P.1877/87

9 1987

informatyka

Prof. Gerhard Goos o produkcji oprogramowania

Multicomp – rozwiązywanie zadań

Wordstar 3.30

WYDAWNICTWO NOT SIGMA

Cena 150 zł

informatyka 9 1987

Nr 9

Miesięcznik Rok XXII

Wrzesień 1987

Organ Komitetu Naukowo-Technicznego NOT ds. Informatyki

KOLEGIUM REDAKCYJNE:

Mgr Jarosław DEMINET, dr inż. Wacław ISZKOWSKI, mgr Teresa JABŁOŃSKA (sekretarz redakcji), Władysław KLEPACZ (redaktor naczelny), dr inż. Marek MACHURA, Maria PAWLAK (sekretarz redakcji), dr inż. Wiktor RZECZKOWSKI, mgr inż. Jan RYŻKO, mgr Hanna WŁODARSKA, dr inż. Janusz ZALEWSKI (zastępca redaktora naczelnego)

PRZEWODNICZĄCY RADY PROGRAMOWEJ:

Prof. dr hab. Juliusz Lech Kulikowski

Materiałów nie zamówionych redakcja nie zwraca

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

Zakł. Graf. „Tamka". Zam. 0575-1300/87. Obj. 4,0 ark. druk. Nakład 8200 egz. K-82.

ISSN 0542-9951, INDEKS 36124

Cena egzemplarza 150 zł
Prenumerata roczna 1800 zł

00-950 Warszawa
skrytka pocztowa 1004
ul. Biała 4

W NUMERZE: Strona

Nowoczesne języki wysokiego poziomu a produkcja oprogramowania
Gerhard Goos 1

MULTICOMP — system rozwiązywania zadań metodą przeszukiwania drzew
Piotr Zielczyński 5

Wordstar 3.30 — zasady działania i sposób użytkowania (1)
Stanisława Ossowska 9

Zastosowanie Prologu w bazach danych
Włodzimierz Grudziński 11

Podstawy grafiki w języku Turbo Pascal (2). Wykreślanie podstawowych obiektów graficznych
Jan Bielecki 14

Język C — wykreślanie podstawowych obiektów graficznych
Jan Bielecki 18

Wprowadzenie do programowania w języku assemblera IBM PC (1)
Oprac. *D. Grabowicz* 20

ZE ŚWIATA 20

FASTBUS — modularny system magistralowy

Ada — konwencje kodowania

Kto jest kim w IFIP. Graham Morris

TERMINOLOGIA 28

Metakody znaków alfanumerycznych

W NAJBLIŻSZYCH NUMERACH:

- Cliff B. Jones o specyfikacjach i programach
- Witold Staniszkis o zarządzaniu rozproszonymi danymi
- Jan Bielecki o programowaniu operacji wejścia-wyjścia w języku C
- Mariusz Kuc podaje przykład wielozadaniowości w Adzie
- Jarosław Deminet omawia II Walny Zjazd PTI

GERHARD GOOS
Gesellschaft für Mathematik
und Datenverarbeitung mbH
Forschungsstelle Karlsruhe
Republika Federalna Niemiec

P. 1877/87

Nowoczesne języki wysokiego poziomu a produkcja oprogramowania

Celem artykułu[1]) jest omówienie minionych, obecnych i ewentualnych przyszłych prób zwiększenia produktywności programistów i niezawodności oprogramowania przy użyciu współczesnych języków wysokiego poziomu, jak Modula-2, Ada lub Chill.

Z ekonomicznego, jak i z technicznego punktu widzenia najbardziej palące problemy występujące w produkcji oprogramowania można scharakteryzować następująco:

- produkcja oprogramowania odbywa się zbyt wolno i nie nadąża za wzrastającym zapotrzebowaniem,
- oprogramowanie nie jest dostatecznie niezawodne,
- koszty pielęgnacji i modyfikacji oprogramowania są zbyt duże.

Już w 1981 roku, B. W. Boehm analizował wykładniczy wzrost zapotrzebowania na oprogramowanie w ciągu ostatnich 25 lat, biorąc pod uwagę, na przykład, wymagania programowe stawiane w programie kosmicznym realizowanym przez Amerykańską Agencję Aeronautyki i Przestrzeni Kosmicznej (Mercury, Gemini, Apollo, prom kosmiczny). Z drugiej strony, produktywność programistów, którzy mieli spełnić te wymagania programowe, zwiększyła się w ciągu ostatnich 30 lat najwyżej dziesięciokrotnie. Większą część wzrostu produktywności należy przypisać wprowadzeniu języków wysokiego poziomu, lecz większość wykładniczego wzrostu zapotrzebowania na oprogramowanie można zaspokoić tylko zatrudniając dodatkowych programistów. Według szacunków japońskiego Ministerstwa Handlu i Przemysłu (MITI) w roku 1985 brakowało około 600 000 programistów. Ekstrapolacja krzywej Boehma prowadzi do wniosku, że około roku 2000, tj. za piętnaście lat, wszyscy Amerykanie i prawdopodobnie wszyscy Niemcy będą musieli zostać programistami, aby zaspokoić potrzeby produkcji oprogramowania. Choć to przewidywanie jest z pewnością tak samo błędne, jak przyjmowane na początku wieku przypuszczenie, że wszyscy mieszkańcy Ziemi staną się operatorami central telefonicznych, to aktualny pozostaje problem, co powinni uczynić programiści, aby zwiększyć produktywność w procesie wytwarzania oprogramowania.

Podobne problemy występują, gdy bierze się pod uwagę jakość obecnie wytwarzanych systemów oprogramowania. Błędy programowe popełnia się bardzo często, a 3 defekty na 1000 instrukcji świadczą o dobrej jakości produktu. Defekty powstają tak w fazach specyfikowania, jak i implementowania programu, pomimo angażowania ogromnych środków w przeciwdziałanie powstawaniu defektów i możliwie najwcześniejsze ich wykrywanie. Natomiast podana miara jakości uwzględnia tylko defekty będące bezpośrednią przyczyną błędnego funkcjonowania systemu. Konieczne jest jednak także uwzględnienie niezadowalającego działania spowodowanego wyborem nieodpowiedniej struktury systemu lub złego algorytmu czy struktury danych. W większości z tych miar bierze się również pod uwagę właściwości sprzężenia użytkowego (ang. human interface) i stopień uwzględnienia wyjątkowych warunków zewnętrznych.

Niedoskonałość struktury systemu i niski stopień niezawodności stanowią główne przyczyny dużych kosztów jego pielęgnacji. Wcale nierzadkie są przypadki długożyciowych systemów oprogramowania, których koszt pielęgnacji stanowi 70—90% wszystkich kosztów ponoszonych w ich okresie istnienia. Okazuje się, że większą część tego kosztu można przypisać decyzjom projektowym, które później uznano za błędne. Systematyczna wiedza służąca zidentyfikowaniu takich błędnych decyzji projektowych jest jednak trudna do zgromadzenia i wykorzystania.

Poniżej omówiono główne czynniki techniczne, które miały wpływ lub mogą mieć wpływ w przyszłości na rozwiązanie wymienionych problemów.

HISTORIA

Najbardziej spektakularnym osiągnięciem prowadzącym do zwiększenia produktywności i niezawodności oraz do zmniejszenia kosztów pielęgnacji było wprowadzenie języków wysokiego poziomu. W latach sześćdziesiątych, a nawet siedemdziesiątych, toczono spory, mające posmak wojen religijnych, o najlepszy język programowania i jego pożądane właściwości. Takie dyskusje były i częściowo nadal są pożądane z następujących przyczyn:

- języki programowania, podobnie jak języki naturalne, programistów i jako takie muszą być udoskonalane;
- językip rogramowania, podobnie jak języki naturalne, kształtują sposób myślenia tych, którzy ich używają. To spostrzeżenie odnosi się nie tylko do fazy implementacji, lecz także do fazy projektowania i rozważań dotyczących możliwości technicznych;
- jakość języka programowania determinuje możliwą do osiągnięcia jakość programów. Dlatego też ma wpływ na to, co można uzyskać w ograniczonym czasie;
- języki programowania są jak dotychczas najważniejszym pojedynczym czynnikiem mającym wpływ na produktywność programistów.

Pomimo tak oczywistych argumentów, w latach siedemdziesiątych stało się jasne, że dyskusja o językach programowania dotyczy w rzeczywistości czegoś zupełnie innego, mianowicie określonej metodologii, ponieważ każdy język wspiera pewną metodologię lub ich klasę. Choć danego języka można używać także w innych metodologiach, jego wykorzystanie nie jest optymalne. Zamiast więc mówić o językach programowania, lepiej jest skupić uwagę na metodologiach projektowania i implementacji oprogramowania. Co więcej, obecnie wiadomo, że zaprojektowanie języka programowania, łącznie z opracowaniem kompilatorów i odpowiedniego środowiska programowego oraz wyszkoleniem tysięcy programistów i kadr innych specjalistów trwa 10—15 lat. Ten okres jest prawdopodobnie bardzo bliski długości zatrudnienia nauczycieli akademickich na stanowiskach wymagających aktywności i wiedzy technicznej (w przeciwieństwie do stanowisk administracyjnych itp.). Jest też zbliżony do czasu wymaganego w innych technologiach na wprowadzenie nowej idei do praktyki. Wskutek tego, nawet takie języki, jak Modula-2, Chill lub Ada, mające dopiero po 6—8 lat i wprowadzane do praktyki tylko częściowo, są już przestarzałe. Nie odzwierciedlają już one najnowszych osiągnięć w dziedzinie metodologii programo-

[1]) Tekst tego artykułu jest — dokonanym za zgodą Autora — tłumaczeniem referatu wprowadzającego (ang. keynote address), wygłoszonego na IV Konferencji nt. języka Chill (Fourth Chill Conference), która odbyła się w Monachium, w dniach 29 września — 2 października 1986 r.

wania. Jednakże najważniejszym wnioskiem z tej dyskusji jest stwierdzenie, że praktycznie nigdy nie będziemy mieli języka programowania reprezentującego najnowszy stan wiedzy w dziedzinie metodologii programowania. Dlatego należy rozważyć problem, jak wspomagać nowsze metodologie stosując pęzyki, które nie były dla tych metodologii przeznaczone.

Historycznie rzecz biorąc, większość konstrukcji w językach, które opracowano do 1975 roku, dotyczyła właściwego formułowania instrukcji do wyrażania algorytmów i struktur danych na poziomie proceduralnym. Dla takiego poglądu na świat programowania ukuto termin **programowanie małoskalowe** (ang. programming-in-the-small). Programowanie strukturalne i weryfikacja algorytmów podczas projektowania stanowią najbardziej skuteczne metodologie na tym poziomie. Programowanie małoskalowe odgrywa istotną rolę w nauczaniu programowania i w obliczeniach na komputerach osobistych. Można powiedzieć, że stanowi podstawową warstwę wiedzy programistycznej, którą powinien opanować każdy informatyk. Sytuację w tej dziedzinie charakteryzują najlepiej języki takie jak Pascal.

Dla dużych systemów należy rozwiązać problem właściwej współpracy wielu algorytmów i struktur danych. Poszczególne fragmenty systemu mogą mieć różnych autorów, mogą pochodzić z bibliotek i mogą nie być dostosowane w szczegółach do przewidywanych zastosowań. W takiej sytuacji dokonuje się statystycznie tak wielu kontroli niesprzeczności jak tylko możliwe, przeprowadzając je w czasie kompilacji, aby zredukować ich liczbę. Dla metodologii i rozwiązań w tej klasie zagadnień ukuto nazwę **programowanie wielkoskalowe** (ang. programming-in-the-large). Teoretycy wprowadzili pojęcie abstrakcyjnego typu danych, jako funkcjonalnego opisu sprzężenia struktury danych i skojarzonych z nią operacji. Ten opis sprzężenia wyraża nie tylko syntaktyczne, lecz także semantyczne właściwości struktury danych. Z kolei praktycy wprowadzili pojęcie modułu, który jest — w różnym stopniu — realizacją procedury, abstrakcyjnego typu danych lub zadania. Z przyczyn technicznych kontrola sprzężeń jest ograniczona do zagadnień zgodności syntaktycznej i zgodności typów. Stan wiedzy w dziedzinie programowania wielkoskalowego reprezentują takie języki jak Modula-2, Ada i Chill.

Jednak żaden z tych języków nie jest zadowalający z technicznego punktu widzenia. W Moduli-2 nie uwzględniono wielu istotnych zagadnień, m.in. obsługi wyjątków, a pozostałe dwa języki są dość skomplikowane. Skonstruowano je z myślą, aby zaspokoiły potrzeby swych użytkowników, lecz mniej uwagi zwrócono na doskonałość techniczną.

