Fixup wskazującą dokładnie na części rozkazów zaznaczone przez asembler symbolem "------R" (adresy segmentowe). Mogą one zatem zostać odpowiednio zmodyfikowane.

[1]) W zasadzie, w języku polskim, program łączący nazywa się konsolidatorem. Jednakże równie często jest przez informatyków używany termin linker (ang. link — łączyć). Sądzimy, że można dopuścić jego użycie, ponieważ nie narusza to żadnych reguł językowych (przyp. red.).

```
0: 4D 5A 44 00 02 00 02 00 20 00 00 00 FF FF 00 00
10: 28 00 75 16 08 00 02 00 1E          01 00 0B
20: 02 00 08 00 03
.
.
.
220: 00 00 00 00 00 00 00 00 9A 07 00 03 00 B4 4C CD
230: 21 48 61 6C 6C 6F 24 B8 03 00 BE DB BA 01 00 B4
240: 09 CD 21 CB
```

Wydruk 3. Fragment programu w kodzie szestnastkowym

* * *

W punkcie dotyczącym plików z rozszerzeniem COM omówiono jedną z metod kończenia programu. Łącznie wyróżnia się cztery metody:

a) długi skok (międzysegmentowy) przez wyrównanie 0 z obszaru PSP;

b) wykonanie przerwania INT 20H przez CS:0000 (tzn. wyrównanie równe 0000, adres segmentu w rejestrze CS), tj. wskaźnik pierwszego bajtu obszaru PSP;

c) wykonanie przerwania INT 21H lub rozkazu dalekiego wywołania (ang. long call) międzysegmentowego przez wyrównanie 50H obszaru PSP, przy czym AH zawiera 00H, a CS:0000 wskazuje na pierwszy bajt obszaru PSP;

d) wykonanie przerwania INT 21H lub rozkazu dalekiego wywołania przez wyrównanie 50H obszaru PSP, przy czym AH zawiera 4CH.

Ponieważ tylko ostatnia z tych metod nie jest poddana żadnym ograniczeniom, należy ją preferować. Adresy wektorów przerwań od 22H do 24H (tzn. od 0000:0088H do 0000:0093H) zostają po zakończeniu programu ustawione zgodnie z ich wartościami w PSP, dzięki czemu wektory tych przerwań można wykorzystywać wewnątrz programu bez potrzeby ich dodatkowego odtwarzania.

LITERATURA

[1] Dworakowski W.: Szczególnie istotne adresy, przerwania i porty IBM PC/XT. Informatyka, nr 4, 1987

[2] Grabowicz R., Zalewski J.: System operacyjny PC-DOS, cz. 1—5. Informatyka, nr 7—12, 1986, 1—2, 1987

[3] Syfert A., Raźnowiecki A.: Funkcje systemu BIOS dla PC/XT. Informatyka, nr 4, 1987

[4] Zalewski J.: Obszar komunikacyjny systemu BIOS. Informatyka, nr 7, 1987.

Obszar komunikacyjny systemu BIOS

W pamięci RAM komputerów IBM PC znajdują się dwa obszary o adresach początkowych 0:400 i 0:500, służące do wymiany informacji z systemem BIOS. W zasadzie są one prywatną własnością BIOS-a, lecz ich zawartość można odczytywać, a częściowo nawet zmieniać, z programów użytkowych. Poniżej przedstawiono opis zawartości najważniejszych komórek z tego obszaru.

410H-411H. Słowo to zawiera opis konfiguracji komputera, tzn. zakodowane dane określające, które urządzenia znajdują się w aktualnie użytkowanym zestawie. Sposób kodowania, w zasadzie ustalony dla komputerów PC i PC/XT, przedstawiono w tabeli 1. Do odczytania tych danych służy przerwanie INT 11H.

412H. Jest to pojedynczy bajt służący do zliczania błędów łącza klawiatury opartego na przesyłaniu promieniowania podczerwonego, w modelu PC jr. W pozostałych modelach komputera nie ma żadnego znaczenia programowego.

Tabela 1. Słowo opisu konfiguracji

Bit	Znaczenie
0	Obecność napędów dyskowych (p. bity 6 i 7)
1	Nieużywany — zawsze ma wartość 0
2—3	Pojemność pamięci RAM na płycie systemowej: 00—16 KB 01—32 KB 11—64 KB (nie dotyczy pamięci o pojemności 256 KB)
4—5	Początkowy tryb pracy ekranu: 01—obraz kolorowy, 40 kolumn, model PC jr 10 —obraz kolorowy, 80 kolumn 11 — obraz monochromatyczny, 80 kolumn
6—7	Liczba napędów dyskietek: 00 — pojedynczy napęd 01 — 2 napędy, itd.
8	Brak mikroukładu DMA (zawierają go wszystkie modele oprócz PC jr)
9—11	Liczba kanałów szeregowych RS232
12	Obecność adaptera do gier
13	Obecność drukarki szeregowej
14—15	Liczba dołączonych drukarek

413H-414H. W tym słowie przechowywana jest liczba oznaczająca pojemność pamięci (jako wielokrotność kilobajtów). W modelu PC jr jest to rzeczywista pojemność pamięci pomniejszona o wielkość obszaru przeznaczonego dla wyświetlacza, natomiast w pozostałych modelach jest to całkowita pojemność pamięci. Do odczytania tej liczby służy przerwanie INT 12H.

417H-418H. Są to dwa bajty określające, jak należy interpretować poszczególne przyciski klawiatury. Z informacji tych korzystają intensywnie podprogramy BIOS-a. Zmiany znaczenia klawiszy dokonuje się ustawiając odpowiednie bity obu bajtów (tab. 2 i 3). Nie zaleca się dokonywania zmian w drugim bajcie.

419H. Jest to bajt przeznaczony do przyszłego wykorzystania w wypadku dołączenia drugiej klawiatury.

Tabela 2. Znaczenie zawartości pierwszego bajtu opisu klawiatury

Bit	Znaczenie
0	Prawy klawisz SHIFT przyciśnięty
1	Lewy klawisz SHIFT przyciśnięty
2	Klawisz CTRL przyciśnięty
3	Klawisz ALT przyciśnięty
4	Klawiatura w stanie SCROLL-LOCK
5	Klawiatura w stanie NUM-LOCK
6	Klawiatura w stanie CAPS-LOCK
7	Klawiatura w stanie INS

Tabela 3. Znaczenie zawartości drugiego bajtu opisu klawiatury

Bit	Znaczenie
0—1	Niewykorzystane
2	Uaktywnienie sygnału dźwiękowego przy wciskaniu klawisza (dotyczy modelu PC jr)
3	Klawiatura w stanie zatrzymania CTRL-NUM-LOCK
4	Klawisz SCROLL-LOCK przyciśnięty
5	Klawisz NUM-LOCK przyciśnięty
6	Klawisz CAPS-LOCK przyciśnięty
7	Klawisz INS przyciśnięty

41AH-41BH. Słowo to wskazuje nagłówek bufora klawiatury (41EH-43DH), gdzie zapamiętywane są kody odpowiednich znaków zaraz po wprowadzeniu.

41CH-41DH. Słowo to wskazuje stopkę bufora klawiatury.

41EH-43DH. Jest to bufor klawiatury złożony z 32 bajtów, używanych jako 16 dwubajtowych słów, służących do zapamiętywania znaków wprowadzonych do klawiatury przed ich użyciem w wywołaniu przerwania INT 16H. Bufor jest cykliczny, dlatego potrzebne są dwa wskaźniki określające jego nagłówek i stopkę (przechowywane w komórkach 41AH i 41CH).

43EH. Jest to bajt określający, czy należy przeprowadzić kalibrację napędu dyskietki przed wyszukaniem ścieżki. Jedynki na pozycjach 0—3 odpowiadają potrzebie kalibracji odpowiednich napędów o numerach od 0 do 3. Na ogół, odpowiedni bit jest wyzerowany po nieudanej próbie użycia napędu, np. w wypadku żądania dostępu bez uprzedniego włożenia dyskietki, co objawia się znanym komunikatem:

Not ready reading drive ...
Abort, Retry, Ignore?

43FH. Jest to pojedynczy bajt określający stan silnika odpowiedniego napędu dyskietek. Jedynki na pozycjach 0—3 oznaczają, że silniki napędów o numerach od 0 do 3 są włączone.

440H. Jest to bajt służący do odliczania czasu potrzebnego do wyłączenia silnika napędu dyskietek. Na początku każdej operacji dotyczącej dyskietek przybiera wartość 37H (co oznacza ok. 2 s). Przy każdym impulsie zegara sieciowego jego zawartość jest dekrementowana, gdy osiągnie 0 — wyłącza się silnik.

441H. Jest to pojedynczy bajt określający błędy przy dostępie do dyskietek. Znaczenie poszczególnych bitów przedstawiono w tabeli 4. Ustawienie bitu odpowiada wystąpieniu odpowiedniego błędu.

Tabela 4. Opis błędów powstających przy dostępie do dyskietek

Bit	Znaczenie
0	Nieważny rozkaz wydany dla dyskietki
1	Nie znaleziony znacznik adresu na dyskietce
2	Nie znaleziony sektor; zniszczona lub niesformatowana dyskietka
3	Błąd DMA przy dostępie do dyskietki
4	Błąd danych CRC (ang. cyclic redundancy check)
5	Uszkodzony mikroukład sterownika dyskietek
6	Nie znaleziona ścieżka
7	Błąd przeterminowania (ang. time out)

442H-448H. Jest to siedem bajtów określających stan sterownika dyskietek.

449H. Ten bajt służy do określania trybu pracy ekranu według tabeli 5. Tryby 8—10 i 13—15 nie dotyczą standardowej karty grafiki kolorowej.

Tabela 5. Słowo opisu trybu pracy ekranu (tryby 8—10 i 13—15 nie dotyczą standardowej karty CGA)

Zawartość	Znaczenie
0	Obraz monochromatyczny, 40 kolumn
1	Obraz kolorowy (16 kolorów), 40 kolumn
2	Obraz monochromatyczny, 80 kolumn
3	Obraz kolorowy (16 kolorów), 80 kolumn
4	Grafika średniej rozdzielczości, 4 kolory
5	Grafika średniej rozdzielczości, obraz monochromatyczny (4 odcienie szarości)
6	Grafika dużej rozdzielczości, 2 kolory
7	Tryb adaptera monochromatycznego
8	Grafika małej rozdzielczości, 16 kolorów
9	Grafika średniej rozdzielczości, 16 kolorów
10	Grafika dużej rozdzielczości, 4 kolory
13	Grafika średniej rozdzielczości, 16 kolorów
14	Grafika dużej rozdzielczości, 16 kolorów
15	Grafika bardzo dużej rozdzielczości, 4 kolory

44AH-44BH. Jest to słowo określające szerokość tekstu na ekranie, tj. liczbę kolumn — 20, 40 lub 80.

44CH-44DH. To słowo określa czas potrzebny na odświeżenie ekranu. Jego wartość jest równa liczbie bajtów przypadających na stronę ekranu, co oczywiście zależy od trybu pracy.

44EH-44FH. To słowo określa wyrównanie w pamięci dla bieżącej zawartości ekranu. Stanowi początkowy adres aktualnie wyświetlanej strony.

450H-45FH. Jest to osiem słów, które określają położenie kursora dla każdej z ośmiu stron obrazu. Pierwszy bajt każdego słowa stanowi numer kolumny (zależnie od trybu pracy, w zakresie 0—19, 0—39 lub 0—79), a drugi — numer wiersza (0—23). Zmiany są skuteczne dopiero od następnego wyprowadzenia obrazu.

460H-461H. Te bajty określają rozmiar kursora. Drugi bajt odpowiada początkowej, a pierwszy — końcowej linii tworzącej kursor. W przeciwieństwie do bajtów określających położenie kursora, zmiana tych wartości nie wystarcza do automatycznej zmiany rozmiaru kursora.

462H. Jest to bajt określający numer bieżącej strony obrazu.