Ponadto, postępy poczynione w dziedzinie metodologii programowania w tych językach (obecnie już sześcio—ośmioletnich) nie są odzwierciedlone w odpowiednich właściwościach tych języków. Największy postęp dotyczy poglądów na wieloużywalność modułów (ang. reusability of modules). Już teraz można by osiągnąć znaczne zmniejszenie ilości oprogramowania koniecznego do napisania, gdyby zamiast wytwarzać każdy system oprogramowania od początku, można brać większą liczbę modułów z bibliotek lub korzystać z wyników wcześniejszych przedsięwzięć programistycznych. Jednakże, jak wskazuje doświadczenie praktyczne, przy projektowaniu modułów do wielokrotnego wykorzystania należy wziąć pod uwagę wiele specyficznych czynników. Ich uwzględnienie jest konieczne, nawet gdyby miało spowolnić proces rzeczywistego projektowania. Wielourzywalność modułu jest znaczna wtedy, gdy jego sprzężenia są dostosowane do dużej liczby zastosowań. Programiści rzadko zgadzają się między sobą, jakie właściwości powinny mieć określone sprzężenia. Dlatego, aby wytwarzać moduły wieloużywalne i wprowadzać je do użytku, należy mieć duże doświadczenie i dobry przegląd całej klasy modułów pokrewnych, a kierownicy zespołów powinni narzucić żelazną dyscyplinę swym programistom. Byłoby korzystne, gdyby klasa modułów o standardowych sprzężeniach była ciągle rozszerzana, a zasady sprzęgania były wykładane na uczelniach itp. Nawet jednak, jeśli tak się stanie, to upłyną dziesięciolecia, zanim tego rodzaju nauczanie przyniesie praktyczne i widoczne skutki w przemyśle.

Okazuje się również, że języki takie jak Modula-2, Ada i Chill, pomimo rodzajowości zawartej w Adzie, w wielu wypadkach nie umożliwiają wyrażenia odpowiedniego poziomu abstrakcji do projektowania modułów wieloużywalnych. Wielokrotne wykorzystanie modułu polega najczęściej na wielokrotnym wykorzystaniu projektu, który należy sparametryzować wieloma właściwościami, zanim otrzyma się wykonywalny segment kodu. Współczesne języki programowania nie mają jeszcze wystarczających konstrukcji do wyrażania takiej parametryzacji.

Te rozważania prowadzą do wniosku, że przyszłe języki programowania będą miały wiele cech, które obecnie kojarzymy z językami specyfikacji. Końcowy, wykonywalny program będzie wynikiem transformacji zastosowanych do specyfikacji i jej parametrów. Wynika stąd, że przyszłych języków programowania nie będzie można rozpatrywać w izolacji. Istotną rolę odegra system wspomagania złożony z podsystemu transformacji programów, specyfikacji itp. Język bez swego środowiska stanie się bezwartościowy. Jakość języka będzie ostatecznie określona przez jakość jego środowiska programowego. Dlatego niezwykle istotne jest, aby już teraz zadbać o ulepszanie środowisk języków programowania.

CELE PRODUKCJI OPROGRAMOWANIA

Wiele przyszłych problemów w dziedzinie języków programowania i ich środowisk wyniknie z rozszerzenia celów produkcji oprogramowania.

Historycznie rzecz biorąc, programy rozumiano jako opisy transformacji wartości wejściowych na wartości wyjściowe. Program jest realizacją funkcji matematycznej:

F: We → Wy

bez żadnych efektów ubocznych dla środowiska, tak że wielokrotne wykonania tego programu wykazują jego identyczne zachowanie.

Jeżeli wszystkie wyniki niejawne, np. aktualizacja bazy danych, potraktuje się jako część wyników oficjalnych określonego segmentu, to podana charakterystyka programu będzie nadal ważna, przynajmniej na poziomie procedur.

Jednakże, wiele współczesnych systemów programowych wykazuje odmienne zachowanie. Nie zakłada się, że wytwarzają one ustalony zbiór wartości wyjściowych w odpowiedzi na określone wartości wejściowe. Wymaga się natomiast, aby działały nieskończenie długo i sterowały pewnymi działaniami zachodzącymi na zewnątrz, tj. jednym lub wieloma „procesami zewnętrznymi". Ada i Chill zostały zaprojektowane do takich właśnie zastosowań w systemach wbudowanych. Ta klasa zastosowań pod wieloma względami powoduje zmianę poglądów na właściwości oprogramowania:

- zakończenie wykonywania programu jest obecnie uważane za awarię, natomiast program interpretowany jako funkcja matematyczna musi się zakończyć;
- w konsekwencji poprawności oprogramowania nie można określić metodą sprawdzania poprawności wartości wyjściowych po zakończeniu programu;
- dominującą rolę zaczynają natomiast odgrywać zagadnienia niezawodności, a nawet bezpieczeństwa. Oprogramowanie powinno zawsze wykazywać akceptowalne działanie, nawet wtedy, gdy pewna część systemu lub używany sprzęt ulegnie awarii albo gdy zostaną dostarczone niedozwolone lub sprzeczne dane. W żadnym wypadku, za wyjątkiem zaniku zasilania lub innych poważnych zdarzeń, oprogramowanie nie powinno dopuścić do gwałtownego zaprzestania działania systemu, lecz umożliwić mu stopniową utratę funkcjonalności (ang. graceful degradation of performance);
- system ma działać nieskończenie długo, powinien więc mieć możliwość dynamicznej rekonfiguracji, aby umożliwiać poprawianie potencjalnych defektów i adaptację do zmiennych wymagań lub zmodyfikowanych systemów macierzystych (ang. host systems).

Pomimo że wymagania te są znane już przynajmniej od dziesięciu lat, współczesne języki i ich implementacje nie zapewniają zadowalającego rozwiązania tych zagadnień, szczególnie jeśli chodzi o dynamiczną rekonfigurację programów.

Przyszłość pokaże, że przedstawione poglądy z pewnością nie są jedynymi, jakie można mieć na temat oprogramowania. Następny etap ewolucji można określić mianem „oprogramowanie jako składowa systemów niejednorodnych". Przez systemy niejednorodne rozumie się systemy

do rozwiązywania problemów w sposób rozłożony. Część systemu mogą stanowić komputery, z oprogramowaniem włącznie, inną częścią mogą być urządzenia techniczne, a nawet ludzie. Większość systemów współpracy człowiek-maszyna już wykazuje lub powinna wykazywać zachowanie typowe dla tej klasy. Rozwiązanie problemu nie jest już określone wyłącznie przez oprogramowanie. Jego część jest przekazywana ludziom lub urządzeniom technicznym, a rozłożenie zadań może się zmieniać stosownie do możliwości jednostek uczestniczących. Takie systemy mają interesujące właściwości;

- nie można już dokładnie określić części rozwiązania problemu wykonywanej przez oprogramowanie. System oprogramowania jest niezawodny, jeśli znajduje poprawne rozwiązanie w tych wypadkach, w których potrafi je określić, i jeśli w pozostałych wypadkach poprawnie wskazuje tę część systemu, której można przekazać problem do rozwiązania;
- oprogramowanie musi być zdolne do dostarczenia dodatkowych danych jednostce rozwiązującej problem, opartych na analizie wartości wejściowych;
- istnieje kilka możliwych wariantów przyjęcia odpowiedzialności za poprawność wyników takiego systemu. Przykładowo, oprogramowanie nie powinno mieć możliwości podejmowania decyzji o życiu lub śmierci ani w zastosowaniach medycznych lub wojskowych, ani w systemach komunikacji, nawet jeśli system oprogramowania wykonuje większą część pracy.

Powyższe rozważania mogą posłużyć jako wskazówka do właściwego spojrzenia na współpracę człowiek-maszyna w środowiskach programowych. Większość narzędzi w środowisku programowym można podzielić na dwie grupy: narzędzia dostarczające rozwiązania problemu (kompilatory, konsolidatory), wywoływane przez programistę z terminala, i narzędzia wspomagające proces wykonywany w rzeczywistości przez użytkownika (najlepszym przykładem z tej drugiej grupy są edytory). Obecnie nie ma jeszcze dobrych przykładów narzędzi, których działanie polegałoby na przełączaniu się między użytkownikiem a komputerem, zależnie od możliwości obu stron.

STAN OBECNY I PERSPEKTYWY

Wypada teraz podsumować, jaki jest stan wiedzy w dziedzinie języków programowania i ich środowisk, i jakie problemy są wciąż nie rozwiązane.

Do końca tego wieku będziemy korzystali z takich języków jak Modula-2, Chill i Ada. Ze względu na długie opóźnienie między wprowadzeniem a praktycznym wykorzystaniem języka, jest bardzo mało prawdopodobne, aby w ciągu najbliższych 15 lat pojawił się w praktyce jakiś nowy język. Wynikają stąd następujące wnioski:

- należy opracować długofalowy plan wsparcia trzech wymienionych języków odpowiednimi narzędziami. Poniesione nakłady będą ekonomicznie uzasadnione w wypadku dłuższego ich użycia;
- każdy postęp w metodologii, który można wykorzystać, należy uwzględnić w przyszłych wersjach języków lub w nowo wytworzonych preprocesorach czy innych narzędziach, które pozwolą zwiększyć poziom abstrakcji służących do wyrażania projektów.

Ponadto należy wyciągnąć wnioski z przeszłości, że języki takie jak Fortran lub Cobol przetrwały ze względu na ogromną ilość oprogramowania, jaką w nich napisano, mimo iż od wielu lat są technicznie przestarzałe. Dlatego jedno z podstawowych pytań związanych z użyciem języków programowania dotyczy możliwości wcześniejszego przygotowania konwersji oprogramowania z istniejących języków na nowe, jeszcze nie wprowadzone. Zwiększenie poziomu abstrakcji opisu projektów jest jednym ze sposobów osiągnięcia tego celu. Jeżeli nie będzie się myśleć o tym problemie już teraz, to nasze dzieci i wnuki zostaną ukarane koniecznością stosowania tak przestarzałych języków, jakimi Modula-2, Ada i Chill będą za 30 lub 50 lat.

Inwestujący w narzędzia w przemyśle wiedzą już, że nakłady te nie przynoszą efektów bez końca. Wydaje się, że projektanci oprogramowania jeszcze nie przeszli tej lekcji. Nie są oni jeszcze przygotowani do sytuacji, gdy ich obecne narzędzia staną się przestarzałe.

Inne ważne zagadnienie wiąże się z redukcją złożoności i modelowaniem systemów. Systemy sprzętowe jak również systemy oprogramowania rozrastają się w swoim okresie istnienia. Ten fakt powinni przewidywać projektanci. Dobry projekt odróżnia się od złego przez porównanie, jak łatwa (lub trudna) jest jego modyfikacja. Doświadczenie uczy, że im prostszy i bardziej usystematyzowany jest model systemu, tym łatwiejsze będzie jego późniejsze modyfikowanie i rozszerzanie. Tak więc projektanci muszą znacznie uzupełnić swoją wiedzę w zakresie tworzenia systematycznych modeli systemów, np. systemów równoległych i komunikacji międzyprocesowej. Powinni także nauczyć się odróżniać wymagania istotne od mniej ważnych, w celu zmniejszenia złożoności swoich projektów. Przykładowo, wielu programistów uważa, że systemy operacyjne i im podobne oprogramowanie należy pisać w językach asemblerowych, ponieważ kompilatory języków wysokiego poziomu nie umożliwiają użycia rozkazów uprzywilejowanych. Z drugiej strony wiadomo, że co najwyżej 5% kodu systemu operacyjnego korzysta z tej właściwości. Dlatego właściwe byłoby postępowanie polegające na wyodrębnieniu odpowiednich części podczas projektowania, w celu potraktowania ich w czasie implementacji odmiennie od reszty systemu.

Integracja systemu i standardy oprogramowania (tzn. sprzężenia standardowe) stanowią kolejne ważne zagadnienie w tej dziedzinie. Szczególnie kierownicy projektów często wyrażają pogląd, że do utworzenia środowiska programowego lub innego systemu wystarczy połączyć razem posiadane narzędzia. Wskutek tego system jest zbyt wolny, ze względu na konieczność dopasowania niezgodnych sprzężeń. Programista musi nauczyć się używania pięciu różnych edytorów i dziesięciu różnych języków poleceń. Dlatego jego produktywność zbliża się do zera, ponieważ jest bardziej zajęty opanowaniem narzędzi niż rozwiązywaniem postawionego zadania. Co więcej, popełnia on ciągle błędy, a niezawodność jego programów maleje.

Środkiem zaradczym na ten stan jest posiadanie długofalowego planu, do którego będą dostosowywane narzędzia powstające w przyszłości. Ten plan musi w szczególności zalecać jednolite sprzężenia użytkowe. Co więcej, ustanowienie faktycznych standardów (ang. de facto standard) na części sprzęgające systemów jest już potrzebą chwili. Techniczna jakość takich standardów jest często nie tak ważna jak sam fakt, że one istnieją.

Stan wiedzy w dziedzinie środowisk programowych jest określony przez względną znajomość potrzebnych narzędzi, takich jak: edytory, kompilatory, konsolidatory, środki uruchomieniowe, biblioteki itp. Jednakże, po pierwsze, narzędzia te nie są dostatecznie zintegrowane, a po drugie, brak jest doświadczeń dotyczących optymalnego wykorzystania stanowisk roboczych (ang. workstation). Przykładowo, już teraz system oparty na mikroprocesorze 68020 o częstotliwości zegara 25 MHz, a z pewnością następna generacja procesorów przeznaczonych dla stanowisk roboczych, umożliwią skompilowanie modułu, o wielkości 500 linii w ciągu jednej sekundy. Wynika stąd, że kompilowanie można traktować jak część interakcyjnego zadania redagowania, a całym procesem — sterować za pomocą narzędzia do zarządzania konfiguracją, zapewniającego kontekst do redagowania, kompilowania i łączenia modułów.

Co więcej, nawet gdy mówi się o językach specyfikacyjnych, prototypizacji (ang. prototyping) opartej na specyfikacjach projektowych itp., nie znaczy to, że istnieje funkcjonujący model wspomagania specyfikacji i prototypizacji w sposób zintegrowany z obecnymi środowiskami. Z perspektywy zarządzania oznacza to, że nadal obowiązuje pewien rodzaj kaskadowego modelu okresu istnienia oprogramowania (ang. software life cycle), choć taki model staje się przestarzały, gdy stosuje się prototypizację lub narzędzia transformacyjne do uczynienia specyfikacji wykonywalną. Mówiąc technicznie, niezintegrowanie narzędzi specyfikacyjnych staje się widoczne, gdy dochodzi do testowania. Integracja oznaczałaby, że istnieją proste środki do zgromadzenia w bazie danych wszystkich testów rozważanych w fazie specyfikacji i do wykonania ich dla końcowej implementacji. Jest oczywiste, że takie rozwiązanie, choć łatwe do opisania, nie istnieje w praktyce, ponieważ stanowiłoby wtedy podstawę schematu kontroli jakości (ang. quality assurance). Nie należy się zatem dziwić, że obecnie kontrola jakości jest jeszcze procesem wykonywanym przede wszystkim ręcznie, jedynie z niewielką pomocą narzędzi automatycznych.

O użyciu systemów baz danych w środowiskach programowych można by powiedzieć bardzo wiele. Przejście od obecnie stosowanych systemów plików do właściwych systemów baz danych, prawdopodobnie opartych na modelu relacyjnym lub modelu E-R (ang. entity relationship), powinno nastąpić w ciągu najbliższych 3—5 lat. Jest to jednak temat wart oddzielnego omówienia.

* * *

Kończąc, warto przytoczyć kilka myśli, które wyraził na Kongresie IFIP '86 w Dublinie Fred Brooks, projektant systemu operacyjnego OS/360. Rozważał on wszystkie współczesne koncepcje udoskonalenia procesu konstruowania oprogramowania, prowadzące nie do wzrostu częściowego, lecz wielokrotnego.

Oprócz stosowania języków wysokiego poziomu, uwzględnił on trzy zasadnicze zalecenia:

- kupowanie oprogramowania, a nie samodzielne pisanie go (autor artykułu interpretuje to jako żądanie wielokrotnego używania modułów zamiast ponownego pisania ich),
- stosowanie szybkiej prototypizacji,
- stopniowe rozszerzanie systemu polegające na kolejnym dodawaniu nowych funkcji do już działających, a nie podejmowanie olbrzymich przedsięwzięć mających małe szanse na zrealizowanie.

W zakończeniu wymienionego referatu autor dochodzi do wniosku, który warto powtórzyć, że kluczowym zagadnieniem jest znalezienie i wykształcenie spośród dorastającego pokolenia projektantów zdolnych do tworzenia koncepcji. Wytwarzanie oprogramowania jest i pozostanie sferą działalności ludzkiej, dlatego najlepszą gwarancją powstawania dobrych systemów są dobrze przygotowani ludzie.

Tłum. i oprac.
JANUSZ ZALEWSKI

G. Goos, G. Persch, J. Uhl: Programmiermethodik mit Ada. Springer-Verlag, Berlin, 1987

Z przedmowy autorów:

Język programowania Ada powstał w wyniku próby podsumowania stanu wiedzy w zakresie technik programowania pod koniec lat siedemdziesiątych oraz w odpowiedzi na potrzebę zaprojektowania wyważonego języka spełniającego wymagania praktyki. Opanowanie takiego języka nie powinno ograniczać się jedynie do wyuczenia się jego podstawowych elementów składniowych i semantycznych. Za pomocą Ady — tak jak za pomocą każdego innego języka programowania — można zapisać niestrukturalne sekwencje instrukcji, co oczywiście nie jest jeszcze dostatecznym powodem do stosowania tego języka w programowaniu. O wiele ważniejsze jest metodyczne posługiwanie się elementami języka i związanymi z nim technikami programowania. Z tego względu w poniższym wprowadzeniu do języka Ada reprezentujemy stanowisko, że zadaniem programisty nie jest badanie możliwych znaczeń elementów języka, poprawnych z punktu widzenia definicji języka lub dopuszczalnych z punktu widzenia translatora, lecz zdobycie umiejętności posługiwania się tylko takimi kombinacjami elementów języka, którym można przypisać określony sens w trakcie rozwiązywania problemu. W książce tej podejmujemy więc próbę określenia, jakie elementy języka, jakie ich kombinacje i w jakich sytuacjach mogą być trafnie stosowane.

Spis treści:

Wprowadzenie. Omówienie języka. Tworzenie programów i rozłączna kompilacja. Pakiety. Sekwencyjne sterowanie przebiegiem programu. Procesy i równoległy przepływ sterowania. Typy, obiekty i operacje zdefiniowane pierwotnie. Jednostki rodzajowe (szablony programów). Obsługa wyjątków. Programowanie na poziomie maszyny. Środowisko języka. Dodatki (reguły leksykalne i składniowe, słowa zastrzeżone, niemiecka terminologia Ady).

M.M.

KSIĄŻKI WYDAWNICTWA MIT PRESS

Seria: Sztuczna inteligencja

K. Sugihara: Machine Interpretation of Line Drawings. 1986.

Książka z dziedziny grafiki komputerowej i widzenia maszynowego (ang. machine vision) poświęcona metodom przestrzennej interpretacji rysunków złożonych z linii. Przedstawiono kompletną teorię matematyczną i omówiono jej wykorzystanie w zagadnieniach widzenia maszynowego i projektowania wspomaganego komputerem.

A. Yonezawa, M. Tokoro (red.): Object-Oriented Concurrent Programming. 1986.

Zbiór prac omawiających zagadnienia leżące u podstaw japońskiego projektu piątej generacji: programowanie logiczne, równoległość obliczeń oraz systemy rozproszone. Przedstawiono kilka propozycji nowych języków programowania oraz omówiono wykorzystanie współbieżnego programowania obiektowego w sztucznej inteligencji, inżynierii oprogramowania, syntezie muzyki, systemach informacji biurowej oraz programowaniu systemowym.

G. Agha. Actors — A Model of Concurrent Computation in Distributed Systems. 1986.

Książka poświęcona podstawowym problemom współbieżności. Autor bada procesy obliczeń równoległych za pomocą tzw. modelu aktorów (ang. actors), wprowadzonego przez Hewitta. Model aktorów pozwala opisać dynamiczną rozbudowę oraz rekonfigurację systemu. Zaproponowane przez autora diagramy umożliwiają przejrzysty opis interakcji w asynchronicznych systemach współbieżnych.

W. Clancey: Knowledge-Based Tutoring — The Guidon Program. 1987.

W książce opisano pierwszą próbę wykorzystania reprezentacji regułowej oraz systemów ekspertowych w nauczaniu. Program Guidon może znaleźć praktyczne zastosowanie w naukach poznawczych (ang. cognitive science) oraz dydaktyce. Autor przedstawia techniki sztucznej inteligencji niezbędne do objaśniania wiedzy i budowy modelu uczenia. Szczegółowe opisy techniczne umożliwiają odtworzenie programu.

Seria: Modele obliczeniowe poznania i percepcji

D. Klahr, P. Langley, R. T. Neches (red.): Production System Models of Learning and Development. 1986.

Pierwsza książka w całości poświęcona modelom procesów poznawczych człowieka. Przedstawiono różne podejścia do modelowania procesów uczenia się, które są stosowane obecnie w naukach poznawczych.

J. H. Holland, K. J. Holyoak, R. E. Nisbett, P. R. Thagard: Induction — Processes of Inference, Learning and Discovery. 1986.

Dwóch psychologów, informatyk oraz filozof przedstawiają wspólną pracę o procesach wnioskowania indukcyjnego i uczenia się, zarówno ludzkiego jak i maszynowego. Autorzy podjęli pierwszą poważniejszą próbę interdyscyplinarnego potraktowania zagadnień rozwiązywania problemów oraz indukcji, stosując regułowe modele umysłu ludzkiego.

Marek Machura

PIOTR ZIELCZYŃSKI
Środowiskowe Centrum Obliczeniowe
CYFRONET
Instytut Energii Atomowej
Otwock-Świerk

MULTICOMP — system rozwiązywania zadań metodą przeszukiwania drzew (I)

W trakcie prac prowadzonych w Instytucie Automatyki Politechniki Warszawskiej nad dużym systemem automatycznego projektowania DIADES [2] pojawiła się potrzeba utworzenia systemu służącego do rozwiązywania zadań występujących w automatycznym projektowaniu układów cyfrowych. System taki powinien charakteryzować się łatwością opisu zadań oraz umożliwiać użytkownikowi oddziaływanie na proces obliczeniowy w celu zwiększenia efektywności wyznaczania rozwiązań.

Właściwą metodą rozwiązywania zadań kombinatorycznych automatycznego projektowania wydaje się być omawiane w tym artykule przeszukiwanie grafów typu drzewiastego. Metoda ta umożliwia uporządkowany przegląd przestrzeni rozwiązywania, a na jej efektywność można wpływać ograniczając przestrzeń poszukiwań oraz wprowadzając dyrektywy heurystyczne [1, 5].

W ostatnich latach powstał język Multicomp (ang. multistrategic combinatorial problem solver) służący do rozwiązywania zadań metodą przeszukiwania drzew [6, 7]. Dziedzinami zastosowań Multicompu są m.in.: projektowanie układów cyfrowych, teoria grafów, kombinatoryka, badania operacyjne, planowanie działań robotów oraz gry i zagadki logiczne.

W pierwszej części artykułu omówiono metody przedstawiania zadań w postaci drzew oraz podstawowe strategie przeszukiwania drzew. W drugiej części omówiony zostanie język Multicomp i przykłady rozwiązywania zadań za pomocą tego języka.

PRZEDSTAWIANIE ZADAŃ W POSTACI DRZEW

Jedną z metod rozwiązania zadania jest przedstawienie go w postaci drzewa i znalezienie rozwiązania na tym drzewie. Drzewem nazywa się graf skierowany, nie zawierający pętli w swojej strukturze. Graf ten składa się ze zbioru punktów zwanych węzłami oraz ze zbioru gałęzi łączących węzły.

Wiele zadań można sprowadzić do następującej postaci:

- dane są stany — początkowy i końcowy (przeważnie stan końcowy jest dany w postaci zbioru warunków, które mają być spełnione);
- celem rozwiązania jest znalezienie ciągu przekształceń przeprowadzających stan początkowy na stan końcowy;
- przekształcenie jednego stanu na drugi polega na zastosowaniu jednego z operatorów.

Przeszukiwanie drzew jest metodą heurystycznego poszukiwania rozwiązania, będącego jednym z zagadnień sztucznej inteligencji. W metodzie poszukiwania rozwiązań na drzewach każdemu stanowi odpowiadają węzły, a operatorom — gałęzie. Drzewo rozwiązania zadania konstruuje się następująco. Do węzła odpowiadającego stanowi początkowemu (węzeł ten nazywa się korzeniem) dołącza się po jednej gałęzi dla każdego z możliwych przekształceń. Gałęzie te łączą korzeń z węzłami odpowiadającymi stanom, które otrzymuje się po zastosowaniu do korzenia wszystkich dopuszczalnych operatorów. Do każdego z nowo otrzymanych węzłów dołącza się kolejne gałęzie według tych samych zasad, aż do otrzymania węzła końcowego. Metody poszukiwania rozwiązań na tak utworzonych drzewach zostaną rozpatrzone w dalszej części artykułu. Do danego zadania należy dobrać taką strategię, aby tworzony w trakcie rozwiązywania fragment drzewa był jak najmniejszy (zwiększa to efektywność rozwiązywania).

Poniżej podano kilka przykładów przedstawienia zadania w postaci drzewa.

Gra w osiem

Klasycznym przykładem zastosowania przeszukiwania drzew jest gra w osiem, cytowana prawie we wszystkich publikacjach na ten temat [4]. Rekwizytem do tej gry jest tablica z ośmioma tabliczkami ponumerowanymi od 1 do 8. Tabliczki można przesuwać na wolne miejsce. Celem gry jest osiągnięcie stanu końcowego ze stanu początkowego za pomocą minimalnej liczby ruchów (rys. 1).

2	8	3
1	6	4
7		5

1	2	3
8		4
7	6	5

Rys. 1. Przykładowy stan początkowy i końcowy dla gry w osiem

Przesuwanie tabliczek można zastąpić przesuwaniem wolnego miejsca. Są zatem następujące operatory:

1) przesuń wolne miejsce w prawo
2) przesuń wolne miejsce w lewo
3) przesuń wolne miejsce w górę
4) przesuń wolne miejsce w dół.

Na rysunku 2 przedstawiono graf dla tej łamigłówki.