463H-464H. W tych dwóch bajtach jest przechowywany adres portu sterownika ekranu mikroukładu 6845, normalnie ustawiony na 3D4H.

465H. Ten bajt określa tryb pracy sterownika ekranu.

466H. Ten bajt służy do ustawiania palety kolorów.

467H-46BH. Jest to pięć bajtów służących do sterowania pamięcią kasetową.

46CH-46FH. Te cztery bajty traktowane jako całość określają liczbę taktów zegara w ciągu doby. Po upływie 24 godzin, o północy, zawartość tych komórek jest zerowana, a do komórki 470H wpisywana jest jedynka. System operacyjny oblicza na tej podstawie bieżący czas lub wpisuje do tych komórek wartości zgodne z wprowadzonym czasem (przerwanie INT 1AH).

470H. Jest to pojedynczy bajt określający upływ jednej doby. Gdy liczba cykli zegarowych przekroczy długość doby, ten bajt przybiera wartość 1, co oznacza, że należy uaktualnić datę. Bajt jest ponownie zerowany przy pierwszym odczycie zegara (przerwanie INT 1AH), co oznacza, że każdy program odczytujący czas dzienny powinien brać pod uwagę uaktualnienie daty. Warto zauważyć, że ten bajt jest ustawiany tylko raz na początku doby i nie jest później zwiększany, tak więc nie może wskazać upływu dwóch dób.

471H. Ten bajt wskazuje użycie klawisza CTRL-BREAK, co jest sygnalizowane ustawieniem bajtu 7.

472H-473H. Te dwa bajty wskazują, że wykonywana jest inicjalizacja spowodowana przyciśnięciem klawiszy CTRL-ALT-DEL. Zawartość tych komórek wynosi wtedy 1234H.

474H-488H. Ten obszar jest związany tylko z użyciem modelu PC jr.

4F0H-4FFH. Jest to 16-bajtowy obszar zwany wewnątrzaplikacyjnym obszarem komunikacyjnym (ang. intra-application communications area, ICA), służący do przechowywania danych współdzielonych przez kilka programów.

500H. Ten pojedynczy bajt opisuje stan operacji drukowania zawartości ekranu (klawisz PRT-SCR); 00H oznacza poprawne zakończenie operacji, 01H — trwanie operacji, a FFH — błędne wykonanie operacji.

504H. Jest to pojedynczy bajt używany w zestawach z jednym napędem dysków elastycznych. Jego zawartość wynosi 0, gdy napęd pracuje jako A, a 1, gdy napęd pracuje jako B.

510H-511H. Te dwa bajty są używane przez intepreter Basica do przechowywania domyślnego numeru segmentu danych.

512H-515H. Są to dwa słowa używane przez interpreter Basica jako wektor przerwania zegarowego, wskazujący własny program obsługi.

516H-519H. Te dwa słowa są używane przez interpreter Basica i służą jako wektor przerwania wskazujący program obsługi klawisza CTRL-BREAK.

51AH-51DH. Kolejne dwa słowa również służą jako wektor przerwania, wskazujący program obsługi błędów dyskietek, i są także używane przez interpreter Basica.

JANUSZ ZALEWSKI (oprac.)

FILEPRO 16 i FILEPRO 16 PLUS — oprogramowanie do zarządzania bazą danych pod nadzorem systemu MS-DOS lub Unix

FilePro 16 i filePro 16 Plus są systemami zarządzania bazą danych firmy Small Computer Company (dla komputerów IBM PC) pracującymi pod nadzorem systemów PC-DOS i MS-DOS oraz Unix i Xenix. Istnieją także wersje tego oprogramowania dla kilku innych komputerów, które nie są dokładnie kompatybilne z IBM PC.

Obydwa programy są napisane częściowo w języku C, a częściowo w języku asemblera. Są one przy tym wyjątkowo szybkie, zwłaszcza jeśli chodzi o kluczowe operacje zarządzania relacyjnymi bazami danych, tj. operacje sortowania, selekcji i przetwarzania dużych grup rekordów (patrz rezultaty eksperymentalne podane w tabeli). Zasadniczo filePro 16 i filePro 16 Plus wymagają konfiguracji z dyskiem stałym. Jednak w wypadku komputerów z dyskiem elastycznym o szczególnie dużej pojemności (np. Tandy 2000), ograniczenie to może być nieco złagodzone — mimo pewnych kłopotów związanych z koniecznością zamieniania dyskietek, może na nich działać system filePro 16 (lecz nie filePro 16 Plus).

Orientycyjne czasy wykonywania niektórych operacji w systemach filePro 16 i filePro 16 Plus. Wyniki uzyskano w warunkach warsztatowych dla bazy danych o 1000 rekordach po 100 znaków każdy. Podane czasy są zaokrąglone do pełnej sekundy.

Operacja	Czas
Sortowanie pliku według pola niezaindeksowanego	26
Dostęp do ostatniego rekordu w pliku niezaindeksowanym	22
Dostęp do ostatniego rekordu w pliku zaindeksowanym	natychmiastowy
Zaindeksowanie pliku	64

Pierwowzorem systemu filePro 16 był bardzo popularny program zarządzania bazą danych dla rodziny komputerów 8-bitowych TRS-80, firmy Radio Shack, zwany Profile. Program ten został rozwinięty i zaadoptowany do środowiska systemu MS-DOS; uzyskał przy tym nową nazwę — Profile 16 — oraz lepsze właściwości. Profile 16 (zaprojektowany dla mikrokomputerów Tandy Model 16 i 6000) oraz filePro 16 są praktycznie identyczne.

FilePro 16 a filePro 16 Plus

Pod względem koncepcyjnym filePro 16 Plus, który ma więcej narzędzi do tworzenia oprogramowania użytkowego jest szczególnie wygodny dla programistów aplikacyjnych, natomiast filePro 16 dobrze nadaje się dla użytkowników chcących założyć bazę danych i zarządzać nią. Opcjonalnie są dostępne środki umożliwiające przenoszenie programów aplikacyjnych między systemami operacyjnymi (np. przenoszenie plików z systemu jednostanowiskowego do systemu wielostanowiskowego), konwersję zewnętrznych plików danych na format akceptowany przez filePro 16 oraz konwersję plików z formatu właściwego dla filePro 16 na inne formaty, takie jak np. DIF.SYLK lub ASCII.

Program filePro 16 Plus, w odróżnieniu od filePro 16, może traktować pliki zewnętrzne tak jak pliki własne. Obydwa programy faktycznie składają się z dwóch głównych bloków. Pierwszym z nich jest wyrafinowany blok zarządzania plikami, a drugim — programowalny procesor plików o bardzo dużych możliwościach. Wielu użytkowników może zaspokoić wszystkie swoje potrzeby korzystając jedynie z pierwszego bloku.

Ograniczenia

Liczba plików, jaką może obsługiwać filePro 16, jest ograniczona jedynie wielkością wolnej pamięci na dysku stałym. Pojedynczy plik może zawierać nie więcej niż 16 milionów rekordów. Rekord może mieć do 999 pól po 4608 znaków każde. Korzystając z filePro 16 można uzyskać 36 plansz [1]) na rekord oraz 200 przetwarzalnych elementów na jedną tablicę. Program filePro 16 dopuszcza natomiast większe rekordy (maksymalnie 16384 znaki) oraz większą liczbę przetwarzalnych elementów na tablicę (1000).

Tworzenie plików

W systemie filePro 16 wykonywanie operacji jest sterowane z użyciem techniki menu. Program nie szczędzi zachęt i podpowiedzi dotyczących możliwych wariantów kontynuowania pracy. Aby założyć plik, należy ustalić dla niego nazwę i wypisać ją, a następnie do dostarczonej tablicy dla każdego rekordu wstawić listę pól i ich długości. Na życzenie, program sporządza plansze ekranowe, formaty wydruków (raportów) oraz indeksy [2]). W miarę potrzeby formaty te można rozbudowywać.

Do czasu wprowadzenia do pliku pierwszych danych można go jeszcze dowolnie modyfikować. Z chwilą jednak, gdy plik przestanie być pusty, nowe pola można tworzyć tylko za polami już istniejącymi. Zapobiega to ewentualnym nieporozumieniom związanym z aktualną numeracją pól. Jeżeli rekordy są bardzo długie, to warto utworzyć kilka rezerwowych pól w segmencie kluczowym. W tym celu wypisuje się po prostu numer pola i określa dla niego długość równą zero. Informacja zawarta w segmentach kluczowych jest przetwarzana szybciej niż informacja w segmentach danych, dlatego jest pożądane, aby pola najczęściej sortowane lub selekcjonowane znajdowały się w segmentach kluczowych.

Typy pól

Definiując pola można dla każdego z nich określić typ danych. Jeżeli na przykład dla jakiegoś pola zostanie wyspecyfikowany typ telefoniczny (ang. phone), to w polu tym będą akceptowane tylko cyfry, a każdy wprowadzony do niego ciąg cyfr będzie automatycznie zamieniany na postać telefoniczną; np ciąg 1112223333 zostanie zamieniony na (111) 222-3333. W celu ułatwienia wprowadzenia danych oraz zapewnienia integralności, oprócz typu telefonicznego przewidziano również inne typy danych — data, czas oraz kilka typów numerycznych. Użytkownik może również określić swoje własne typy.

Tworzenie plansz wejściowych

Tworzenie plansz do wprowadzania danych jest w systemie filePro 16 prawdziwą przyjemnością, gdyż odbywa się bezpośrednio, tzn. w każdym kroku dokładnie widać uzyskany efekt. Takie udogodnienia jak wyświetlanie negatywne (ang. reverse video), proste linearne znaki gra-

[1]) Przez planszę należy tu rozumieć obraz wyświetlony na ekranie w postaci formularza (w oryginale — screen) — przyp. red.

[2]) W rzeczywistości chodzi tu o skorowidze. Ponieważ termin skorowidz (ang. directory) ma w informatyce inne znaczenie, w artykule pozostawiono dosłowne tłumaczenie angielskiego wyrazu index — przyp. red.

ficzne (ang. line graphics characters) oraz automatyczne kreślenie prostokątów (ramek) sprawiają, że tworzenie plansz o wysokiej, profesjonalnej jakości jest bardzo łatwe.

Użytkownik dysponuje również kilkoma użytecznymi polami utrzymywanymi przez system; mogą one być sortowane i selekcjonowane dokładnie w ten sam sposób jak pozostałe pola. Na przykład, zapis ¤UD informuje program, że w poszczególnych rekordach ma być automatycznie utrzymywane pole systemowe, zawierające datę ostatniej aktualizacji. Załóżmy, że dla jakiegoś typu rekordu określono kilka plansz, z których każda zawiera wspólne pole — nazwisko klienta. Wprowadzenie nazwiska klienta na pierwszą planszę powoduje, że automatycznie trafi ono także na wszystkie pozostałe plansze. Istnieje możliwość zapewnienia, że dane pole będzie można zmienić tylko podczas pracy z jedną wybraną (np. pierwszą) planszą. Dla osiągnięcia takiego efektu można posłużyć się znacznikiem ochrony pola (jest nim znak „!"), który należy umieścić na wszystkich planszach z wyjątkiem pierwszej. Użycie innego wskaźnika, tzw. wskaźnika obowiązkowego wypełnienia (ang. mustfill indicator; znak „%"), gwarantuje, że każdy rekord wprowadzany do pliku ma wypełnione wszystkie pola uznane za krytycznie ważne. Ponadto użytkownik może ustalić ścieżkę dla kursora, co umożliwia operatorowi przechodzenie po polach w określonej uprzednio wyspecjalizowanej kolejności.

Formaty wyjściowe

Tworzenie wyjściowych formatów wydruku jest podobne do tworzenia plansz wejściowych. Pola umieszcza się na wyświetlanym formularzu (ang. video form) i w ten sposób sygnalizuje, w którym miejscu mają pojawić się one na wydruku. Program filePro 16 pozwala tworzyć formaty dla raportów, etykiet, formularzy całostronicowych (które mogą być tak przygotowane, aby pasowały do druków ubezpieczeniowych, czeków płatniczych, faktur itp.) oraz dokumentów o strukturze swobodnej (ang. free form documents). Można także zdefiniować tzw. formaty tylko do przetwarzania, umożliwiające realizację przetwarzania wsadowego. Określając format wyjściowy można wskazać pola, które mogą być drukowane w postaci posortowanej. Nie jest to jednak jedyny sposób uzyskania uporządkowanego wydruku. Ochrona danych jest typu hasłowego (ang. password) i jest opcjonalna.