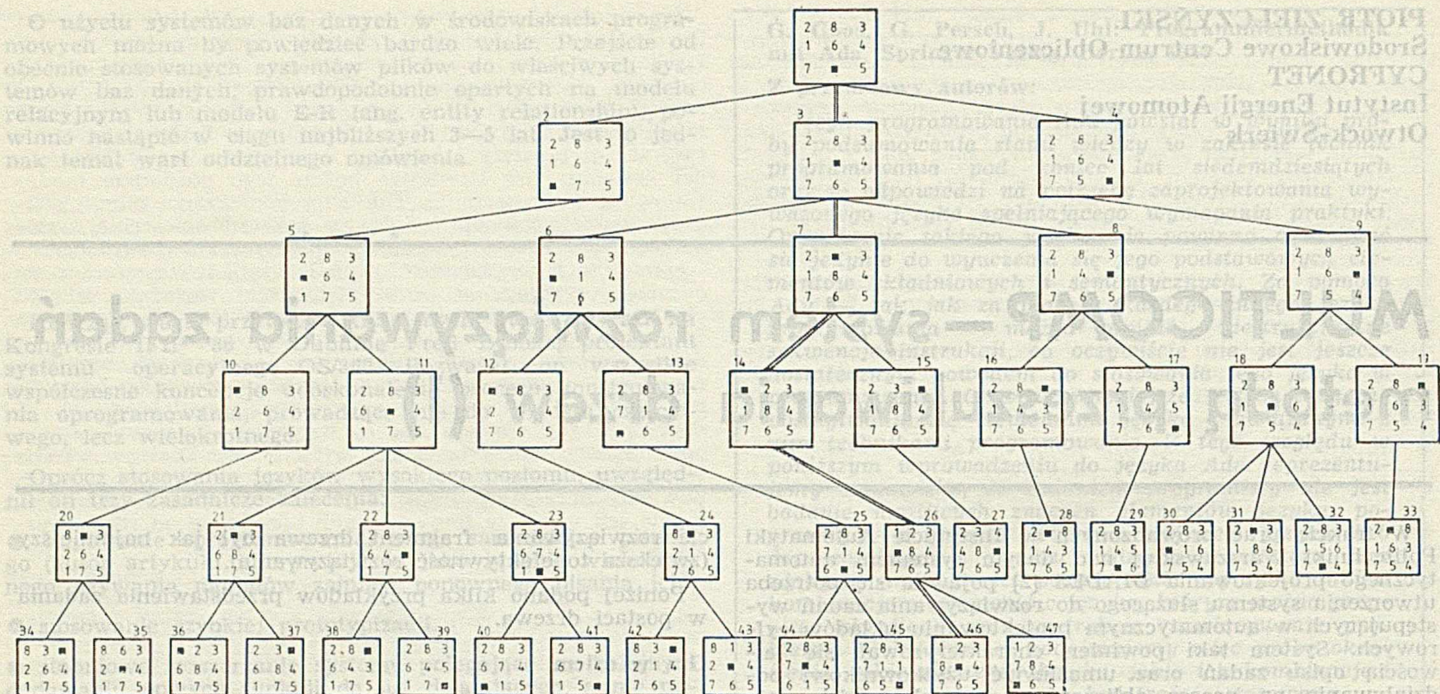
Znajdowanie ciągu wyrazów

Dane są dwa wyrazy o tej samej liczbie liter. Jeden z nich jest wyrazem początkowym, drugi — wyrazem końcowym. Należy znaleźć ciąg wyrazów spełniających następujące warunki:

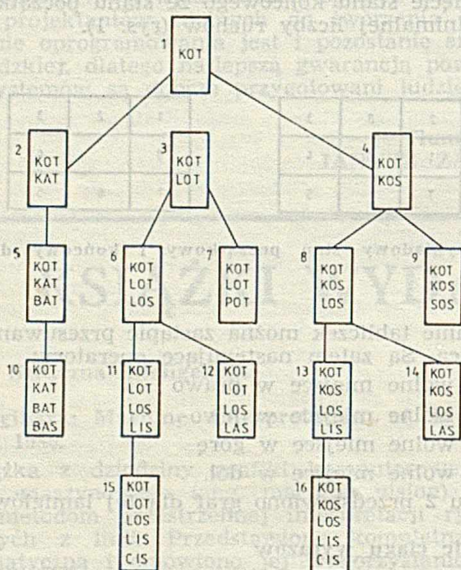
1) pierwszym elementem ciągu jest zadany wyraz początkowy;
2) każdy następny wyraz ciągu różni się od poprzedniego tylko jedną literą;
3) ostatnim elementem ciągu jest zadany wyraz końcowy.

Opisem stanu może tu być, na przykład, ciąg dotychczas wybranych wyrazów. Wówczas stanem początkowym jest jednoelementowy ciąg zawierający wyraz początkowy, a stanem końcowym — każdy ciąg, którego ostatnim elementem jest wyraz końcowy. Operatorem jest dołączenie (do aktualnego ciągu) wyrazu różniącego się od ostatniego tylko jedną literą. Na rysunku 3 przedstawiono przykładowe drzewo rozwiązania tego zadania dla wyrazu początkowego KOT i wyrazu końcowego CIS.

W przykładzie gry w osiem rozwiązaniem zadania była droga od stanu początkowego do końcowego. Zaś w przykładzie znajdowania ciągu wyrazów rozwiązaniem jest sam stan końcowy, gdyż w opisie stanu jest zapisana cała historia dojścia do niego. W implementacjach maszynowych pierwsze podejście oszczędza pamięć komputera, lecz wymaga bardziej skomplikowanych struktur sterowania.



Rys. 2. Fragment grafu dla gry w osiem

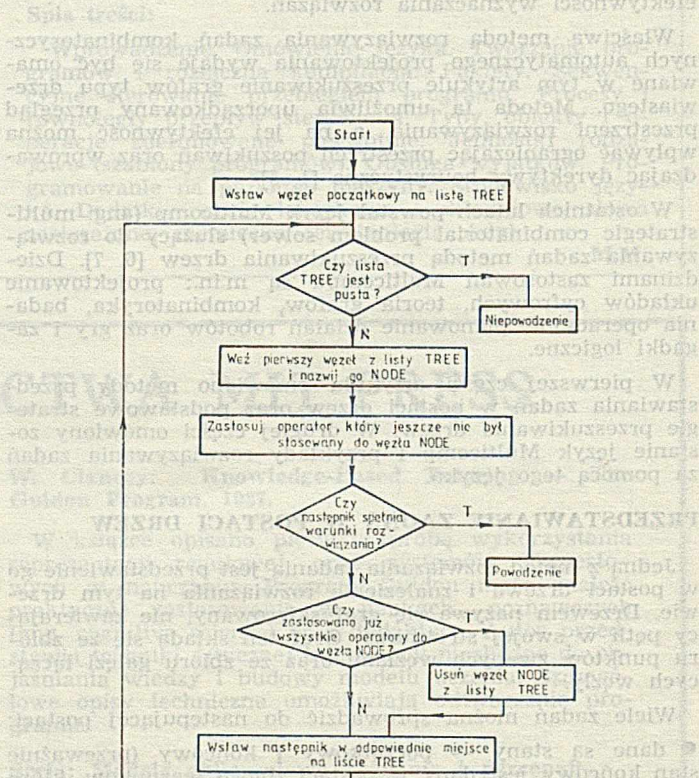


Rys. 3. Przykładowe drzewo znajdowania ciągu wyrazów

METODY PRZESZUKIWANIA DRZEW

Przed omówieniem podstawowych metod przeszukiwania przestrzeni stanów (w tym wypadku reprezentowanej przez drzewo), należy wprowadzić kilka pojęć. Następnikami danego węzła nazywa się węzły, które można otrzymać w wyniku zastosowania do niego poszczególnych operatorów. Następnikami węzła 3 na rysunku 2 są węzły 6, 7 i 8. Rozwinięciem węzła nazywa się utworzenie następników tego węzła. Rozwijany węzeł nazywa się poprzednikiem.

Ogólną metodę przeszukiwania drzew można zapisać, na przykład, w postaci schematu blokowego przedstawionego na rysunku 4. Główną strukturą jest w nim lista zawierająca węzły do rozwinięcia (nazwana TREE). Miejsce na liście TREE, w które wstawia się otrzymany następnik, zależy od wybranej strategii poszukiwania rozwiązania. Rysunek 4 przedstawia jeden z możliwych algorytmów przeszukiwania drzewa. Na wydruku przedstawiono program w Lispie realizujący ten algorytm. W programie założono, że zostały zdefiniowane funkcje SOLUTION, FIND_OPERATOR, PUTNODE, zależne od rozwiązywanego problemu (tabela 1). Funkcja PUTNODE zależy od strategii przeszukiwania drzew. Poniżej przedstawiono najczęściej używane strategie.



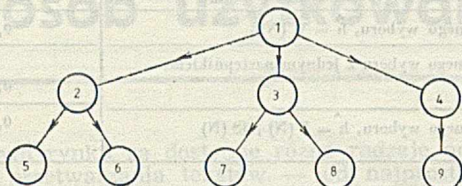
Rys. 4. Jeden z możliwych algorytmów przeszukiwania drzewa

Tabela 1. Funkcje SOLUTION, FIND_OPERATOR i PUTNODE

Nazwa funkcji	Argumenty	Wartość
SOLUTION	węzeł nowoutworzony	NIL, jeśli węzeł będący argumentem nie jest rozwiązaniem, dowolne inne wyrażenie — jeśli jest rozwiązaniem
FIND_OPERATOR	węzeł	operator (funkcja, za pomocą której można otrzymać następnik); jeśli jest to ostatni z operatorów, które można zastosować do węzła, to zmiennej FLAG musi zostać nadana wartość T
PUTNODE	węzeł, lista TREE	lista TREE z wstawionym węzłem

Strategia przeszukiwania wszerz

W metodzie tej węzły są rozwijane w takiej kolejności, w jakiej były tworzone. Na rysunku 5 przedstawiono kolejność rozwijania węzłów dla pewnego grafu metodą przeszukiwania wszerz. Dla drzewa z rysunku 2 węzły będą rozwijane w kolejności od 1 do 26. W trakcie rozwijania węzła 26 zostanie utworzony węzeł 46, co zakończy rozwiązywanie zadania. Dla drzewa z rysunku 3 węzły będą rozwijane od 1 do 11.



Rys. 5. Kolejność rozwijania węzłów metodą przeszukiwania wszerz

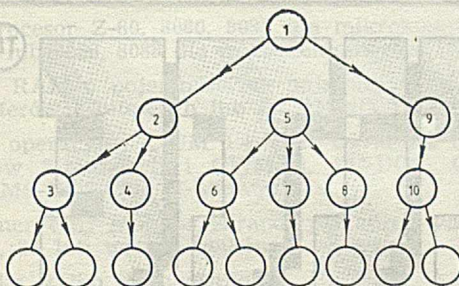
Zaletą tej metody jest jej prostota, a wadą — długi czas rozwiązywania i duże wymagania pamięciowe.

Funkcja PUTNODE realizująca strategię wszerz powinna wstawiać następnik na koniec listy TREE:

```
(DEF (PUTNODE (N TREE) (APPEND TREE (LIST N))))
```

Strategia przeszukiwania w głąb

Przeszukiwanie w głąb polega na pobieraniu do rozwinięcia węzła ostatnio utworzonego. Dla tego węzła znajduje się od razu wszystkie następniki. Ustala się pewną graniczną głębokość, po osiągnięciu której węzły nie są dalej rozwijane. Na przykład, dla granicznej głębokości 3, węzły drzewa z rysunku 6 będą rozwijane we wskazanej kolejności.



Rys. 6. Kolejność rozwijania węzłów metodą przeszukiwania w głąb

Przy tej strategii węzły z rysunku 3 będą tworzone w kolejności: 2, 3, 4, 5, 10, 6, 7, 11, 12, 15 (kolejność rozwijania węzłów: 1, 2, 5, 3, 6, 11). Metoda przeszukiwania w głąb wymaga mało pamięci, lecz czas jej działania jest długi. Przy niewłaściwie ustalonej głębokości granicznej metoda ta nie zapewnia znalezienia rozwiązania.

Funkcja PUTNODE realizująca tę strategię powinna wstawiać następnik na początek listy TREE, jeśli poprzednik został już z niej usunięty, natomiast w przeciwnym wypadku — na drugie miejsce (zaraz za poprzednikiem).

Strategia przeszukiwania w głąb z jednym następnikiem

Strategia ta różni się od poprzedniej tym, że nie dokonuje się od razu pełnego rozwinięcia węzła, lecz następniki tworzy się pojedynczo. Dla grafu z rysunku 3 węzły będą więc tworzone w kolejności: 2, 5, 10, 3, 6, 11, 15.

Funkcja PUTNODE powinna wstawiać następniki na początek listy TREE:

```
(DEF (PUTNODE (N TREE) (CONS N TREE)))
```

Strategia uporządkowanego wyboru

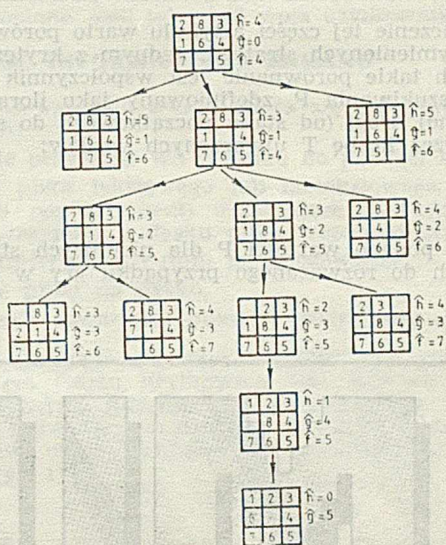
Niech h(N) oznacza cenę najkrótszej drogi od węzła N do najbliższego węzła końcowego, g(N) — cenę najkrótszej drogi od węzła początkowego do węzła N, natomiast f(N) — cenę najkrótszej drogi od stanu początkowego do końcowego przy założeniu, że droga ta przechodzi przez węzeł N. Z powyższych określeń wynika następująca równość:

$$f(N) = g(N) + h(N)$$

Bardzo rzadko funkcje h, g i f są dane jawnie, dlatego w czasie rozwiązywania zadań używa się na ogół oszacowań $\hat{f}$, $\hat{g}$ i $\hat{h}$ (znak $\hat{}$ wskazuje, że funkcja jest szacowana i może ulec zmianie po otrzymaniu nowych informacji).