Indeksy

Tworzenie indeksów odbywa się w specjalnym trybie pracy systemu, zwanym Index Maintenance. Indeksy mogą być typu automatycznego lub żądaniowego. Indeksy automatyczne są utrzymywane przez program, który je aktualizuje dynamicznie w miarę usuwania, dostawiania, bądź modyfikowania rekordów. Indeks automatyczny może być, na przykład, użyty w celu utrzymywania rekordów w porządku alfabetycznym nazwisk klientów. Każdy rekord jest wówczas dostępny prawie natychmiast z chwilą wypisania pierwszych kilku liter nazwiska klienta. Indeksy typu żądaniowego, pozwalające realizować złożone operacje selekcji i sortowania, mogą być użyte do przygotowania odpowiedzi na zapytania oraz w operacjach druku. Można także tworzyć hierarchię indeksów (indeksy do indeksów), co przyspiesza wyszukiwanie. Jest osiem rodzajów sortowania: rosnące lub malejące według zawartości pól tekstowych, numerycznych oraz pól typu data i czas.

Możliwości wyszukiwania są imponujące. Nawet użytkownik o bardzo dużej wyobraźni miałby poważne trudności, aby znaleźć takie zastosowanie, w którym można by je w pełni wykorzystać. Rekordy mogą być selekcjonowane jednocześnie według 72 kryteriów. Rozszerzona tablica wyszukiwania zawiera 12 linii przeznaczonych do definicji tych kryteriów. Grupom linii zapisanym w tej tablicy można nadać nazwy oraz zachować je w postaci tzw. zbiorów selekcji, w celu późniejszego użycia. Etykiety takich grup mogą być dalej łączone za pomocą operatorów logicznych AND i OR oraz za pomocą operatorów porównania EQ, NE, GE, GT, LE, RG (selekcja zakresów), CO (skrót od angielskiego wyrazu contains — zawiera) oraz xxF (porównanie pól; xx jest jednym z operatorów porównania EQ, NE itd.). Na dole tablicy wyszukiwania podaje się wyrażenie selekcji. W wyrażeniu tym mogą być użyte etykietowane linie selekcji (z bieżącej tablicy wyszukiwania), uprzednio zachowane zbiory selekcji oraz operatory AND i NOT. Zapamiętane zbiory selekcji mogą być także użyte w tzw. tablicach przetwarzania.

Bloki przetwarzania

Bloki przetwarzania umożliwiają tworzenie złożonych interakcyjnych programów, takich jak na przykład pełnowartościowy system rozliczeniowy. W blokach przetwarzania, podobnie jak w programach w Basicu, można używać wyrażeń warunkowych (IF ... THEN) oraz operatorów porównania. Istnieją trzy typy bloków przetwarzania: wejściowy, automatyczny i wyjściowy, wykonywany podczas przebiegów wsadowych. Polecenia umieszcza się w tablicy przetwarzania. Jest 19 różnych poleceń, m.in. AVG, COPY, DELETE, END, GOSUB, RETURN, GOTO, TOT oraz polecenie LOOKUP, pozwalające czasowo powiązać ze sobą różne pliki.

Właściwości specyficzne

Jedna z bardziej specyficznych właściwości dotyczy wydruku. Na dole ekranu występuje komunikat „F-Print Form". Naciskając klawisz F i wypisując nazwę któregoś z formatów wyjściowych, np. formatu raportu, faktury czy koperty, powoduje się natychmiastowe wydrukowanie rekordu we wskazanym formacie. Jest to właściwość bardzo przydatna, choć nie zawsze doceniana. W większości innych systemów zarządzania bazami danych operację podobną do opisanej można wykonać tylko w trybie wsadowym. Listy menu definiowane przez użytkownika pozwalają zautomatyzować dowolne kombinacje funkcji. Mając odpowiednie menu można przez naciśnięcie jednego klawisza uruchomić niewidoczną, lecz często skomplikowaną procedurę realizującą uprzednio zdefiniowane ciągi poleceń.

Dla wielu użytkowników ulubioną właściwością systemu filePro 16 są tzw. skojarzone pola (ong. associated fields). Po przełączeniu systemu do pracy w trybie definicji plików (ang. define file mode), użytkownik może skojarzyć ze sobą kilka różnych pól określając dla nich pewien specjalny kod. Jeżeli w dalszym ciągu zamiast numeru konkretnego pola zostanie podany ten kod, to program wykona wyspecyfikowane sortowanie lub wyszukiwanie na wszystkich skojarzonych ze sobą polach w jednym przebiegu.

W celu zilustrowania tej właściwości załóżmy, że w bazie danych są opisane leki, po jednym w jednym rekordzie. Oto jak w uproszczeniu można przedstawić przykładowe skojarzone pola utworzone w trybie definicji plików (przypisano im kod A1):

Pole

1-Nazwa leku
2-A1) Negatywny efekt uboczny
3-A1) Negatywny efekt uboczny
4-A1) Negatywny efekt uboczny
5-A1) Negatywny efekt uboczny

Dla każdego leku można teraz zapamiętać cztery najbardziej typowe efekty uboczne. Oczywiście, w pliku może być opisanych wiele setek leków.

Przypuśćmy, że chcemy wyszukać wszystkie leki, które powodują skutki uboczne w postaci zawrotów głowy oraz senności. Jeżeli zadamy selekcję według zawartości pola A1, to program przeszuka wszystkie skojarzone ze sobą pola 2, 3, 4 i 5 i dostarczy te rekordy, które w polach opisujących efekty uboczne mają wymienione zarówno zawroty głowy, jak i senność.

W systemie konwencjonalnym, najpierw należałoby wykonać poszukiwanie każdego z tych czterech pól oddzielnie dla deskryptora **zawroty głowy**, a potem to samo dla deskryptora **senność**. Potem należałoby ręcznie przeszukać wyselekcjonowane rekordy i wyeliminować te, w których nie są wymienione jednocześnie obydwa efekty uboczne.

Usprawnienia

Wcześniejsze wersje programu filePro 16 miały kilka niedociągnięć technicznych i koncepcyjnych. Procedura odczytująca skorowidz pliku (ang. file director) nie wyświetlała zbiorów selekcji, typów przetwarzania i formatów wyjściowych. Program zawieszał się czasami w trybie Index Maintenance i wymagał częściowego inicjowania (ang. warm reboot). Ponadto, po dłuższym okresie użytkowania automatyczne indeksy nie zapewniały utrzymywania rekordów w ustalonym porządku. Ostatnie wersje programu nie mają już tego rodzaju wad.

Program filePro 16 Plus wyposażono w wiele nowych cech szczególnie przydatnych dla programistów aplikacyj-

nych. Do tych nowych właściwości należą np.: dobrze zaprojektowany ekranowy samouczek elektroniczy (ang. customized help screens), program uruchomieniowy do przetwarzania, zwiększona szybkość przetwarzania dzięki prekompilowaniu tablic przetwarzania, 31 dodatkowych poleceń i funkcji do przetwarzania, rozszerzone badanie błędów w funkcjach bloków przetwarzania, 3 dodatkowe pola utrzymywane systemowo, 12 dodatkowych instrukcji do określania menu użytkownika (ang. user-menu command-line instructions), rozbudowane funkcje wyjściowe, a także kwalifikatory nazw plików, umożliwiające wielokrotne użycie tych samych struktur plikowych, plansz i raportów.

* * *

Programy filePro 16 i filePro 16 Plus niewątpliwie należą do najlepszych systemów zarządzania bazami dla mikrokomputerów. Dobrze spełniają one potrzeby użytkowników oraz twórców oprogramowania i z tego powodu na pewno zasługują na uwagę.

Oprac. **WIKTOR RZECZKOWSKI**
na podstawie BYTE, No 12, 1986

Samotesty

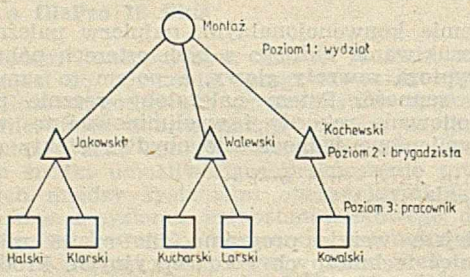
V.D. Podejście hierarchiczne w projektowaniu baz danych

Samotesty, opracowane i opublikowane przez Association for Computing Machinery, nie stanowią standardu egzaminacyjnego i są przeznaczone wyłącznie do samodzielnego sprawdzania własnej wiedzy [1]).

PYTANIA

11. Które z poniższych stwierdzeń są prawdziwe w odniesieniu do systemu hierarchicznej bazy danych?
   a) pomiędzy każdymi dwoma typami rekordów należących do drzewa bazy danych może występować dowolna liczba typów powiązań,
   b) zbiór ścieżek hierarchicznych w drzewie bazy danych odpowiada relacji w pierwszej postaci normalnej,
   c) istnieje funkcjonalna zależność pomiędzy artybutem występującym w rekordzie podrzędnym, a atrybutem występującym w rekordzie korzenia drzewa bazy danych,
   d) w jednym drzewie bazy danych nie można bezpośrednio odwzorować związków typu wiele-wiele (ang. many-to-many).

12. Z chwilą wybrania jednego z węzłów drzewa bazy danych rekordu lub segmentu można się nim posłużyć do jednoznacznego wyróżnienia zbioru odpowiadających mu węzłów wstępnych (ang. ancestor) i zbioru odpowiadających mu węzłów zstępnych (ang. descendent). Takie postępowanie, zwane „wymiataniem" (ang. broom), można stosować do kwalifikowania rekordów, które mają być wyszukiwane w hierarchicznej bazie danych. Niech drzewo bazy danych będzie określone jak na poniższym rysunku, a ciąg



pytań skierowanych do tej bazy danych ma postać następującą:

I. PRINT NAZWISKO-BRYGADZISTY WHERE BRYGADZISTA IN WYMIATANIE (NAZWISKO-PRACOWNIKA='KUCHARSKI')

II. PRINT NAZWISKO-BRYGADZISTY WHERE BRYGADZISTA IN WYMIATANIE (NAZWISKO-PRACOWNIKA='KUCHARSKI') AND (NAZWISKO-PRACOWNIKA='LARSKI')

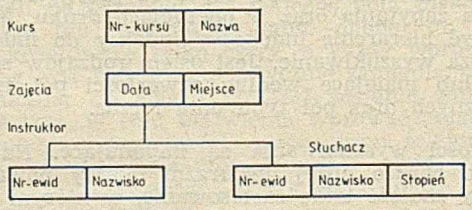
III. PRINT NAZWISKO-BRYGADZISTY WHERE BRYGADZISTA IN WYMIATANIE (NAZWISKO-PRACOWNIKA='KUCHARSKI') AND BRYGADZISTA IN WYMIATANIE (NAZWISKO-PRACOWNIKA='HALSKI')

IV. PRINT NAZWISKO-BRYGADZISTY WHERE BRYGADZISTA IN WYMIATANIE (NAZWISKO-PRACOWNIKA='KUCHARSKI') AND BRYGADZISTA IN WYMIATANIE (NAZWISKO-PRACOWNIKA='LARSKI')

Wskaż prawidłowy ciąg odpowiedzi:

   a) I — Walewski; II — Walewski; III — nie znaleziono; IV — Walewski
   b) I — Walewski; II — nie znaleziono; III — nie znaleziono; IV — Walewski
   c) I — Walewski; II — Walewski; III — nie znaleziono; IV — nie znaleziono
   d) I — Walewski; II — Walewski; III — Jankowski, Walewski; IV — Walewski

13. Rozpatrzmy poniższą hierarchiczną bazę danych:



oraz następujące trzy pytania w języku DL/I (język zapytań systemu zarządzania hierarchiczną bazą danych IMS):

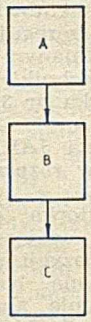
```
I.      GU      KURS
                ZAJĘCIA(MIEJSCE='MOKOTÓW')
II.     GU      KURS(KURS='M23')
                ZAJĘCIA(DATA='86-10-25')
        GO TO NP.  SŁUCHACZ
    NP  GNP     ZAJĘCIA(MIEJSCE='MOKOTÓW')
III.    GU      KURS
                SŁUCHACZ
    NS  GN      SŁUCHACZ
        GO TO NS.
```

Dobierz do każdego pytania odpowiadające mu określenie werbalne z poniższego zestawu:

[1]) Self-Assessment Procedure V. Part D. Hierarchical Database Approach (questions 14, 16—18). Communications of the ACM, Vol. 21, No. 8 (August 1978), pp. 687—693. Przedruk za generalnym zezwoleniem stowarzyszenia ACM na wykorzystanie Self-Assessment Procedure V do celów niekomercyjnych.

a) podaj wszystkie wystąpienia słuchaczy odbywających zajęcia na Mokotowie, począwszy od pierwszego uczestnika najwcześniejszego kursu,
b) podaj wszystkich słuchaczy uczestniczących w zajęciach na kursie M23 w dniu 25 października 1986,
c) podaj nazwisko pierwszej osoby z którejś listy słuchaczy mokotowskich,
d) podaj pierwsze wystąpienie zajęć na Mokotowie.