W metodzie uporządkowanego wyboru, do rozwinięcia bierze się węzeł o najmniejszej wartości funkcji $\hat{f}$, zwanej funkcją ocen. Efektywność algorytmu zależy od tego, jak bardzo $\hat{f}$ jest zbliżona do f. Przeważnie rozwiązanie znajduje się wielokrotnie szybciej niż w przeszukiwaniu wszerz (rozwija się mniej węzłów).

Na przykład, dla gry w osiem jako funkcję $\hat{h}$ można przyjąć liczbę tabliczek nie znajdujących się na swoich miejscach: $\hat{h} = W(N)$. Wówczas węzły z rysunku 2 będą rozwijane w kolejności: 1, 3, 6, 7, 14, 26. Na rysunku 7 przedstawiono drzewo rozwiązania tego zadania metodą uporządkowanego wyboru. Obok węzłów podano wartości funkcji $\hat{f}$, $\hat{g}$ i $\hat{h}$.



Rys. 7. Drzewo rozwiązania gry w osiem metodą uporządkowanego wyboru

Dla gry w osiem jednym z najlepszych przybliżeń funkcji $\hat{h}$ jest:

$$\hat{h}(N) = \sum_{i=1}^{8} (P_i(N) + 3\,S_i(N))$$

gdzie: $P_i(N)$ oznacza odległość i-tej tabliczki od miejsca, w którym powinna się ona znaleźć;

$$S_i(N) = \begin{cases} 1 & \text{— jeżeli tabliczka znajduje się na pozycji centralnej;} \\ 0 & \text{— jeżeli w aktualnej konfiguracji za daną „niecentralną'' tabliczką znajduje się tabliczka, która ma się znajdować za nią w końcowej konfiguracji;} \\ 2 & \text{— w pozostałych wypadkach.} \end{cases}$$

Dla zadania z ciągiem wyrazów za h(n) można przyjąć liczbę liter, którymi różni się wyraz aktualny od wyrazu końcowego. Dla takiej funkcji ocen po rozwinięciu korzenia zostanie rozwinięty węzeł 4 (rys. 3).

W wypadku rozważanej strategii funkcja PUTNODE powinna obliczać wartość $\hat{f}(N)$ dla danego węzła i wstawiać go na listę TREE między węzły o sąsiednich wartościach $\hat{f}(N)$ (przy czym następnik nie może być wstawiony przed poprzednik, nawet gdy ma mniejszą wartość funkcji ocen).

Strategia uporządkowanego wyboru jest zwykle o wiele efektywniejsza niż poprzednio omówione metody.

```
(DEF (SOLVER (FIRST-NODE))                   Argumentem programu jest ko-
 (PROG NIL                                   rzeń FIRST-NODE.
   (SETQ TREE (LIST FIRST-NODE))             Wstaw korzeń  na listę TREE,
 ET (COND ((NULL TREE)                       Jesli TREE pusta, to
          (RETURN '$$$ PORAZKA$)))           zakończ pracę.
   (SETQ NODE (CAR TREE)))                   Weź pierwszy  węzeł  z TREE,
   (SETQ OPERATOR (FIND_OPERATOR NODE))           wybierz  operator,
   (SETQ SUCCESSOR (EVALQUOTE OPERATOR NODE))     utwórz   następnik
                                                  działając operato-
                                                  rem na poprzednik.
   (COND ((SOLUTION SUCCESSOR)               Jeśli nastepnik jest rozwią-
                                             zaniem, to
          (PRINT SUCCESSOR)                  wydrukuj go
          (RETURN)))                         i zakończ działanie.
   (COND (FLAG                               Jeśli zmienna FLAG (oznacza-
                                             jąca  zakonczenie rozwijania
                                             węzla) ma wartosc T,to
        (SETQ TREE (CDR TREE))))             usuń ten węzeł z TREE.
   (PUTNODE SUCCESSOR TREE)                  Wstaw następnik  w odpowied-
                                             nie  miejsce na liście TREE.
   (GO ET]                                   Powtórz procedurę.
```

Strategia uporządkowanego wyboru z jednym następnikiem

Strategia ta różni się od poprzedniej tylko tym, że następnik wstawia się na listę TREE według rosnących wartości funkcji ocen, niezależnie od tego, czy poprzednik miał większą czy mniejszą wartość funkcji $\hat{f}(N)$.

* * *

Na zakończenie tej części artykułu warto porównać efektywność wymienionych strategii. Jednym z kryteriów umożliwiających takie porównanie jest współczynnik kierunkowości przeszukiwania P, zdefiniowany jako iloraz długości L znalezionej drogi (od stanu początkowego do stanu końcowego) przez liczbę T utworzonych węzłów:

$$P = \frac{L}{T}$$

W tabeli 2 podano wartości P dla niektórych strategii zastosowanych do rozważanego przypadku gry w osiem.

Tabela 2. Współczynnik kierunkowości P dla niektórych strategii rozważanego przypadku gry w osiem

Strategia przeszukiwania	Wartość P
W głąb (głębokość graniczna 6)	0,100
W głąb z jednym następnikiem (głębokość graniczna 6)	0,104
Wszerz	0,109
W głąb (głębokość graniczna 5)	0,156
W głąb z jednym następnikiem (głębokość graniczna 5)	0,167
Uporządkowanego wyboru, $\hat{h} = W(N)$	0,385
Uporządkowanego wyboru z jednym następnikiem, $\hat{h} = W(N)$	0,385
Uporządkowanego wyboru, $\hat{h} = P(N)+3S(N)$	0,417

LITERATURA

[1] Łokucijewski R.: Kryteria wyboru struktury układu kombinacyjnego. Praca dyplomowa, Politechnika Warszawska, 1977

[2] Mielicki S.: Optymalizacja sieci działań w systemie automatycznego projektowania układów cyfrowych DIADES. Praca dyplomowa, Politechnika Warszawska, 1981

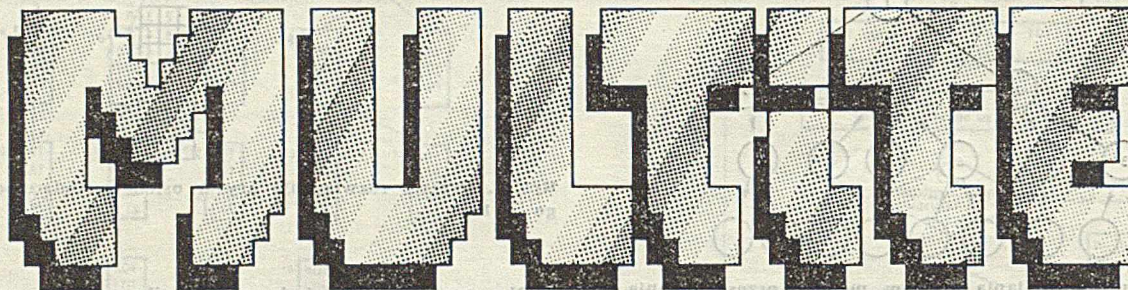
[3] Nilsson N. J.: Principles of artificial intelligence. Tioga P. C., Palo Alto, 1980

[4] Nilsson N. J.: Problem-solving methods in artificial intelligence. McGraw-Hill, New York, 1971

[5] Perkowski M.: Metoda rozwiązywania zadań kombinatorycznych w automatycznym projektowaniu układów cyfrowych. Praca doktorska, Politechnika Warszawska, 1980

[6] Perkowski M.: Wielocelowy i wielostrategiowy program rozwiązywania zadań kombinatorycznych. III Sympozjum „Metody Heurezy". Polskie Towarzystwo Cybernetyczne, Warszawa, 1976

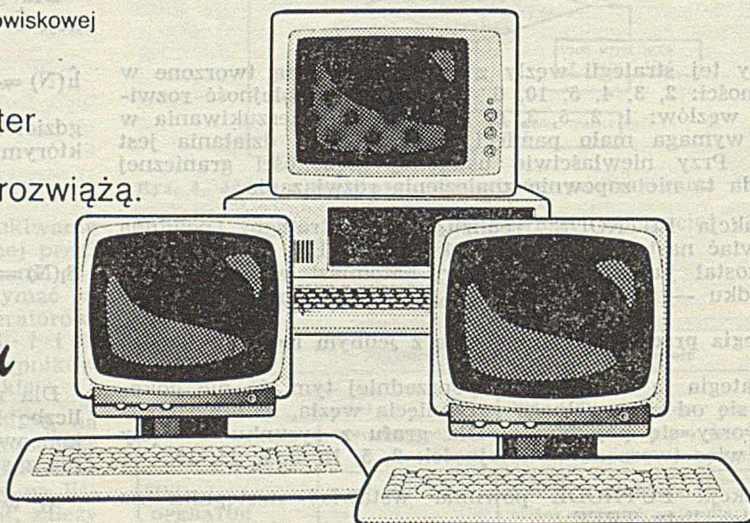
[7] Zielczyński P.: Implementacja nowej wersji języka opisu zadań kombinatorycznych Multicomp. Praca dyplomowa, Politechnika Warszawska, 1982



Zestawy wieloterminalowe do pracy wielostanowiskowej z komputerem klasy IBM PC/XT/AT

Jeśli chcecie Państwo lepiej wykorzystać swój komputer to zestawy multiTe sprawnie i szybko ten problem rozwiążą.

Liczba stanowisk stosownie do potrzeb. Możliwe nawet zestawy 8-terminalowe.



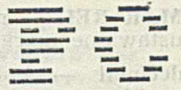
Nasze terminale gwarancją sukcesu

ZEKOM

ZAKŁAD ELEKTRONIKI KOMPUTEROWEJ
Skrytka pocztowa nr 35, 90-955 Łódź 8 tel. 34-30-49

EO/307/87

STANISŁAWA OSSOWSKA
Gdańsk



Wordstar 3.30 — zasady działania i sposób użytkowania (I)

Na naszym rynku są dostępne różne rodzaje oprogramowania do przetwarzania tekstów — od najprostszych, np. ED i TEX, przez edytory ekranowe Wordstar, Word 2000, Perfect Writer, aż po najbardziej wyszukane, jak MacPrint. Istnieją również edytory tekstowe, będące częścią bardziej złożonych systemów, jak Turbo Pascal czy Framework.

Wordstar 3.30 jest pakietem programowym czyniącym z mikrokomputera wygodniejszą niż tradycyjna maszynę do pisania. Może być stosowany na wszystkich niemal mikrokomputerach profesjonalnych dostępnych na rynku polskim. Jest narzędziem już sprawdzonym, niezawodnym, powszechnie używanym w krajach o wyższym poziomie zastosowań informatyki.

Celem artykułu jest przybliżenie czytelnikowi struktury, podstawowych zasad działania oraz funkcji Wordstara. Bogatszy opis działania jego instrukcji można znaleźć w podręcznikach opracowanych w języku polskim albo w oryginalnej dokumentacji.

WYMAGANIA SPRZĘTOWO-PROGRAMOWE

Wordstar 3.30 może być wykorzystywany na mikrokomputerach 8- i 16-bitowych. Wymagane są następujące parametry sprzętu:

- mikroprocesor Z-80, 8080, 8085 dla mikrokomputerów 8-bitowych lub 8086, 8088 dla mikrokomputerów 16-bitowych,
- pamięć RAM o pojemności co najmniej 48 KB dla mikrokomputerów 8-bitowych lub 64 KB dla 16-bitowych,
- system operacyjny CP/M 1.4 bądź CP/M 2.2 dla mikrokomputerów 8-bitowych i CP/M-86, PC-DOS, Concurrent DOS lub MS-DOS dla 16-bitowych,
- alfanumeryczny monitor ekranowy z adresowalnym kursorem (64, 80 lub więcej znaków w wierszu),
- drukarka (tylko do wydruku),
- pamięć dyskowa.

Na dysku powinny być zapisane następujące pliki:

- WS.COM (lub plik o innej nazwie utworzony podczas instalowania, będący główną częścią Wordstara),
- WSMSGS.OVR,
- WSOVLY1.OVR.

Oprócz nich na dysku może się znajdować nakładka generatora wydruków MAILMRGE.OVR oraz korektor składni SPELSTAR.OVR wraz ze słownikiem SPELSTAR.DCT (Spellstar jest opracowany dla języka angielskiego).

Na naszym rynku rozpowszechniły się dwie wersje Wordstara: Wordstar 3.30, omówiony w tym artykule, oraz Wordstar 3.40, umożliwiający konwersację za pomocą znaków ikonograficznych, stosowany w mikrokomputerach 16-bitowych zgodnych z IBM PC.

Wordstar 3.30 umożliwia pisanie tekstu w dowolnym języku, jeżeli pozwala na to dostępny zestaw znaków; dialog z systemem jest prowadzony w języku angielskim. Oprócz wersji angielskich proponowane są także polskie odpowiedniki Wordstara, np. TEKSTCSK, CX-TEKST, umożliwiające częściową konwersację w języku polskim.

Wordstar instaluje się wprowadzając parametry dotyczące określonego środowiska komputerowego oraz potrzeb użytkownika. Fakt instalowania Wordstara w różnorodnych warunkach sprzętowo-użytkowych spowodował powstanie wielu jego odmian użytkowych. Wygenerowane podczas instalowania odmiany użytkowe mogą się różnić między sobą formatem strony (liczbą wierszy na stronie, liczbą znaku w wierszu, odstępem między wierszami, marginesami pionowymi i poziomymi), stopniem rozjaśniania wyświetlanej informacji, ustawieniem przełączników, zadaniami realizowanymi przez klawisze funkcyjne, sposobem interpretacji drukowania, czy wreszcie możliwościami osiągania efektów specjalnych podczas drukowania. W czasie instalowania ustalane są określone parametry, które mogą pozostać niezmienione, jeśli taka jest wola użytkownika.

ZAWARTOŚĆ INFORMACYJNA EKRANU

Przy opracowywaniu tekstu zaleca się następującą kolejność działań:

a) wczytanie programu WS z dysku do pamięci komputera,

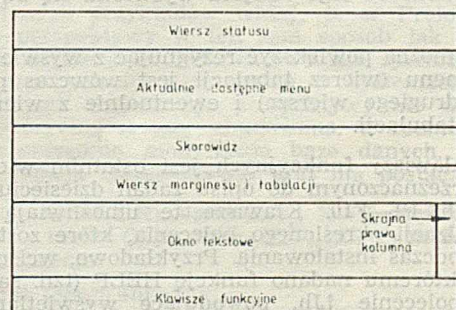
b) otwarcie pliku tekstowego lub nietekstowego (typu document lub non-document) oznaczające założenie nowego pliku lub wczytanie z dysku pliku wcześniej utworzonego,

c) pisanie treści pliku i (lub) jej poprawianie,

d) zapisanie pliku na dysku,

e) drukowanie zawartości pliku w miarę potrzeb.

W wyodrębnionych powyżej etapach prac związanych z redagowaniem tekstu użytkownik jest wspomagany przez Wordstar różnymi informacjami pomocniczymi. W związku z tym ekran jest podzielony na kilka różnych fragmentów, tzw. okien, pełniących różnorodne funkcje informacyjne (rys. 1).



Rys. 1. Zawartość informacyjna ekranu

Wiersz statusu jest pierwszym od góry wierszem na ekranie zawierającym informacje o aktualnym stanie redagowania pliku:

— aktualne polecenie, jeśli takie zostało zainicjowane, np. ↑Ji,

— nazwę otwartego pliku poprzedzoną symbolem przydzielonego napędu dyskowego, np. **A:NAUKA,**

— aktualny adres kursora (numer strony, numer wiersza i numer kolumny w wypadku pliku tekstowego (np. **PAGE 1 LINE 1 COL 1**) lub numer znaku w pliku, numer wiersza w pliku oraz numer znaku w wierszu — w wypadku pliku nietekstowego (np. **FC=1, FL=1 COL=1**)),

— wskaźniki dodatkowe, wskazujące na aktualny stan ustawienia poleceń:

WAIT — wykonywanie operacji dyskowej, podczas której należy wstrzymać się z pisaniem ze względu na możliwość utraty wprowadzanych znaków,

MAR REL — możliwość wpisywania treści poza aktualnie ustawione marginesy,

decimal — kolumna tabulacji dziesiętnej; wprowadzane znaki są przesuwane w lewo do wprowadzenia kropki dziesiętnej bądź wystąpienia nadmiaru (wówczas są przesuwane w prawo),

INSERT ON — tryb wstawiania znaków powodujący przesuwanie w prawo dotychczasowego tekstu będącego na prawo od kursora (jeśli nie jest wyświetlany, to pisana treść jest umieszczana na miejscu usuwanej treści poprzedniej),

LINE SPACING n — wielkość odstępu między wierszami (n może przybierać wartości od 1 do 9; odstęp równy jeden, zwykle nie jest wyświetlany),

PRINT PAUSED — wstrzymanie wydruku na drukarce zgodnie z uprzednim poleceniem użytkownika,

REPLACE (Y/N) — pytanie związane z poleceniem zmiany ciągów znaków wymagające potwierdzenia przez użytkownika.
Przed otwarciem pliku w wierszu statusu wyświetla się informacja **not editing**, wskazująca stan przed otwarciem pliku.

Aktualnie dostępne menu zajmuje osiem kolejnych wierszy, czyli wiersze 2—9 od góry ekranu i jest wyświetlane, jeśli ustawiono najwyższy poziom samouczka (ang. help). Siedem rodzajów menu udostępnia różne rodzaje poleceń.

Skorowidz (zwany również **katalogiem**) jest listą nazw plików, znajdujących się na przydzielonym dysku. Wyświetlanie skorowidza zależy zarówno od odmiany użytkowej Wordstara, jak i od polecenia użytkownika. Zwykle skorowidz jest wyświetlany w trakcie otwierania pliku.

Wiersz marginesów i tabulacji jest kolejnym wierszem od góry ekranu pojawiającym się po otwarciu pliku. Wskazuje miejsce ustawienia lewego (litera L) i prawego (litera R) marginesu, a ponadto miejsca tabulacji zmiennej, oznaczone znakiem ! (wykrzyknika) w wypadku tabulacji normalnej, albo znakiem # w wypadku tabulacji dziesiętnej. Są to miejsca, do których zostanie przesunięty kursor po naciśnięciu klawisza <TAB>, <HT>, lub ↑I w zależności od rodzaju klawiatury.

Okno tekstowe jest to dolna część ekranu znajdująca się poniżej wiersza marginesów i tabulacji, wykorzystywana do udostępniania fragmentu redagowanego pliku. Zajmuje ona czternaście kolejnych wierszy, jeśli nie polecono inaczej. Okno tekstowe jest udostępniane po otwarciu pliku (przed jego otwarciem od tego miejsca wyświetla się zwykle skorowidz).

Okno to można powiększyć rezygnując z wyświetlania dostępnego menu (wiersz tabulacji jest wówczas przemieszczany do drugiego wiersza) i ewentualnie z wiersza marginesów i tabulacji.

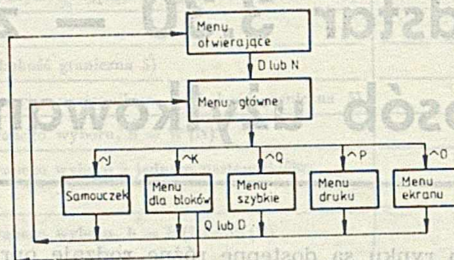
Wiersz **klawiszy funkcyjnych** jest ostatnim wierszem na ekranie, przeznaczonym do opisu zadań dziesięciu klawiszy funkcyjnych F1—F10. Klawisze te umożliwiają zainicjowanie realizacji określonego polecenia, które zostało zdefiniowane podczas instalowania. Przykładowo, wciśnięcie klawisza F1, któremu nadano funkcję HELP (tzn. samouczek), generuje polecenie ↑Jh, powodujące wyświetlanie informacji o poziomach samouczka i umożliwiające ustawienie pożądanego poziomu.

Skrajna prawa kolumna zawiera znaki mówiące o rodzaju informacji zawartych w danym wierszu tekstu. Znaczenie tych znaków jest następujące:

- **dwukropek (:)** — puste miejsce przed początkiem pliku,
- **spacja** — wiersz jest częścią większego akapitu (domyślne przejście do następnego wiersza),
- **znak mniejszości (<)** — koniec akapitu spowodowany naciśnięciem klawisza RETURN,
- **plus (+)** — kontynuacja wiersza poza ekranem,
- **znak zapytania (?)** — włączona analiza syntaktyczna poleceń lub sygnalizacja błędnego polecenia z kropką,
- **duże P** — znacznik końca strony występujący przy plikach tekstowych,
- **gwiazdka (*)** — puste miejsce za końcem pliku,
- **minus (—)** — zaznaczenie wiersza, w którym będzie drukowany wiersz następny.

POLECENIA WORDSTARA

W trakcie pracy Wordstar wyświetla listę aktualnie dostępnych poleceń w postaci siedmiu list menu. Sekwencję, w jakiej są one dostępne, oraz polecenia, które je inicjują, przedstawiono na rys. 2.



Rys. 2. Struktura list menu wspomagających przetwarzanie tekstowe

Menu otwierające (ang. opening menu) wyświetla się po wczytaniu pliku WS do pamięci komputera i umożliwia wykonanie czynności wstępnych przed przystąpieniem do redagowania tekstu. Realizacja poleceń zawartych w tym menu jest inicjowana przez naciśnięcie odpowiedniej litery na klawiaturze.

Menu otwierające zawiera pięć grup poleceń (rys. 3).

```
      not editing
              < < <  O P E N I N G   M E N U  > > >
 ---Preliminary Commands---   | --File Commands--  | -System Commands-
L  Change logged disk drive   |                    |  R  Run a program
F  File directory    now ON   |  P  PRINT a file   |  X  EXIT to system
H  Set help level             |                    |
 ---Commands to open a file---|  E  RENAME a file  | -WordStar Options-
   D  Open a document file    |  C  COPY   a file  |  M  Run MailMerge
   N  Open a non-document file|  Y  DELETE a file  |  S  Run SpellStar

directory of disk A:
 SPELSTAR.DCT COMMAND.COM  WS.COM        MAILMRGE.OVR SPELSTAR.OVR WSMSGS.OVR
 WSOVLY1.OVR

1HELP  2INDENT 3SET LM 4SET RM 5UNDLIN 6BLDFCE 7BEGBLK 8ENDBLK 9BEGFIL 10ENDFIL
```

Rys. 3. Polecenia zawarte w menu otwierającym

● **Polecenia wstępne** (ang. preliminary commands)

L — zmiana przydzielonego napędu dyskowego;

F — wyświetlanie bądź wyłączanie wyświetlania skorowidza dysku w przydzielonym napędzie dyskowym;

H — ustawienie pożądanego poziomu samouczka (w miarę opanowania poleceń można rezygnować z wyświetlania pełnej listy menu oraz komentarzy, co zwiększa efektywność pracy Wordstara).

● **Polecenia dotyczące otwarcia pliku** (ang. commands to open a file):

D — otwarcie pliku tekstowego (nowo tworzonego lub już istniejącego na dysku); plik tekstowy jest rozumiany jako dowolny maszynopis, składający się z jednej lub wielu stron, np. list, artykuł, sprawozdanie; pracując z plikiem tekstowym użytkownik ma do dyspozycji takie udogodnienia, jak tabulację, ustawianie marginesów, stronicowanie itp.;

N — otwarcie pliku nietekstowego, mające zastosowanie głównie przy pisaniu programów źródłowych.
Można otwierać plik już istniejący na dysku lub nowo tworzony. W celu otwarcia pliku należy podać jego nazwę, która składa się z ośmiu znaków (z wyjątkiem spacji), przy czym pierwszy jest literą.

● **Polecenia dla plików** (ang. file commands):

P — drukowanie treści pliku zapisanego na dysku;

E — zmiana nazwy pliku zapisanego na dysku;

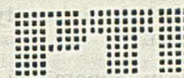
O — kopiowanie pliku zapisanego na dysku;

Y — usuwanie pliku z dysku.

Realizacja tych poleceń wymaga podania nazwy pliku lub plików oraz udzielenia odpowiedzi na zadane pytania.

dokończenie na str. 13

WŁODZIMIERZ GRUDZIŃSKI
Instytut Informatyki
Uniwersytet Warszawski



Zastosowanie Prologu w bazach danych (2)

W pierwszej części artykułu omówiono w zarysie idee leżące u podstaw programowania w logice, jego związki z bazami danych oraz podstawowe właściwości języka Prolog. Część druga jest poświęcona sposobom wykorzystania Prologu w systemach baz danych.

BAZA DANYCH W PROLOGU

Z poprzedniej części artykułu widać, w jaki sposób można zaimplementować w Prologu prostą relacyjną bazę danych. Dane, czyli krotki, są opisywane za pomocą klauzul unarnych, tj. klauzul zawierających jedynie nagłówek, np.

```
ojciec(jan, piotr).
mężczyzna(jan).
```

Klauzule te nie zawierają zmiennych (nie zezwala na to relacyjny model danych), mimo że np. klauzula unarna ojciec(X, jan) może reprezentować informację, o tym, że Jan ma ojca. Tego rodzaju klauzule, wprowadzające do bazy danych niepełną informację, nie będą tutaj rozważane. Warto jednak zwrócić uwagę na to, w jak naturalny sposób można zapisywać takie informacje, choć oczywiście nie zmniejsza to trudności w określeniu semantyki związanych z nimi operacji.

W związku z brakiem typów danych w Prologu nie ma bezpośrednich możliwości definiowania dziedzin atrybutów relacji. Można to zrobić korzystając z dostępnych procedur standardowych, np. procedury **integer (T)**, sprawdzającej czy **T** jest liczbą całkowitą. Utrudnia to operowanie bazą danych, gdyż trzeba pisać programy sprawdzające poprawność danych. Istnieją już jednak wersje Prologu, np. Turbo Prolog na IBM PC, w których wprowadzono proste typy danych. Definicja relacji i krótek w Turbo Prologu wygląda następująco:

```
domains
   osoba = symbol
   wiek = integer
predicates
   mężczyzna(osoba, wiek)
   ojciec(osoba,osoba)
   kobieta(osoba)
clauses
   mężczyzna(piotr,35).
   mężczyzna(jan,68).
   ojciec(jan,piotr).
   kobieta(maria).
   kobieta(X) :— not(mężczyzna(X,_)).
```

W powyższej definicji relacji **kobieta** tylko jedna krotka została wprowadzona jawnie i tylko ją otrzyma się w odpowiedzi na żądanie znalezienia wszystkich kobiet. Taka definicja jest wygodna wówczas, gdy przewiduje się potrzebę sprawdzania, czy ktoś jest kobietą.