14. Rozważmy następujący logiczny pogląd w systemie IMS:



oraz cztery opisy fizycznych baz danych:

a) A jest fizycznym poprzednikiem B
B jest fizycznym poprzednikiem C
b) A jest logicznym poprzednikiem B
B jest logicznym poprzednikiem C
c) A jest fizycznym poprzednikiem B
C jest fizycznym poprzednikiem D
D jest logicznym następnikiem A
B jest wirtualnym obrazem D (segmenty sparowane)
d) A jest fizycznym poprzednikiem D
D jest logicznym następnikiem B
C jest fizycznym następnikiem B

Który z opisów a), b), c), d) pozwala wywnioskować podany na rysunku logiczny pogląd?

ROZWIĄZANIA

11 — b, d; 12 — b; 13 — 1d, IIb, IIIa; 14 — a, c.

LITERATURA I KOMENTARZE

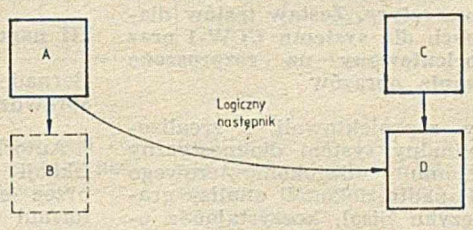
ad 11. Wiederhold G.: Data Base Design, McGraw-Hill, 1977, pp. 342, 426
Tsichritzis D.C. and Lochovsky F.H.: Data Base Management Systems, Academic Press, 1977, pp. 99—105

ad 12. Wiederhold, ibid., pp. 426—429,
ACM Computing Surveys, Special Issue on Data-Base Management Systems, Vol. 8, No 1, pp. 110—111, 116—118, March 1976

ad 13. Date C.J.: Wprowadzenie do baz danych. WNT, Warszawa, 1981, str. 221—225
Martin J.: Organizacja Baz Danych, WNT, Warszawa, 1983, str. 158—182

ad 14. Curtice R.M.: Access Mechanisms and Data Structure Support in Data Base Management Systems. Q.E.D. Monograph Series, pp. 35—43
Date, ibid., str. 252—273

W przypadku c) mamy:



Adaptowali z angielskiego:
ADAM B. EMPACHER, JERZY L. ROSSOWSKI

Konkurs PTI na najlepsze prace magisterskie z informatyki rozstrzygnięty

Zakończył się III Ogólnopolski Konkurs Polskiego Towarzystwa Informatycznego na najlepsze prace magisterskie z informatyki, zorganizowany przez Dolnośląski Oddział PTI. Na swym drugim posiedzeniu w dniu 12 grudnia 1986 r. które się odbyło w Centrum Obliczeniowym Politechniki Wrocławskiej, Jury konkursu w składzie: prof. dr hab. Andrzej Blikle, doc. dr inż. Czesław Daniłowicz (przewodniczący), dr inż. Zbigniew Huzar, doc. dr hab. Maciej Sysło, dr inż. Zbigniew Szpunar (sekretarz), prof. dr hab. Władysław M. Turski i doc. dr hab. inż. Jan Zabrodzki — powzięło następujące decyzje.

Pierwszą nagrodę w wysokości 25 tys. zł otrzymali **Jolanta** i **Jacek Łaboccy** za pracę pt. „Oprogramowanie systemowe specjalizowanych mikrokomputerów dla zastosowań w energetyce. Zestaw testów diagnostycznych dla systemu UCW-1 oraz język zorientowany na rozproszone przetwarzanie obrazów" (Uniwersytet Wrocławski, Wydział Matematyki, Fizyki i Chemii, Instytut Informatyki, opiekun dr inż. Jan Kwiatkowski).

Drugą nagrodę w wysokości 20 tys. zł otrzymał **Jerzy Chrząszcz** za pracę pt. „Specjalizowany system mikroprocesorowy do pomiaru wilgotności względnej powietrza przy wykorzystaniu metody psychometrycznej" (Politechnika Warszawska, Wydział Elektroniki, Instytut Informatyki, opiekun mgr inż. Janusz Rzeszut).

Trzecią nagrodę w wysokości 15 tys. zł otrzymała **Bernadetta Hała** za pracę pt. „Automatyczne sprawdzanie specyfikacji programów" (Politechnika Poznańska, Wydział Elektryczny, Ośrodek Informatyki, opiekun dr inż. Zdzisław Habasiński).

Jury konkursu przyznało ponadto trzy wyróżnienia po 8 tys. zł.

Otrzymali je:

Joanna Jarek za pracę pt. „Dekompozycja pytań w rozproszonych bazach danych" (Politechnika Wrocławska, Wydział Informatyki i Zarządzania, Centrum Obliczeniowe, opiekun dr Zygmunt Mazur);

Przemysław Rokita za pracę pt. „Program szeregujący wysokiego poziomu (HLS) dla systemu operacyjnego George 3" (Politechnika Warszawska, Wydział Elektroniki, Instytut Informatyki, opiekunka mgr inż. Ilona Bluemke) oraz

Ewa Kołczyk i **Małgorzata Malicka** za szczególne walory dydaktyczne pracy pt. „Materiały pomocnicze do nauczania języka Logo w szkole śred-

niej" (Uniwersytet Wrocławski, Wydział Matematyki, Fizyki i Chemii, Instytut Informatyki, opiekunki: dr Ewa Gurbiel i dr Helena Krupicka).

Laureatom konkursu serdecznie gratulujemy!

Barbara Osuchowska
rzecznik prasowy PTI

Omówienie nagrodzonych i wyróżnionych prac

I nagroda:

Jolanta i Jacek ŁABOCCY — **Oprogramowanie systemowe specjalizowanych mikrokomputerów dla zastosowań w energetyce. Zestaw testów diagnostycznych dla systemu UCW-1 oraz język zorientowany na rozproszone przetwarzanie obrazów**

Autorzy zaprojektowali i zrealizowali oryginalny system diagnostyczny dla systemu mikrokomputerowego UCW-1. Ponadto dokonali analizy gramatyki języka Sisal, korzystając z opracowanego do tego celu systemu badania gramatyki Grammar. Na podstawie otrzymanych wyników zaproponowali modyfikacje języka Sisal, umożliwiające stosowanie automatycznego analizatora składni oraz rozwinięcie języka o instrukcje graficzne.

Przyznając pierwszą nagrodę Jury konkursu podkreśliło duże znaczenie praktyczne opracowanej metody diagnozowania systemu mikrokomputerowego wykorzystywanego przez państwową dyspozycję mocy. Komisja uwzględniła także nowoczesność zaproponowanych w pracy rozwiązań teoretycznych z zakresu przetwarzania sterowanego przepływem danych.

II nagroda:

Jerzy CHRZĄSZCZ — **Specjalizowany system mikroprocesorowy do pomiaru wilgotności względnej powietrza przy wykorzystaniu metody psychometrycznej**

Autor zaprojektował i zrealizował mikroprocesorowy system do pomiaru wilgotności względnej powietrza. Użyte przez dyplomanta urządzenie składa się z dwóch części: przetwornika psychometrycznego wilgotności względnej powietrza i współpracującego z nim mikroprocesorowego systemu sterująco-przetwarzającego.

Przyznając drugą nagrodę Jury konkursu podkreśliło, że Autor w sposób praktyczny wykazał, iż wykorzystanie mikroprocesorów w urządzeniach pomiarowych umożliwia budowę przyrządów o parametrach metrologicznych i ergonomicznych znacznie lepszych niż w przyrządach tradycyjnych.

III nagroda

Bernadetta HAŁA — **Automatyczne sprawdzenie specyfikacji programów**

Autorka opracowała metodę weryfikacji programów współbieżnych przez sprawdzenie prawdziwości ich formuł specyfikacyjnych w modelach grafowych tych programów. Konstruując i uruchamiając złożony program TESTER sprawdziła, że ta semantyczna metoda weryfikacji jest praktycznie realizowalna.

Przyznając trzecią nagrodę Jury konkursu podkreśliło bardzo dobrą znajomość teoretyczną i praktyczną trudnej problematyki programowania współbieżnego oraz weryfikacji programów.

Wyróżnienia:

Przemysław ROKITA — **Program szeregujący wysokiego poziomu (HLS) dla systemu operacyjnego George 3**

Autor opracował program szeregowania zadań, który jest rozszerzeniem systemu operacyjnego George 3. Dzięki użyciu algorytmu uwzględniającego charakterystykę potrzeb użytkownika uzyskano znaczne skrócenie czasu oczekiwania na wykonanie zadań.

Wyróżniając pracę, Jury konkursu podkreśliło jej wysoki poziom merytoryczny oraz znaczenie praktyczne, udokumentowane wdrożeniem programu szeregującego do eksploatacji w uczelnianym ośrodku obliczeniowym Politechniki Warszawskiej (por. Informatyka nr 3, 1987).

Joanna JAREK — **Dekompozycja pytań w rozproszonych bazach danych**

Autorka opracowała i oprogamowała w języku Pascal algorytm dekompozycji pytań w systemie z rozproszoną bazą danych. Zilustrowała działanie algorytmu dla pytań sformułowanych za pomocą podstawowych operacji języka, opartego na algebrze relacji.

Wyróżniając pracę Jury konkursu wzięło pod uwagę wysoki poziom merytoryczny pracy oraz dobre przygotowanie Autorki w zakresie teoretycznych i praktycznych problemów rozproszonych baz danych.

Ewa KOŁCZYK, Małgorzata MALICKA — **Materiały pomocnicze do nauczania języka Logo w szkole średniej**

Autorki opracowały materiały dydaktyczne do nauczania języka programowania Logo. Podały systematyczny opis języka oraz interesujące wskazówki metodyczne, które mogą być pomocne w nauczaniu elementów informatyki w szkołach średnich.

Wyróżniając pracę Jury konkursu podkreśliło jej szczególne walory dydaktyczne.

W imieniu Jury Konkursu
CZESŁAW DANIŁOWICZ
i ZBIGNIEW SZPUNAR

Kompilatory Ady na komputery IBM PC

Jeszcze trzy lata temu twórcy oprogramowania systemów wojskowych niecierpliwie czekali na zatwierdzone i efektywne kompilatory Ady. Dzisiaj są one dostępne nawet na komputery osobiste i Adę wykorzystuje się również w przedsięwzięciach cywilnych. Początkowo, kompilatory Ady były nabywane przez firmy zajmujące się inżynierią oprogramowania — eksperymentujące z nowym językiem. Gdy po okresie wstępnego zainteresowania rynek uspokoił się, wskutek przejściowego spadku sprzedaży i ogólnej słabości przemysłu komputerowego dystrybutorzy kompilatorów Ady, szukając większych rynków zbytu, skupili uwagę na indywidualnych stanowiskach roboczych.