W systemach baz danych przyjmuje się, że prawdziwe są jedynie te fakty, które są zapisane w bazie danych (klauzule występujące w programie) lub dające się z nich wyprowadzić. Jest to założenie o tzw. zamkniętości świata bazy danych [14]. W podobny sposób jest zaimplementowana w Prologu operacja zaprzeczenia. W konsekwencji odpowiedź na pytanie, czy Antoni jest kobietą, będzie twierdząca. Nie jest on wymieniony wśród mężczyzn, a zatem — na mocy założenia o zamkniętości świata — nie jest mężczyzną, a więc — zgodnie z definicją relacji **kobieta** — jest kobietą.

W następnych przykładach pokazano, w jaki sposób można zapisać podstawowe operacje algebry relacji (jeszcze jedna interpretacja procedury). Załóżmy, że mamy zdefiniowane dwie dwuargumentowe relacje R(A,B) i S(C,D) oraz:

rzut	$\pi_A(R)$	R1(A):—R(A,B).
selekcja i rzut	$\pi_B(\sigma_{A=5}(R))$	R1(B):—R(5,B).
selekcja	$\sigma_{C<D}(S)$	S1(C,D):—S(C,D),C<D,
złączenie	$R \underset{A=D}{\rhd\!\lhd} S$	RS(A,B,C):—R(A,B),S(C,A).

Koniunkcje warunków selekcji zapisuje się w jednej klauzuli, a dysjunkcje w kilku kolejnych klauzulach. Inne podstawowe operacje przedstawia się równie prosto; sumę relacji jako:

```
suma(X,Y) :— R(X,Y).
suma(X,Y) :— S(X,Y).
```

a różnicę jako:

```
różnica(X,Y) :— R(X,Y), not (S(X,Y)).
```

Warto zauważyć, że gdy dopuści się występowanie klauzul unarnych ze zmiennymi, to powyższa procedura da nieprawidłowe rezultaty.

Aktualizację można wykonywać za pomocą standardowych procedur **assert** i **retract**, odpowiednio wstawiającej i usuwającej dowolną klauzulę. Efekty tych procedur nie są anulowane podczas nawrotu, co utrudnia nieco ich stosowanie, trzeba bowiem zmienić sposób myślenia o programie. Maleje także czytelność programu. Istnieją propozycje wprowadzenia innych operacji umożliwiających aktualizację klauzul (bazy danych) i nie naruszających logicznej semantyki języka [18].

Z omówionych przykładów widać, że w Prologu można definiować perspektywy w ten sam sposób jak zapytania. Jednak tak definiowane perspektywy nie spełniają swoich funkcji ochronnych (co jest ważne w systemach baz danych), ponieważ aktualizacja perspektywy nie wpływa na relacje, z których jest ona zbudowana. W rezultacie użytkownik bezpośrednio modyfikuje bazę danych. Na przykład, jeśli relacja (perspektywa) **brat** ma postać:

```
brat(X,Y) :— ojciec(Z,X), ojciec(Z,Y), X≠Y, mężczyzna(X).
brat X,Y) :— matka (Z,X), matka(Z,Y), (X≠Y), mężczyzna(X).
```

to jej aktualizacja:

```
:—assert(brat (adam,anna)).
```

nie spowoduje żadnych zmian ani w procedurze **ojciec**, ani w procedurze **matka**. Jej skutkiem będzie dodanie trzeciej klauzuli w procedurze **brat**. Z drugiej strony widać, że w ten sposób można łatwo definiować wyjątki od reguł. Dużą zaletą Prologu jest także możliwość stosowania rekurencji do tworzenia perspektyw znacznie bardziej skomplikowanych niż jest to możliwe w systemach baz danych.

W Prologu można również w naturalny sposób zapisywać i utrzymywać więzy integralnościowe. Na przykład, reguła o posiadaniu tylko jednego ojca:

```
jedenojciec(X,Y) :— ojciec(Z,Y), !, fail.
jedenojciec(X,Y).
```

może wchodzić w skład procedury:

```
poprojciec(X,Y) :— jedenojciec(X,Y), !, ......
```

Ta procedura może być stosowana przy wprowadzaniu nowych elementów relacji **ojciec**, na przykład w sposób następujący: insert(**ojciec**(X,Y)) :— poprojciec(X,Y), assert(**ojciec**(X,Y)). Tego rodzaju więzy jak określone za pomocą procedury **jednojciec** opisują zależności funkcyjne. W [12] pokazano, jak można je zaimplementować w Prologu,

aby odcięcie wstawiane było automatycznie w sposób niewidoczny dla użytkownika.

Oczywiście nie należy zapominać, że Prolog jest normalnym (choć niekonwencjonalnym) językiem programowania, a nie wyspecjalizowanym językiem programowania baz danych, mimo iż ma wiele odpowiednich możliwości. Prolog znakomicie nadaje się na przykład do pisania translatorów, co dobrze ilustrują implementacje takich relacyjnych języków zapytań jak Query-by-Example [13] i Sequel. Implementację Sequela wykonano w Instytucie Informatyki Uniwersytetu Warszawskiego. Cały program (500 linii w Prologu!) implementujący nieco okrojoną wersję Sequela — Toy-Sequel — jest zamieszczony w [8]. Dzięki swojej zwięzłości i mocy Prolog świetnie się nadaje do szybkiego tworzenia i badania prototypów baz danych, a także zastosowań i systemów interakcyjnych zawierających wiele reguł i niezbyt dużo danych. Z tego powodu coraz chętniej jest stosowany do tworzenia systemów ekspertowych.

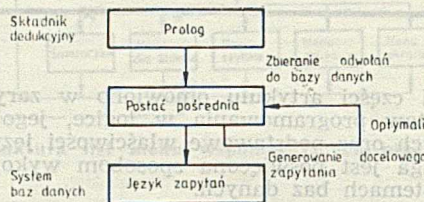
Istnieją jednak poważne niedogodności w stosowaniu Prologu do implementacji nawet średnich, np. rzędu kilku megabajtów, baz danych. Klauzule Prologu są przechowywane w pamięci operacyjnej, co znacznie ogranicza rozmiary bazy danych, a ponadto są wyszukiwane pojedynczo (jedna klauzula jest uzgadniana przy jednym odwołaniu do procedury), co czyni go bardzo wolnym. W wielu wersjach Prologu próbuje się złagodzić te wady. Wprowadzono operacje (np. **setof** i **bagof**), których wynikiem są zbiory klauzul (krotek). W niektórych implementacjach Prologu (MU-Prolog, Turbo Prolog) można przechowywać dane na plikach. W [17] pokazano, jak za pomocą prostego indeksowania można zmienić standardowy sposób wyszukiwania klauzul. Oczywiście, należy pamiętać o tym, że od systemu bazy danych wymaga się między innymi zapewnienia odtwarzania i ochrony danych, współbieżnego dostępu wielu użytkowników itp. Więcej informacji o zmianach w Prologu, ukierunkowanych na zastosowanie go jako języka implementacji baz danych, podano w dalszej części artykułu.

PROLOG JAKO JĘZYK ZAPYTAŃ

Dedukcyjne możliwości Prologu coraz częściej próbuje się wykorzystać w zastosowaniach wymagających większej liczby danych. Najpopularniejsza metoda polega na połączeniu go, jako tzw. składnika dedukcyjnego, z istniejącym systemem zarządzania bazą danych. Składnik dedukcyjny (SD) zarządza regułami dedukcyjnymi odnoszącymi się do danych utrzymywanych przez system bazy danych. Służy także do komunikacji z użytkownikami. Zwolennicy tego podejścia twierdzą, że oba łączone systemy mają tak różne cele i mechanizmy ich realizacji, iż próba stworzenia jednolitej całości nie może dać zadowalających rezultatów. Połączenie takie może być dokonane w sposób „luźny" lub „ścisły".

Pierwszy z nich polega na wyekstrahowaniu odpowiedniego fragmentu bazy danych i przeniesieniu go do pamięci zarządzanej przez SD (tzn. ten fragment staje się częścią systemu napisanego w Prologu). Operację tę wykonuje się przed rozpoczęciem korzystania z SD. Wyszukanie odpowiednich danych zapewnia system bazy danych. Należy jednak dokonać konwersji tych danych na postać wymaganą przez Prolog. Poważną wadą jest to, że nie wydaje się możliwe, aby proces wyboru fragmentu bazy danych do skopiowania mógł być (poza bardzo prostymi zastosowaniami) zautomatyzowany. Inną wadą jest statyczny charakter danych. Każda zmiana w bazie danych wymaga powtórzenia operacji kopiowania. Niewątpliwą zaletą jest łatwość stworzenia omawianego połączenia, ponieważ nie wymaga ono zmian w żadnej ze składowych, oczywiście pod warunkiem, że kopiowane dane mieszczą się w pamięci operacyjnej. W przeciwnym wypadku należy zaimplementować w SD obsługę danych przechowywanych na plikach, co znacznie zmniejsza atrakcyjność tego podejścia.

Sposób drugi polega na takim połączeniu obu składowych, aby baza danych stanowiła rozszerzenie SD. Krotki (rekordy) w bazie danych są traktowane jak klauzule unarne procedur (predykatów) z SD odpowiadających właściwym relacjom zdefiniowanym w schemacie pojęciowym. Konsekwencją naturalnego korzystania z takiego połączenia, często nazywanego podejściem interpretacyjnym, jest odwoływanie się do systemu bazy danych za każdym razem, gdy nastąpi uaktywnienie takiej klauzuli unarnej. Oznacza to obciążenie systemu bazy danych bardzo dużą liczbą niezależnych żądań poddawanych standardowemu procesowi kompilacji, optymalizacji, szeregowania itp. Stwarza to olbrzymi narzut czasowy nawet przy prostych zapytaniach. Innym problemem jest niemożliwość bezpośredniego przełożenia skompilowanych pytań (procedur) wyrażonych w Prologu, np. rekurencyjnych, na język zapytań systemu bazy danych. Proponowane rozwiązanie, tzw. podejście kompilacyjne, postuluje stworzenie języka pośredniego, służącego do komunikacji między Prologiem a systemem bazy danych. Architekturę takiego połączenia przedstawiono na rysunku.



Architektura połączenia między Prologiem a systemem bazy danych

Prolog wstrzymuje realizację odwołań do bazy danych tak długo, jak to jest możliwe. Oznacza to, że procedury tworzące reguły dedukcyjne są wykonywane dopóty, dopóki nie pozostaną jedynie odwołania do bazy danych. Żądanie użytkownika wyrażone w Prologu jest przetwarzane na postać pośrednią, w której odwołania do pojedynczych klauzul (krotek) są grupowane, a następnie zamieniane na żądania odnoszące się do zbiorów klauzul (krotek), np. relacji. Językiem pośrednim może być, na przykład, podzbiór Prologu bez zmiennych i z odwołaniami tylko do procedur odpowiadających relacjom bazy danych [5]. W kolejnym kroku następuje optymalizacja, m.in. przez usuwanie redundantnych żądań. Korzysta się przy tym z więzów integralnościowych z SD, a być może także ze schematu bazy danych. W tym momencie podejmowana jest również decyzja, czy wyniki zapytania mają być zapamiętane. Decyzja ta jest szczególnie ważna przy przetwarzaniu zapytań rekurencyjnych. W ostatnim etapie następuje przetłumaczenie otrzymanego zbioru żądań na język zapytań systemu bazy danych.

Przedstawione podejście, dzięki właściwościom Prologu, daje znacznie większe możliwości optymalizacji zapytań niż jest to aktualnie robione w systemach baz danych. Szczególnie ważna i trudna jest optymalizacja w wypadku zapytań rekurencyjnych, ponieważ problemy, o których wspomniano w pierwszej części artykułu, nie zależą od sposobu połączenia Prologu z bazą danych. W celu wykonania optymalizacji niezbędne jest także pobranie odpowiedniej informacji ze schematu bazy danych. Nie jest to wielkim problemem, gdy przyjmie się tradycyjne założenie o niewielkich rozmiarach schematu. W przeciwnym wypadku, a także, gdy otrzymane wyniki pośrednie zapytań są zbyt duże, powstaje problem zarządzania tymi danymi. Można odsyłać je do systemu bazy danych, tworząc oddzielną bazę danych, powoduje to jednak znaczne zwiększenie wzajemnych odwołań między obiema składowymi. Innym rozwiązaniem jest odpowiednie rozszerzenie możliwości Prologu.

Zaletą podejścia kompilacyjnego jest wykorzystanie istniejących już systemów. Pozwala to, przy pewnych ograniczeniach, dosyć szybko tworzyć różnorodne zastosowania korzystające z eksploatowanych w praktyce baz danych, bez konieczności wykonania bardzo dużych prac implementacyjnych. Znika zatem poważna przeszkoda przy tworzeniu systemów operujących bazami wiedzy, korzystających z metod modelowania i wnioskowania uzyskanych w badaniach nad sztuczną inteligencją. Ten sposób zastosowania Prologu do baz danych został przyjęty w japońskim projekcie komputerów piątej generacji [9]. Omawiany jest on także m.in. w [5], [10], [6], [20], [11].