Firmy wchodzące na rynek opracowują pośpiesznie kompilatory przeznaczone na komputery osobiste, mając nadzieję wykorzystać oczekiwany wzrost popytu wywołany przez rozpowszechnienie tego sprzętu. Zainteresowanie Adą wynika w głównej mierze z tego, że oprogramowanie napisane w tym języku ma cechy przenośności i wieloużywalności (ang. reusability). Ada zezwala również — podobnie jak Modula 2 — na rozłączną kompilację, co oznacza, że kilku programistów może pracować niezależnie nad jednym projektem. Dopóki przestrzegają oni specyfikacji odrębnych pakietów, to połączenie ich kodów w większy program odbywa się bez kłopotów.

Kierownik zespołu projektowego Ady, obecnie prezes firmy Alsys, Jean D. Ichbiah, uważa że przełomowym momentem będzie dla niego opanowanie rynku komputerów osobistych. Przewiduje on również, że w najbliższej przyszłości 70—90% oprogramowania będzie tworzone na stanowiskach roboczych. Kompilatory firmy Alsys są przeznaczone dla komputerów osobistych pracujących w sieci, a sama firma prowadzi agresywny marketing objawiający się ostrym współzawodnictwem cenowym i szeroką akcją reklamową.

Chociaż Alsys sprzedaje również kompilatory na takie komputery jak SUN lub VAX, to był pierwszą firmą,

która uzyskała atestację (zaświadczenie Departamentu Obrony USA) wskazujące, że kompilator przeszedł wszystkie wymagane testy kompilatora Ady na IBM PC/AT. Później firma przeprowadziła także atestację na sześć komputerów zgodnych z IBM PC/AT: Sperry PC/IT, Compaq Deskpro 286, Zenith Z-200, Tandy 3000 HD, Hewlett-Packard Vectra i Goupil G40.

Następnie do współzawodnictwa przystąpiła firma Verdix, oferując system VADS (Verdix Ada Development System), oraz firma Oasys, która przeniosła VADS na kartę stanowiącą dodatkowe wyposażenie komputerów PC/XT, PC/AT i zgodnych. PC/VADS pracuje na płycie Oasys PC opartej na procesorze 32032 (firmy National Semiconductor) wyposażonym w pamięć 2 MB lub 4 MB. System PC/VADS uzyskał atestacje na komputery IBM PC/AT, Zenith 248, Sperry IT, Compaq PC, Wyse PC, AT&T 6300+ i NCR PC.

Kompilator firmy Alsys

Firma Alsys reklamuje swój kompilator jako pierwszy mający bezpośredni dostęp do pełnej przestrzeni adresowej IBM PC/AT w trybie wirtualnym. Prezes Ichbiah twierdzi, że teraz wszyscy skierują się w stronę Ady, nie dlatego, że lubią ten język, ale dlatego, że pozwala robić rzeczy do tej pory niewykonalne. Jest to jedyny kompilator pozwalający w pełni wykorzystać możliwości IBM PC/AT, gdyż wszystkie inne są objęte ograniczeniami systemu operacyjnego MS-DOS. Jednakże sceptycy twierdzą, że firma Alsys była zmuszona do użycia trybu wirtualnego z powodu nierozsądnych rozmiarów kompilatora.

Żeby udostępnić środowisko programistyczne zgodne z innymi własnymi kompilatorami Ady, firma Alsys przeniosła kod źródłowy istniejącego kompilatora na IBM PC/AT. Ponieważ kompilatory Ady są bardzo skomplikowanymi produktami, to na ogół preferuje się pracę z dobrze sprawdzonym kodem. W rezultacie postać źródłowa kompilatora Alsys, napisana w Adzie dla dużych środowisk takich jak VAX, wytwarza zbyt duży kod wynikowy nie mieszczący się na IBM PC/AT. Sam kompilator może działać na PC/AT, ale tylko przy wykorzystaniu dodatkowej pamięci udostępnianej w trybie adresowania wirtualnego. Mimo to, kompilator wytwarza kod wynikowy działający na wszystkich komputerach osobistych IBM i zgodnych, a firma Alsys podkreśla jego efektywność. Kompilator Ady przebadano dla kilku programów wzorcowych i porównano je z testami wykonanymi w Uniwersytecie Stanforda dla kompilatorów Turbo Pascala i Lattice C. Wyniki poszczególnych testów różnią się, ale ogólne podsumowanie wskazuje, że kompilator Alsys może z powodzeniem konkurować z obydwoma porównywanymi kompilatorami (tab. 1, 2). Przedstawiciel firmy Alsys stwierdził, że głównym celem tych badań nie było ustalenie dokładnej miary względnej jakości różnych kompilatorów, ale wykazanie, że dla Ady można wytworzyć kod, którego efektywność jest conajmniej porównywalna z efektywnością translatorów innych języków.

Tabela 1. Testy porównawcze wykonane dla kompilatorów Lattice C (wersja 2.15), Turbo Pascala (wersja 2.00A) i Alsys Ada, na komputerze IBM PC/AT (8 MHz) z koprocesorem 80287. Programy w języku C wykonano z koprocesorem, w Pascalu bez koprocesora. Kontrola w kompilatorze Ady czyni niedostępnymi chronione segmenty pamięci IBM PC/AT. Czas jest podany w sekundach.
A1 — kompilator Ady z kontrolą,
A2 — kompilator Ady bez kontroli,
C1 — kompilator Lattice C, mały model,
C2 — kompilator Lattice C, duży model.

Rodzaj testu	Alsys Ada (A1)	Alsys Ada (A2)	Lattice C (C1)	Lattice C (C2)	Turbo Pascal
Permutacje siedmiu liczb całkowitych (wykonane 5 razy, rekurencyjnie)	3,34	2,31	2,86	2,41	4,56
Mnożenie macierzy liczb całkowitych (2 macierze 40×40)	6,09	5,10	12,31	2,53	3,52
Mnożenie macierzy liczb zmiennoprzecinkowych (2 macierze 40×40 liczb 32-bitowych)	11,09	9,87	40,48	29,23	79,58
Szybka transformacja Fouriera (256 punktów 20 razy)	13,46	11,85	61,29	52,78	121,33
Problem 8 hetmanów (rozwiązany 50 razy)	2,75	1,81	4,11	1,31	2,96
Wieże z Hanoi (14 krążków, rozwiązanie rekurencyjne)	3,51	2,25	2,31	2,26	4,40

Tabela 2. Ogólne wyniki testów kompilatorów Lattice C (wersja 2.15), Turbo Pascala i Alsys Ady. Testy obejmowały: permutacje, wieże z Hanoi, mnożenie macierzy liczb całkowitych, mnożenie macierzy liczb zmiennoprzecinkowych, rozwiązanie skomplikowanej łamigłówki, szybkie sortowanie, sortowanie bąbelkowe, sortowanie drzewa binarnego 5000 liczb całkowitych, szybką transformację Fouriera i wyznaczanie funkcji Ackermana.

Kompilator	Bez liczb zmiennoprzecinkowych	Z liczbami zmiennoprzecinkowych
Alsys Ada (A1)	98,1	24,55
Alsys Ada (A2)	76,83	21,72
Lattice C (C1)	82,78	101,77
Lattice C (C2)	64,64	82,06
Turbo Pascal	117,21	200,91

Znaczenie edukacji

Obecnie, Ada nie jest tak popularnym narzędziem do tworzenia uniwersalnego oprogramowania na komputery osobiste lub klasy VAX, jakim jest język C. Przypuszcza się jednak, że w związku z rosnącą znajomością Ady wśród programistów, wyniesioną ze studiów, wkrótce Ada zostanie językiem wiodącym. Ostatnie ankiety wykazują, że studenci, którzy poznali Adę jako pierwszy język programowania, nie chcą przechodzić na inne języki.

Alsys i inne firmy wchodzące na rynek indywidualnych stanowisk roboczych zdają sobie sprawę z wagi właściwej edukacji i wzmagają działania edukacyjne. Nawet firma Telesoft, która zlekceważyła fakt stworzenia kompilatora Ady na IBM PC, aktywizuje działalność w tym kierunku, chociaż nie ma konkretnych planów marketingowych, czy przeprowadzenia atestacji kompilatora.

B. Sherman, odpowiedzialny za marketing w firmie Telesoft, uważa, że szeroka edukacja jest najistotniejsza dla rozpowszechnienia Ady. Dlatego firma zaoferowała swoje kompilatory ze znaczną zniżką, choć stawiając pewne warunki. W konsekwencji, ponad 200 uniwersytetów wykorzystuje te kompilatory w salach wykładowych.

Wydaje się, że na rynku edukacyjnym przewodzi na razie firma Alsys oferująca pakiet Alsys Compiler Plus, zawierający kompilator i zestaw programów wspomagających nauczanie. Pakiet kosztuje 4995 dolarów i pracuje na stanowiskach roboczych SUN i Apollo, a także komputerach IBM PC/AT i zgodnych. Firma sprzedaje również „elektroniczny" podręcznik języka o nazwie Ada QUERY.

Firma RR Software oferuje na rynek edukacyjny kompilator, który był bardzo popularny wśród wykładowców kilka lat temu (por. Informatyka, 4, 5, 1987). Jest to Janus/Ada, na razie nie mająca atestu. Firma pracuje w dalszym ciągu nad pełnym kompilatorem, dodając kolejne cechy przybliżające go do implementacji pełnej Ady. Jednakże Janus/Ada ma pewną cechę wyróżniającą we współzawodnictwie rynkowym — jest to cena. Podstawowy kompilator kosztuje 99 dolarów. Dla tych, którzy nie potrzebują bardziej egzotycznych cech języka, takich jak wielozadaniowość, Janus/Ada jest najtańszym sposobem poznania Ady. Ponieważ oprogramowanie to nie jest zabezpieczone przed kopiowaniem, liczbę używanych kompilatorów szacuje się na 8000, z czego prawdopodobnie 4000—5000 jest legalne.

Istnieje również wersja tego kompilatora dla procesora Z80, pracująca pod kontrolą systemu operacyjnego CP/M. Przedstawiciel firmy twierdzi,

że armia amerykańska zakupiła większą ilość komputerów Zenith 100 (opartych na Z80) i jest zainteresowana implementacją Ady na te komputery. Wydaje się, że wojsko może stanowić ukryty rynek wchłaniający tanie kompilatory Ady. Jednakże rynek ten będzie dopóty nieaktywny, dopóki tanie kompilatory nie uzyskają atestacji lub atestowane nie staną.

Kompilatory firmy Artek

Wśród wchodzących na rynek dystrybutorów Ady zwraca na siebie uwagę firma Artek (Islandia), planująca sprzedaż tanich kompilatorów na komputery IBM PC i zgodne (por. Informatyka, 7—8, 1986, str. 39). Przewiduje ona sprzedaż gotowego do atestacji kompilatora w cenie poniżej 1000 dolarów już od połowy 1987 roku. Jeśli firma dotrzyma terminów, to na pewno zajmie mocną pozycję na rynku „taniej" Ady. Tymczasem, za 495 dolarów można kupić niekompletny kompilator, w którym zaimplementowano wszystkie ważniejsze cechy Ady oprócz wielozadaniowości i typów stałoprzecinkowych. Kompilator pracuje komputerach IBM PC, PC/XT, PC/AT i zgodnych, wyposażonych w 384 KB pamięci i dwa napędy dysków elastycznych. Wersja atestowana wymaga 640 KB pamięci.

W przeciwieństwie do kompilatora firmy Alsys, kompilator Arteka nie kompiluje programu źródłowego Ady na kod wynikowy procesora 8088 lub 80286, ale na język pośredni zwany A-kodem. Następnie A-kod jest wykonywany bezpośrednio w środowisku wykonawczym, dostarczanym z kompilatorem, lub dalej redukowany do wykonywalnego kodu wynikowego za pomocą dołączonych programów usługowych. A-kod stanowi implementację zoptymalizowanej dla Ady listy rozkazów RISC ukierunkowanej na stos wirtualny.

Pakiet dostarczany przez firmę Artek zawiera środowisko programowe Ady (APSE) komunikujące się z użytkownikiem przez menu, w sposób podobny do Turbo Pascala. Wbudowane narzędzia zawierają pełnoekranowy edytor pokazujący położenia i rodzaje błędów kompilacji. Edytor umożliwia tworzenie makrodefinicji i zapisywanie ich na pliku. Konsolidator i administrator biblioteki umożliwiają pielęgnowanie biblioteki Ady, jak również łączenie programów.

Emulator i program uruchomieniowy wykonują pośredni A-kod przez zaimplementowanie maszyny wirtualnej o liście rozkazów napisanej w A-kodzie. Program uruchomieniowy oprócz typowych czynności, takich jak śledzenie i krokowe wykonywanie programu, umożliwia przejrzenie zawartości poszczególnych rejestrów, stosu i stogu. Ponieważ program uruchomieniowy pracuje w A-kodzie, firma Artek twierdzi, że użytkownik ma lepsze możliwości w porównaniu z tradycyjnymi mechanizmami symbolicznymi. Disasembler umożliwia deasemblację plików wynikowych, bezpośrednio lub z biblioteki programu. W celu wytworzenia kodu wynikowego generator kodu wrodzonego (ang. native code generator) przekształca A-kod na wykonywalne pliki typu EXE. Generator kodu wykorzystuje różne techniki optymalizacyjne.

Firma Artek projektuje również dwa skrośne kompilatory dla systemu opartego na IBM PC, jako komputerze macierzystym. Jeden z nich będzie wytwarzał kod procesora National Semiconductor 32032, drugi — rodziny procesorów Motorola 68000. Według prezesa firmy, w obydwu wypadkach generatory kodu wykorzystują jako kod źródłowy A-kod wytworzony przy pierwszym przejściu kompilatora głównego. W związku z tym, aby w istniejącym systemie uzyskać kompilację skrośną, użytkownicy muszą dołączyć tylko odpowiedni generator kodu.

Państwowy Instytut Badawczy w Islandii przyznał firmie Artek fundusze na prace nad stworzeniem mikroprocesora zoptymalizowanego dla Ady. Ostatnio zbudowany prototyp wykonuje 2 miliony operacji stosowych na sekundę. Przedstawiciel firmy twierdzi, że procesor będzie gotowy w połowie 1988 roku.

Kompilator Verdix/Oasys

Do tej pory Verdix i Alsys, wiodące firmy na rynku zatwierdzonej Ady, polegają na dodatkowym sprzęcie zwiększającym efektywność kompilatorów. Na przykład, firma Alsys dostarcza dodatkową płytę pamięci do komputerów IBM PC/AT i zgodnych. W tej pamięci jest wystarczająco dużo miejsca na załadowanie kompilatora, który zajmuje 3 MB. Kompilator PC/ /VADS firmy Oasys nie pracuje na mikroprocesorze 80826 lub 8088, ale na płycie wyposażonej w mikroprocesor serii 32000, pod kontrolą systemu operacyjnego Unix V.5.

System jest tak skonfigurowany, że użytkownik nie musi znać Unixa, aby wykorzystywać kompilator Ady. Środowisko MS-DOS jest przeznaczone dla użytkowników, którzy nie chcą wyjść poza nie. Obecność Unixa zapewnia natomiast stały dostęp do środowiska wieloużytkowego i wielozadaniowego. Korzyść z używania procesora 32032 polega na efektywniejszej pracy systemu, przewyższającej minikomputery VAX. Procesor 32032 umożliwia rozszerzenie przestrzeni adresowej IBM PC/AT do 16 MB, a ponadto zapewnia wykonywanie 5 milionów operacji na sekundę. Krytycy tego modelu mocno podkreślają, że generuje on kod wykonywalny tylko przez siebie. Ograniczone środowisko docelowe i konieczność używania tej samej płyty jako środowiska wykonawczego niweluje pewne oszczędności wynikające z możliwości wykorzystania stacji roboczych opartych na komputerach osobistych. Ponieważ dla kompilatora Verdix procesor 8086 nie może być macierzystym ani docelowym, prawdziwy problem pojawia się wtedy, gdy ktoś chce kupić większą liczbę jednostek.

Argumentuje się, że prawdziwa wartość indywidualnych stanowisk roboczy (nie przeznaczonych do tworzenia aplikacji na komputery osobiste) polega na udostępnieniu wszechstronnego, ale taniego środowiska, w którym można tworzyć kod źródłowy Ady, kompilowany według potrzeb w dowolnym komputerze macierzystym, dla dowolnego komputera docelowego. Sztywna normalizacja języka gwarantuje przenośność wszystkich funkcji niezależnych od maszyny. Z tego punktu widzenia komputery osobiste uzyskują — w dziedzinie tworzenia oprogramowania — możliwości podobne do tych jakie ma obecnie VAX. Tak więc, ta klasa komputerów może zastąpić VAX-y — prawdopodobnie najczęściej wykorzystywane w różnych zastosowaniach jako komputery macierzyste Ady.

Inne kompilatory

Większość firm programistycznych, tworząc swój pierwszy kompilator Ady, wybiera drogę wytyczoną przez firmy Artek i RR Software. Sprzedają kompilatory częściowe lub bez atestacji, co pozwala pokryć późniejsze koszty związane z przeprowadzeniem atestacji. Przy takim podejściu konsultacje z użytkownikiem powinny być częścią oferty. Większość klientów przed nabyciem kompilatora domaga się stałego kontaktu z wytwórcą. W rezultacie specjaliści tracą czas na niekończące się rozmowy i lokalizowanie błędów — nieuniknionych w takich przedsięwzięciach. Pomimo takich kłopotów, wprowadzenie na rynek niepełnego, ale pracującego kompilatora jest chyba najwłaściwszym krokiem na drodze do atestacji. Inną firmą postępującą w ten sam sposób jest General Systems, której kompilator o nazwie GS/ /ICC — adaptowany z kompilatora Irvine Computer — jest jednym z pierwszych porządnych, ale jeszcze nieatestowanych kompilatorów. Atestacja kompilatora GS/ICC miała być przeprowadzona pod koniec 1986 roku, a pierwsze dostawy przewidywano na marzec 1987. Kompilator pracuje na komputerach IBM PC/XT z pamięcią 640 KB i wymaga koprocesora 8087. Wersja bez atestu — obecnie dostępna — kosztuje 1999 dolarów.

W New York University powstało inne narzędzie dla programujących w Adzie. Jest to interpreter o nazwie Ada/ED-C będący pełną implementacją Ady, który przeszedł wszystkie testy ACVC (ang. Ada compiler validation capability) wersji 1.7 i uzyskał atest. Interpreter, pracuje na komputerach IBM PC/XT, IBM PC/AT i zgodnych, wyposażonych w stały dysk i 640 KB pamięci RAM. Na komputerze IBM PC/AT pod kontrolą systemu operacyjnego Concurrent DOS 286 może wykorzystywać 4 MB pamięci, pracując tylko o połowę wolniej niż na stanowisku roboczym SUN-2. Z powodu dużej zajętości pamięci, interpreter na komputerze o pamięci operacyjnej 640 KB przyjmuje programy o długości kilkuset linii i jest przeznaczony głównie do celów edukacyjnych. Jego cena wynosi 95 dolarów.

Podejście firmy Sequent Computer Systems

Prawdopodobnie, spory na temat jak najlepszego wykorzystania komputerów osobistych do tworzenia oprogramowania w Adzie będą trwały dopóty, dopóki sam status tych komputerów będzie niewyjaśniony. Strategia koncernu IBM wyznacza im miejsce w klasie satelitarnych stanowisk roboczych sprzężonych z dużym komputerem.

Firma Sequent Computer Systems wykorzystuje przyjazne sprzężenie człowiek-maszyna charakteryzujące komputery IBM PC i dołącza je do własnego sprzętu. Powstałe stacje programistyczne BSS/PC są wykorzystywane do tworzenia oprogramowania w Adzie jako inteligentne terminale lub stacje graficzne połączone z komputerami Balance. Zapewniają one operowanie techniką okien i pełną komunikację. IBM PC ma dostęp do stałego dysku komputera Balance w trybie wirtualnym. W rzeczywistości, sprzężenie IBM PC lub PC/AT z dyskiem wirtualnym powoduje pracę szybszą w porównaniu z sytuacją, gdy IBM PC/AT wykorzystuje własny, istniejący fizycznie dysk. Użycie przez komputery Balance plików zapisywanych według konwencji systemu MS-DOS i hierarchicznej struktury skorowidzów zapewnia większe możliwości pamięciowe.

Użytkownik pracujący w sieci BSS/PC może wykorzystywać Dynix (wersja Unixa firmy Sequent) i programy aplikacyjne w systemie DOS, tworząc wiele okien, z których każde odpowiada odrębnej powłoce Dynixa. Terminale systemu są oparte na programie GEM (ang. graphical environment manager) firmy Digital Research, wykorzystującym ekran z mapą bitową i mysz. Firma dostarcza również oprogramowanie spełniające wymagania standardów GKS (ang. graphical kernel system) i VDI (ang. virtual device interface), umożliwiające korzystanie z metaplików VDI. Graficzne programy aplikacyjne powstałe przy użyciu GKS i VDI pracują również na komputerach IBM PC.

Podobnie jak Oasys, firma Sequent przeniosła VADS firmy Verdix na swoje komputery oparte na procesorze 32032. Pracując w systemie BSS/PC twórcy oprogramowania mogą uzyskać dostęp do dużego środowiska wspomagającego tworzenie oprogramowania systemowego, jednocześnie nie tracąc nic z przyjaznego sprzężenia z użytkownikiem, tak charakterystycznego dla mikrokomputerów IBM PC.

Obecnie trwają spory, który system jest użyteczniejszy — płyta wykorzystująca IBM PC w operacjach we-wy, czy kompilator pracujący na komputerze osobistym i wytwarzający jego kod wrodzony.

Znaczenie atestacji

Inną kwestią sporną wymagającą rozstrzygnięcia jest konieczność uzyskania atestacji. Większość dystrybutorów niezatwierdzonych kompilatorów przyznaje, że atestacja staje się niezbędna, szczególnie teraz, gdy istnieją już zatwierdzone kompilatory Ady na komputery osobiste.

Wydaje się, że większość klientów czeka z zamawianiem większej liczby sztuk na zatwierdzenie kompilatora. Obecnie, kilka ośrodków wojskowych może zamówić wszystko, co ma atest.

Firma Artek przyznaje, że atestacja jest koniecznością. Wielu jej potencjalnych klientów rezygnuje z zakupu po stwierdzeniu, że kompilator nie ma atestacji. Przeprowadzenie atestacji jest więc obecnie kwestią kluczową dla tej firmy. Z drugiej strony, prezes firmy Artek uważa, że obecność na rynku wielu zatwierdzonych kompilatorów na komputery osobiste umacnia wiarygodność Arteka. Jak udowodniła firma Alsys, można stworzyć kompilator Ady na komputer osobisty.

Według firmy Telesoft problem atestacji jest również punktem newralgicznym. Argumentuje się, że nikt nie wybierze kompilatora niezatwierdzonego, gdy od kogoś innego może kupić zatwierdzony. Byłoby to świadomym pozbawieniem się możliwości poznania wszystkich cech Ady.

Jednakże, różnorodność sądów na temat kompilatorów Ady na komputery osobiste ilustruje podstawowe różnice między firmami programistycznymi. Firmy Alsys, Verdix i inne widzą komputery osobiste jako dodatkową płaszczyznę wykorzystania już istniejących kompilatorów. Nowi dystrybutorzy to małe firmy próbujące umocnić się na rynku za pomocą kompilatorów na komputery osobiste, przeznaczonych dla małych i początkujących firm programistycznych. Prawdopodobnie, są oni gotowi zaoferować niższe ceny i przystosować swoje produkty do wymagań sprzętowych. Wyrzekają się też chwilowo wielkiego rynku otwierającego się przed tym, kto przedstawi rozwiązania standardowe dla wielu środowisk. Potencjalnie istniejący rynek komputerów osobistych jest jednak dość duży i stabilny, ponieważ małe firmy programistyczne — jako nabywcy — są zwabione konkurencyjnymi cenami.

Przedstawiciel firmy RR Software stwierdza, że kompilator Janus/Ady i nowy pakiet Turbo Pascala mają taką samą cenę (99 dolarów) i dodaje że jego firma będzie opierać się na dotychczasowych klientach.

Chociaż niektórzy dystrybutorzy mogą nie przetrwać, to Jean Ichbiah przewiduje podjęcie przez nich działań zmierzających do rozszerzenia rynku Ady. Osiągają to dając kompilatory Ady w ręce każdego programisty. Programowanie w Adzie nie jest teraz droższe od programowania w Fortranie lub języku C.

Jakkolwiek ciągle jest za wcześnie na wyrokowanie, które firmy przetrwają i które kompilatory uzyskają największą popularność, to prawdopodobnie powstaną co najmniej trzy rodzaje ofert, proponujących kompilatory zróżnicowane pod względem przenośności, efektywności i ceny.

Oprac. MARIUSZ KUC na podstawie Computer Design, 15 Oct. 1986

Kto jest najszybszy

Czasopismo San Francisco Examiner zamieściło następujące porównanie mocy obliczeniowych i cen kilku aktualnie produkowanych komputerów:

Eta 10 jest obecnie najszybszym superkomputerem na świecie. W wyniku testów przeprowadzonych w ubiegłym roku osiągnięto maksymalną szybkość 7 (wobec planowanych 10) miliardów operacji zmiennoprzecinkowych na sekundę.

M. MACHURA

Typ komputera	Liczba operacji zmiennoprzecinkowych na sekundę (w milionach)	Cena (w milionach dolarów)
Eta 10 (8 procesorów)	10 000	23
Cray II	250	17
Cray XMP — 48	125	15,5
Convex C—1	2	0,5—1,5
Vax 8600	0,49	0,5
IBM 4361	0,46	0,18
IBM PC	0,00056	0,0025

Kto jest kim w IFIP

Jacques Fourot

Jacques Fourot, sekretarz IFIP od 1981 roku i przedstawiciel Francji w Zgromadzeniu Ogólnym od 1980 roku, urodził się w Paryżu, w roku 1928. Po uzyskaniu pierwszych stopni naukowych w dziedzinie nauk podstawowych w Ecole des Hautes Etudes Commerciales i w dziedzinie prawa na Uniwersytecie Paryskim rozpoczął pracę w Compagne des Machines Bull. Pracuje tam nieprzerwanie od 1952 roku.

Zaczynając jako stażysta w dziale sprzedaży, przeszedł różne stopnie kariery w tej firmie: przez stanowiska kierownika działu, kierownika sprzedaży w dziale przemysłowo-handlowym, kierownika sprzedaży promocyjnej (wprowadzając na rynek pierwszą generację komputerów firmy Buli), kierownika grupy sterującej planowaniem zaawansowanych systemów informacyjnych, szefa służby handlowej Honeywell-Bull, dyrektora marketingu CII-Honeywell Bull i wicedyrektora grupy sterującej wspólną strategią i marketingiem w firmie Bull.

W 1966 roku Jacques Fourot zastał przeniesiony do Stanów Zjednoczonych, gdzie spędził cztery lata w Bridgeport, w siedzibie grupy systemów informacyjnych firmy General Electric.

Jacques Fourot odgrywa główną rolę w organizacji AFCET (Association Francaise pour la Cybernetique Economique et Technijue), będącej francuskim członkiem IFIP, koordynując międzynarodową działalność AFCET i propagując jej osiągnięcia. Oprócz piastowania stanowiska sekretarza IFIP, pan Fourot jest wiceprzewodniczącym Komitetu IFIP ds. Łączności Międzynarodowej (ICIL) i działa na rzecz utrzymania stosunków między UNESCO i Komitetem IFIP Informatyka dla Rozwoju (ICID). Dzięki temu IFIP uzyskuje poparcie UNESCO dla podejmowanych działań, szczególnie w przypadku delegatów z krajów rozwijających się. Jest nadzieja, że nowy kształt stosunków między UNESCO i IFIP zostanie zrealizowany w Międzynarodowym Programie Informatycznym (IIP), opierającym się na technicznych zasobach IFIP.

Jacques Fourot i jego żona Brenda, właścicielka sklepu mody damskiej, mieszkają z trójką dzieci w centrum Paryża. Obok niezbyt zobowiązującej gry w tenisa jego ulubionym zajęciem jest wyszukiwanie nowych i interesujących restauracji, co w Paryżu jest zadaniem bez końca.

(MK)

WARUNKI PRENUMERATY NA 1987 R.

Prenumeratorzy zbiorowi — jednostki gospodarki uspołecznionej, instytucje i organizacje społeczne zamawiają prenumeratę dokonując wpłat na blankiecie „polecenie przelewu".

Prenumeratorzy indywidualni — osoby fizyczne zamawiają prenumeratę dokonując wpłaty w UPT lub NBP na blankiecie Wydawnictwa lub blankiecie NBP. Na odwrocie wszystkich odcinków blankietu należy wpisać tytuł czasopisma, okres prenumeraty, liczbę zamawianych egzemplarzy oraz wartość wpłaty.

Wpłacać należy na konto NBP III O/M Warszawa 1036-7490-139-11.

Prenumerata ulgowa — przysługuje wyłącznie osobom fizycznym — członkom SNT, studentom i uczniom szkół zawodowych. Warunkiem prenumeraty ulgowej jest poświadczenie blankietu wpłaty (przed jej dokonaniem) na wszystkich odcinkach pieczęcią Koła SNT, wyższej uczelni lub szkoły.

Sposób zamawiania prenumeraty taki sam jak dla prenumeraty indywidualnej.

Prenumerata ze zleceniem wysyłki za granicę — zamawia się tak jak prenumeratę indywidualną. Dodatkowo należy podać na blankiecie wpłaty nazwisko i dokładny adres odbiorcy. Cena prenumeraty ze zleceniem wysyłki za granicę jest dwukrotnie wyższa.

Przedpłaty na prenumeratę przyjmowane są w terminach:

— do 10 listopada na I kwartał, I półrocze i cały rok następny,
— do 28 lutego na II, III, IV kwartał i II półrocze,
— do 31 maja na III, IV kwartał i II półrocze,
— do 31 sierpnia na IV kwartał.

U w a g a!

Wpłaty na dwumiesięczniki przyjmowane są na okresy półroczne lub roczne.

Informacji o prenumeracie udziela — Zakład Kolportażu Wydawnictwa NOT-SIGMA, ul. Bartycka 20, 00-716 Warszawa, lub skr. poczt. 1004, 00-950 Warszawa, tel. 40-00-21 w. 249, 293, 297, 299 oraz 40-35-89 i 40-30-86.

Egzemplarze archiwalne czasopism — można nabyć za gotówkę w Klubie Prasy Technicznej w Warszawie ul. Mazowiecka 12, tel. 27-43-65 lub zamówić w Dziale Handlowym Wydawnictwa, ul. Bartycka 20, skr. poczt. 1004, 00-950 Warszawa, tel. 40-37-31, na rachunek dla instytucji lub za zaliczeniem pocztowym dla osób fizycznych.

Cena miesięcznika INFORMATYKA została ustalona na 150 zł za numer (50 zł — cena ulgowa).

Cena prenumeraty wg cennika					
kwartalna		półroczna		roczna	
normalna	ulgowa	normalna	ulgowa	normalna	ulgowa
450	150	900	300	1800	600

Przypominamy Czytelnikom, że 31 sierpnia upływa termin wnoszenia wpłaty na prenumeratę INFORMATYKI w czwartym kwartale bieżącego roku

Terminologia języka Logo (3)

dokończenie z III str. okładki

Warto jednakże, spojrzeć na tę terminologię z nieco innego punktu widzenia, łącząc instrukcje w grupy, w zależności od pełnionej funkcji, lecz nie różnicując ich pod względem treści. Najważniejsze kryterium takiego podziału uwzględnia istotne rozróżnienie instrukcji udostępniających wartość od instrukcji będących poleceniami kierowanymi do komputera. Nazwy tych ostatnich powinny więc mieć formę trybu rozkazującego.

Choć takich nazw (w trybie rozkazującym) jest ponad 50 (od CLEAN — OCZYŚĆ, do .SERIALOUT — .NADAJ), nie obejmują one wszystkich instrukcji stanowiących polecenia. Przy bliższej analizie łatwo przekonać się, że dość trudno byłoby nadać wszystkim instrukcjom tego rodzaju formę trybu rozkazującego. Jednakże, niektóre istniejące lub proponowane nazwy można skorygować, np. OGRODZENIE na OGRÓDŹ i ZAWIJANIE na ZAWIŃ. Nietrudno też stwierdzić, że prawie wszystkie nazwy instrukcji ustawiania parametrów, zaczynające się w wersji angielskiej od wyrazu SET, zarówno w pierwotnej wersji polskiego Logo jak i w propozycji zmodyfikowania, są dobrane niepoprawnie. Pozostała część instrukcji tej kategorii ma nazwy stanowiące, na ogół, rzeczownikowe skróty pełnych nazw poleceń (PUNKT, NEGATYW, POZYTYW, WYNIK, NADRUK, NUTA, OKNO, .ZAWARTOŚĆ, SZYBKOŚĆ. Te i inne nazwy w swej pełnej postaci powinny być poprzedzone czasownikiem w trybie rozkazującym.

Niewielka grupa instrukcji stanowiących polecenia ma nazwy dobrane, tak jak powinno być, w formie rozkazującej, lecz bez użycia czasowników a w postaci równoważników zdań — WSTECZ, NAPRZÓD, DOŚĆ, JUŻ, DOŚRODKA, WLEWO, WPRAWO, STOP, OTO. Ten sposób tworzenia nazw polskich wypada uznać za poprawny, gdyż sankcjonuje go praktyka tworzenia komend wojskowych (np. Baczność !, Biegiem marsz!), sportowych (np. Na miejsca!, Gotów!) i in.

Pozostałe instrukcje Logo udostępniają wartości. W grupie operacji arytmetycznych i logicznych należałoby utrzymać nazwy oryginalne, tak jak robi się to w matematyce. Niewielka grupa operacji badania parametrów odpowiadających operacjom ustawiania parametrów (ang. SET) została w polskim Logo opatrzona strzałką, co wydaje się dość trafne, gdyż różnicuje te instrukcje względem pozostałych. Podobnie wyróżniona, znakiem zapytania, jest grupa operacji udostępniających wartości logiczne ELSE i TRUE (WIDOCZNY?, PUSTE?, RÓWNE?, LISTA?, NALEŻY?, LICZBA?, SŁOWO?, MAWARTOŚĆ?, KLAWISZ?, ZDEFINIOWANE?, PIERWOTNE?). Ich nazwy mogą mieć bardzo różne formy językowe, dlatego że decydujące znaczenie nadaje im znak zapytania.

Jak widać, analiza polskich odpowiedników nazw angielskich nie jest pozbawiona sensu, gdyż analizując je dokładniej pod względem językowym można wykryć pewne nieprawidłowości i lepiej dobrać nazwy rodzime po to, aby ułatwić posługiwanie się językiem programowania. Dając się wciągnąć w dyskusję na temat polskiej terminologii Logo, nie mogę się jednak oprzeć wrażeniu, że ktoś chce spolszczyć Esperanto. Każdy język programowania pozostanie bowiem językiem sztucznym, niezależnie od tego, na czym go oparto.

JANUSZ ZALEWSKI

Shaw M. M.: Poza programowanie wielkoskalowe — kolejne wyzwanie dla inżynierii oprogramowania (1)

INFORMATYKA 1987, nr 7, s. 1

Charakterystyka zmian w inżynierii oprogramowania w wyniku wzrostu rozmiarów i złożoności systemów informatycznych.

Shaw M. M.: Beyond programming-in-the-large — next challenges for software engineering (1)

INFORMATYKA 1987, No. 7, p. 1

Characteristics of changes in software engineering as result of data processing system's size and complexity increase.

Shaw M. M.: Ausserhalb der Grossmassstabprogrammierung — nächste Herausforderung für die Software-Engineering (1)

INFORMATYKA 1987, Nr, 7, S. 1

Eine Charakteristik von Veränderungen in Software-Engineering infolge Steigerung der Grösse und Kompliziertheit der EDV-Systeme.

Tadeusiewicz R.: Głosowe wprowadzanie informacji do komputera (1)

INFORMATYKA 1987, nr 7, s. 5

Omówienie istoty oraz celów komputerowego rozpoznawania i wprowadzania informacji głosowych oraz charakterystyka krajowych prac badawczych w tej dziedzinie.

Tadeusiewicz R.: Voice information input for computer (1)

INFORMATYKA 1987, No. 7, p. 5

Presentation of essence and purposes of voice information input for computer, as well as characteristics of scientific research in Poland in this area.

Tadeusiewicz R.: Eingabe phonetischer Information in einem Computer (1)

INFORMATYKA 1987, Nr 7, S. 5

Eine Besprechung von Wesen und Ziele der Eingabe phonetischer Information in einem Computer und eine Charakteristik der in Polen geführten Forschung in diesem Bereich.

Madey J.: Problematyka systemów operacyjnych na przykładzie systemu Unix (2)

INFORMATYKA 1987, nr 7, s. 8

Druga część charakterystyki współczesnych rozwiązań systemów operacyjnych, zawierająca omówienie zagadnienia zarządzania zasobami systemu komputerowego.

Madey J.: Unix as an example of operating system (2)

INFORMATYKA 1987, No. 7, p. 8

Second part of contemporary operating systems characteristics, which includes discussion of computer system resources management.

Madey J.: Betriebssystemeproblematik an Beispiel des Unix-Systems (2)

INFORMATYKA 1987, Nr 7, S. 8

Zweiter Teil einer Charakteristik von Lösungen der heutigen Betriebssysteme, der eine Besprechung des Problems von Computerresourcenverwaltung umfasst.

Bielecki J.: Turbo Pascal

INFORMATYKA 1987, nr 7, s. 10

Omówienie różnic między językiem Turbo Pascal a językiem wzorcowym oraz prezentacja przykładu programu graficznego.

Bielecki J.: Turbo Pascal

INFORMATYKA 1987, No. 7, p. 10

Characteristics of Turbo Pascal language, which includes discussion of differences against standard language, as well as an example of graphic program.

Bielecki J.: Turbo Pascal

INFORMATYKA 1987, Nr 7, S. 10

Eine Charakteristik von Turbo Pascal Sprache, die eine Besprechung von Unterschieden zu Mustersprache, sowie ein Beispiel des grafischen Programms, umfasst.

Raznowiecki A., Syfert A., Zalewski J.: Struktura systemu operacyjnego PC-DOS (1)

INFORMATYKA 1987, nr 7, s. 17

Pierwsza część charakterystyki systemu operacyjnego PC-DOS na poziomie opisu dostosowanym do potrzeb programisty.

Raznowiecki A., Syfert A., Zalewski J.: Structure of PC-DOS operating system (1)

INFORMATYKA 1987, No. 7, p. 17

First part of PC-DOS operating system characteristics with regard to programmer's description level.

Raznowiecki A., Syfert A., Zalewski J.: Struktur des PC-DOS Betriebssystems (1)

INFORMATYKA 1987, Nr 7, S. 17

Erster Teil einer Charakteristik des PC-DOS-Betriebssystems mit einen an Programmiererbedürfnisse angepassten Beschreibungsniveau.

Terminologia języka Logo (3)

W przedstawionej niżej tabeli zebrano angielskie nazwy instrukcji Logo oraz ich odpowiedniki w PTI Logo. Zaproponowano także zmiany niektórych nazw, zgodnie z argumentacją zawartą w dwóch poprzednich częściach tego artykułu.

Proponowane zmiany nazw w polskim Logo

Nazwa angielska	Nazwa polska	Proponowana zmiana
AND	I	AND
ARCCOS	ARCCOS	—
ARCCOT	ARCCTG	—
ARCSIN	ARCSIN	—
ARCTAN	ARCTG	—
ASCII	ASCII	KOD
BACK	WSTECZ	—
BACKGROUND	TŁO↑	—
BRIGHT	JASKRAWO	—
BUTFIRST	BEZPIERW	—
BUTLAST	BEZOST	—
BYE	DOŚĆ	—
CHAR	ZNAK	—
CLEAN	ZMAŻ	OCZYŚĆ
CLEARSCREEN	CZYŚĆ	ZMAŻ
CLEARTEXT	ZMAŻTEKST	—
COPYDEF	POWIEL	—
COPYSCREEN	DRUKOBRAZU	DRUKUJOBRAZ
COS	COS	—
COT	CTG	—
COUNT	DŁUGOŚĆ	—
CURSOR	KURSOR↑	—
DEFINE	OKREŚL	ZDEFINIUJ
DEFINEDP	OKREŚL?	ZDEFINIOWANE?
DIV	ILORAZ	—
DOT	PKT	—
EDIT	RED	—
EDNS	REDWN	REDNAZWY
EMPTYP	PUSTE?	—
END	JUŻ	—
EQUALP	RÓWNE?	—
ERASE	USUŃ	—
ERALL	USW	USUŃWSZ
ERN	USN	USUŃNAZWĘ
ERNS	USWN	USUŃNAZWY
ERPS	USWP	USUŃPROCEDURY
FALSE	FAŁSZ	FALSE
FENCE	POLE	OGRODZENIE
FIRST	PIERW	PIERWSZY
FLASH	MIGAJ	—
FORWARD	NAPRZÓD	—
FPUT	NAP	NAPOCZĄTEK
HEADING	KĄT↑	—
HIDETURLE	SŻ	ZASŁOŃ
HOME	WRÓĆ	DOŚRODKA
IF	JEŚLI	— (IF)
INT	ENTIER	—
INVERSE	NEGATYW	—
ITEM	ELEMENT	—
KEYP	KLAWISZ?	—
LAST	OST	OSTATNI
LEFT	LEWO	WLEWO
LIST	LISTA	—
LISTP	LISTA?	—
LOAD	ŁADUJ	—
LOADD	ŁADUJR	ŁADUJRED
LOADSCR	ŁADUJO	ŁADUJOBR
LPUT	NAK	NAKONIEC
MAKE	PRZYPISZ	—
MEMBERP	ELEMENT?	NALEŻY?
NAMEP	JEST	MAWARTOŚĆ?
NODES	WOLNE	—
NORMAL	POZYTYW	—
NOT	NIE	NOT
NUMBERP	LICZBA?	—
OR	LUB	OR
OUTPUT	WYNIK	—
OVER	NADRUK	—
PENCOLOUR	PISAK↑	PIÓRO↑
PENDOWN	OPU	OPUŚĆ
PENERASE	ŚCIERANIE	ŚCIERAJ
PENREVERSE	ODWRACANIE	ODWRÓĆ
PENUP	POD	PODNIEŚ
PO	PO	WYDRUKUJ
POALL	POW	WYDRUKWSZ
PONS	POWN	WYDRUKNAZWY
POPS	POWP	WYDRUKPROCEDURY
POSITION	POZ↑	—
POTS	POTP	WYDRUKTYTUŁY
PRIMITIVEP	PIERWOTNE?	—
PRINT	PISZ	DRUKUJ (PRINT)
PRINTOFF	WYŁĄCZDRUK	—
PRINTON	WŁĄCZDRUK	—
PRODUCT	ILOCZYN	—
RANDOM	LOSOWA	RANDOM
READCHAR	CZYTAJZNAK	—
READLIST	CZYTAJLISTĘ	—
RECYCLE	ODŚMIEĆ	ZWOLNIJ
REMAINDER	RESZTA	—
REPEAT	POWTÓRZ	— (REPEAT)
RIGHT	PRAWO	WPRAWO
ROUND	ZAOKR	ROUND
RUN	ZRÓB	WYKONAJ
SAVE	ZAPISZP	ZAPISZ
SAVEALL	ZAPISZ	ZAPISZWSZ
SAVED	ZAPISZR	ZAPISZRED
SAVESCR	ZAPISZO	ZAPISZOBR
SCRUNCH	PROPORCJA↑	SPŁASZCZENIE↑
SENTENCE	ZDANIE	—
SETBG	TŁO	KOLTŁA
SETBORDER	RAMKA	KOLRAMKI
SETCURSOR	KURSOR	RUSZKURSOR
SETHEADING	KĄT	KURS
STEPC	PISAK	KOLPIÓRKA
SETPOS	POZ	POZXY
SETSCR	PROPORCJA	SPŁASZCZ
SETTC	KOLORYT	KOLTEKSTU
SETX	XPOZ	POZX
SETY	YPOZ	POZY
SHOW	POKAŻ	—
SHOWNP	WIDAĆ?	WIDOCZNY?
SHOWTURTLE	PŻ	ODSŁOŃ
SIN	SIN	—
SOUND	NUTA	—
SQRT	PIERWIASTEK	SQRT
STOP	STOP	—
SUM	SUMA	—
TAN	TG	—
TEXT	TREŚĆ	—
TEXTCOLOUR	KOLORYT↑	TEKST↑
TEXTSCREEN	TEKSTY	TEKSTOWO
THING	WARTOŚĆ	—
TO	OTO	—
TOPLEVEL	PRZERWIJ	WRÓĆ
TOWARDS	AZYMUT	KIERUNEK
TRUE	PRAWDA	TRUE
TYPE	WPISZ	WYŚWIETL (TYPE)
WAIT	CZEKAJ	—
WINDOW	OKNO	—
WORD	SŁOWO	—
WORDP	SŁOWO?	—
WRAP	SKLEJ	ZAWIJANIE
XCOR	XPOZ↑	—
YCOR	YPOZ↑	—
.BLOAD	.ŁADUJBIN	—
.BSAVE	.ZAPISZBIN	—
.CALL	.WYWOŁAJ	—
.CONTENTS	.PAMIĘĆ	.ZAWARTOŚĆ
.DEPOSIT	.UMIEŚĆ	—
.EXAMINE	.ZOBACZ	.SPRAWDŹ
.PRIMITIVES	.PIERWOTNE	—
.RESERVE	.ZAJMIJ	—
.RESERVED	.ZAJĘTE	—
.SERIALIN	.PRZYJMIJ	.ODBIERZ
.SERIALOUT	.WYŚLIJ	.NADAJ
.SETSERIAL	.TRANSMISJA	.SZYBKOŚĆ

dokończenie na str. 31

KONIEC TWOICH PROBLEMÓW

Nareszcie wszystkie potrzebne Ci dane dotrą na czas
w przejrzystej formie tabel i wykresów.

Pakiet oprogramowania na mikrokomputery 16-bitowe

MEGA — BANK

to nowe MEGA możliwości jakie otwierają się przed
Twoim przedsiębiorstwem.

Wyobraź sobie

MILIARD

rekordów, które możesz
zapełnić według własnych potrzeb i uznania.

Miliard kooperantów, miliard pracowników, miliard produktów
— z tym wszystkim nasz MEGA-BANK poradzi sobie bez trudu.

System jest łatwy w obsłudze i opracowany w języku polskim.

Gwarantujemy satysfakcję!

COMPUTER STUDIO KAJKOWSCY
PROFESJONALNE OPROGRAMOWANIE MIKROKOMPUTERÓW
ul. Balladyny 3B, 81-524 Gdynia, tel.: 29-00-18, 24-01-50

BO/74/87