PROLOG JAKO JĘZYK IMPLEMENTACJI BAZ DANYCH

Podejście łączeniowe nie jest jednak powszechnie akceptowane i ma swoich krytyków. Wskazują oni między innymi na zły podział pracy między obie składowe takiego systemu, duże narzuty związane z komunikacją oraz niepotrzebne powtarzanie wielu czynności (np. dwukrotna optymalizacja zapytań). Oba podsystemy wymagają także różnej

postaci danych, stąd częsta ich konwersja. W sumie, jak twierdzą krytycy tego podejścia, daje to produkt niezgrabny i nieefektywny. Dąży się zatem do stworzenia jednego, mającego wszystkie potrzebne cechy, systemu opartego na Prologu. Byłby on używany jako jednolity język implementacji systemu bazy danych i zastosowań. Nie jest to proste zadanie, wymaga bowiem rozwiązania wielu problemów, a także dużo pracy implementacyjnej. Nie należy więc spodziewać się w najbliższej przyszłości powstania takiego pełnosprawnego systemu, choć poczyniono już pierwsze próby w tym kierunku i przedstawiono propozycje pewnych rozwiązań.

Podstawowym zadaniem jest zapewnienie sprawnego przechowywania i wyszukiwania dużej liczby danych. Jednym ze sposobów jest użycie techniki dynamicznego kodowania mieszającego do organizacji plików klauzul. Jest to wygodne przy częstym w Prologu stosowaniu wyszukiwania z częściową odpowiedniością (ang. partial match retrieval). W tak skonstruowanym systemie obsługi bazy danych dla Prologu (napisanym w języku CDL-2 [3]) programista dysponuje nowymi procedurami systemowymi, umożliwiającymi działanie na klauzulach unarnych przechowywanych na plikach. Jednak w dalszym ciągu przetwarzane są pojedyncze klauzule, co silnie wpływa — zwłaszcza przy nawrotach — na efektywność wykonania zadania.

W innych pracach proponuje się zastosowanie znanych metod indeksowania. Zarys takiej implementacji w Prologu wraz z mechanizmem zarządzania buforami (również w Prologu) naszkicowano w [16]. Jedną z zalet tej propozycji jest możliwość indeksowania wszystkich klauzul, co jest istotne wówczas, gdy występuje wiele reguł dedukcyjnych (perspektyw). Poruszono tam również problemy optymalizacji i sterowania współbieżnością w takim systemie. Tym, co różni go od tradycyjnych systemów baz danych, jest m.in. występowanie nawrotów, inny rodzaj aktualizacji (zastąpionej usunięciem i wstawieniem), jednolity zapis reguł dedukcyjnych (schematu) i danych umożliwiających aktualizację jednych i drugich. Nie można zatem skopiować dobrze znanych i sprawdzonych algorytmów. Propozycje takich mechanizmów odtwarzania i sterowania współbieżnością, uwzględniających specyfikę Prologu, przedstawiono w [2]. Odtwarzanie opiera się na metodzie plików różnicowych, nazywanej także hipotetyczną bazą danych [19], rozszerzonej o obsługę efektów nawracania podczas poszukiwania odpowiedzi na zapytanie. Przy omawianiu drugiego zagadnienia rozważa się jednolitą metodę współbieżnego wykonania programów w Prologu sekwencyjnym i Concurrent Prologu [15]. Z kilku przedyskutowanych rozwiązań wybrano wersję dwufazowego blokowania [4], zmodyfikowaną odpowiednio do potrzeb Prologu.

* * *

Zagadnienie związane ze stosowaniem Prologu w bazach danych i stworzeniem bardziej „inteligentnych" systemów zarządzania bazami danych budzą coraz większe zainteresowanie. Wiele różnych propozycji, wraz z argumentacją na rzecz jednego lub drugiego podejścia, przedstawiono ostatnio na dwóch dużych konferencjach [7] i [1]. W wielu ośrodkach prowadzi się intensywne badania w tej dziedzinie. Należy się spodziewać, że wkrótce pojawią się także raporty opisujące praktyczne zastosowania.

LITERATURA

[1] Brodie M. L. (ed.): Proc. of Workshop on Large Scale Knowledge Base and Reasoning Systems, Islamorada, 1985
[2] Carey M., DeWitt D. J., Graete G.: Mechanisms for Concurrency Control and Recovery in Prolog — A Proposal. [7]
[3] Chomicki J., Grudziński W.: A Database Support System for Prolog, Proc. of Logic Programming Workshop. Albufeira, 1983
[4] Jordan J., Banerjee J., Batman R.: Precision Locks. Proc. of ACM SIGMOD Conf. on Management of Data, 1981
[5] Jarke M., Clifford J., Vassiliou Y.: An Optimizing Prolog Front-End to a Relational Query System. Proc. of ACM SIGMOD Conf. on Management of Data, Boston, 1984
[6] Jarke M., Vassiliou Y.: Coupling Expert Systems with Database Management Systems. Reitman W. (ed.) Artificial Intelligence Applications for Business. Ablex, Norwood, 1984
[7] Kerschberg L. (ed.): Proc of 1st Intern. Workshop on Expert Database Systems, Kiawah Island, 1984
[8] Kluźniak F., Szpakowicz S.: Prolog for Programmers, Academic Press, 1985
[9] Kunifuji S., Yokota H.: Prolog and Relational Database for Fifth Generation Computer Systems. Nicolas J-M (ed.), Proc. of Workshop on Logical Bases for Data Bases, Toulouse, 1982
[10] Li D.: A Prolog Database System, Research Institute Press, 1984
[11] Marque-Pucheu G. et al.: Interfacing Prolog and Relational Data Base Management Systems. Gardarin G., Gelenbe E. (eds.), New Applications of Data Bases, Academic Press, 1984
[12] Mendelzon A. O.: Functional Dependencies in Logic Programs. Proc. of 11th Conf. of VLDB, Stockholm, 1985
[13] Neves J. C., Anderson S. O., Williams M. H.: A Prolog Implementation of Query-by-Example. Schneider H. J. (ed.), Proc. of 7th Int. Computing Symposium, Nurnberg, 1983
[14] Reiter R.: On Closed World Databases. Gallaire H., Minker J. (eds.): Logic and Databases. Plenum Press, 1978
[15] Shapiro E. Y.: A Subset of Concurrent Prolog and Its Interpreter. Report TR-003, ICOT, Tokyo, 1983
[16] Sciore E., Warren D. S.: Towards an Integrated Database — Prolog System. [7]
[17] Warren D. H. D.: Efficient Processing of Interactive Relational Data Base Queries Expressed in Logic. Proc. of 7th Conf. of VLDB, Cannes, 1981
[18] Warren D. S.: Database Updates in Pure Prolog. Proc. of Intern. Conf. on Fifth Generation Computer Systems, Tokyo, 1984
[19] Woodfill J., Stonebraker M.: An Implementation of Hypothetical Relations. Proc. of 9th Conf. of VLDB, 1983
[20] Yokota H. i inni: An Enhanced Inference Mechanism for Generating Relational Algebra Queries. Proc. ACM SIGACT/ /SIGMOD Conf. on Principles of Database Systems, 1984.

Wordstar 3.30

dokończenie ze str. 10

● **Polecenia dla systemu operacyjnego** (ang. system commands):

R — inicjowanie realizacji programu; polecenie to umożliwia wykonanie innego programu lub polecenia systemowego; po wykonaniu programu następuje powrót do menu głównego;

X — powrót do systemu operacyjnego w celu zakończenia pracy Wordstara.

● **Opcje Wordstara** (ang. Wordstar options):

M — przejście do programu MailMerge działającego tylko w połączeniu z Wordstarem; program ten jest przydatny zwłaszcza przy tworzeniu wielu dokumentów z niewielką liczbą zmiennych, wprowadzanych do dokumentu podstawowego (np. przy przygotowywaniu korespondencji adresowanej do różnych odbiorców); program ten generuje bowiem wydruki na podstawie dwóch zbiorów informacji: zbioru podstawowego i zbioru zmiennych wprowadzanych albo z dysku, albo z klawiatury;

S — przejście do programu Spellstar, sprawdzającego poprawność ortograficzną i syntaktyczną redagowanego tekstu w języku angielskim (jest to możliwe dzięki słownikowi wyrazów oraz korektorowi składni — założonymi na dysku).

Przedsiębiorstwo Robót Wiertniczych i Górniczych
w Warszawie
poszukuje wykonawcy bądź zakupi
SYSTEM FINANSOWO-KSIĘGOWY
na mikrokomputer SM-4

Istnieje możliwość zlecenia osobie prywatnej opracowania systemu.

Poważne oferty: tel. 49-24-51 w. 189 lub 146
Przedsiębiorstwo Robót Wiertniczych i Górniczych
ul. Puławska 18, 00-975 Warszawa

EO/636/87

JAN BIELECKI
Instytut Informatyki
Politechnika Warszawska

Podstawy grafiki w języku Turbo Pascal (2)

Wykreślanie podstawowych obiektów graficznych

Przedstawione tu zasady wykreślania podstawowych obiektów graficznych dotyczą w równym stopniu trybów średniej rozdzielczości — ustanawianych za pomocą procedur GraphColorMode i GraphMode, jak trybu wysokiej rozdzielczości — ustanawianego za pomocą procedur HiRes. Należy jedynie przypomnieć, że w trybach średniej rozdzielczości obraz przedstawiany na ekranie składa się z układu 200×320 pikseli, a w trybie wysokiej rozdzielczości — z układu 200×640 pikseli. W każdym z tych wypadków punkt o współrzędnych (0,0) znajduje się w lewym górnym narożniku ekranu, a współrzędne punktów zwiększają się w prawo i do dołu. Oznacza to w szczególności, że prawy dolny narożnik ekranu ma w trybie średniej rozdzielczości współrzędną x=319 i współrzędną y=199.

WYKREŚLANIE PUNKTÓW

Wykreślanie punktów może być realizowane za pomocą procedury Plot. W ogólnym przypadku wywołanie tej procedury ma postać:

Plot (xCoord,yCoord,Color)

W zapisie tym xCoord, yCoord i Color są wyrażeniami typu integer. Wykonanie procedury Plot powoduje wyświetlenie jednego piksela w tym punkcie ekranu, który ma współrzędne (xCoord,yCoord). Wyświetlanie odbywa się w kolorze o numerze Color bieżącej palety barw. Paleta ta jest brana pod uwagę jedynie w trybach GraphColorMode i GraphMode. W trybie HiRes dla argumentu Color=0 następuje wyświetlenie punktu w kolorze czarnym, a dla argumentu Color=1 następuje wyświetlenie punktu w kolorze określonym za pomocą procedury HiResColor. W trybie HiRes tło wykresu jest zawsze czarne.

Przykład 1. Wykreślanie punktu w trybie GraphColorMode

```
program Dot;
begin
  GraphColorMode;
  Palette (1);
  GraphBackground(Blue);
  Plot (160,100,2);
  repeat until KeyPressed;
  TextMode
end.
```

● Wykonanie programu powoduje wykreślenie w trybie graficznym kolorowym średniej rozdzielczości, jednego punktu w środku ekranu.

● Cały ekran zostaje wypełniony kolorem tła — niebieskim, a punkt zostaje wyświetlony w kolorze 2 wybranej palety, tj. karmazynowym.

● Gdyby z programu usunięto wywołania procedur Palette i GraphBackground, to domniemaną paletą była paleta nr 0, a domniemanym kolorem tła — czarny. W takim wypadku punkt zostałby wyświetlony w kolorze czerwonym na czarnym tle.

WYKREŚLANIE ODCINKÓW I PROSTOKĄTÓW

Wykreślanie odcinków może być realizowane za pomocą procedury Draw. W ogólnym wypadku wywołanie tej procedury ma postać:

Draw(x1,y1,x2,y2,Color)

W zapisie tym x1, y1, x2, y2 i Color są wyrażeniami typu integer. Wykonanie procedury Draw powoduje wyświetlenie odcinka linii prostej łączącego punkty o współrzędnych (x1,y1) i (x2,y2). Odcinek jest wyświetlany w kolorze o numerze Color. Paleta jest brana pod uwagę jedynie w trybach GraphColorMode i GraphMode. W trybie HiRes dla argumentu Color=0 następuje wyświetlenie odcinka w kolorze czarnym, a dla argumentu Color=1 następuje wyświetlenie odcinka w kolorze określonym za pomocą procedury HiResColor. W trybie HiRes tło wykresu jest zawsze czarne.

Przykład 2. Wykreślanie odcinka w trybie HiRes

```
program Line;
begin
  HiRes;
  HiResColor(Red);
  Draw(0,0,639,199,1);
  repeat until KeyPressed;
  HiResColor(Green);
  Delay(10000);
  TextMode
end.
```

● Wykonanie programu powoduje wykreślenie głównej przekątnej ekranu.

● Przekątna jest wyświetlana w kolorze czerwonym.

● Po naciśnięciu dowolnego klawisza klawiatury kolor przekątnej zmienia się na zielony. Po upływie 10 s obraz znika.

Wśród predefiniowanych procedur graficznych nie występuje procedura do wykreślania prostokątów. Nic oczywiście nie stoi na przeszkodzie, aby procedurę taką zdefiniować za pomocą procedury Draw.

Przykład 3. Wykreślanie prostokąta

```
program Square;
procedure Box(xMin,yMin,xMax,yMax,Color : 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
  GraphMode;
  Box(0,0,319,199,3);
  repeat until KeyPressed;
  TextMode
end.
```

● Wykonanie programu powoduje wykreślenie na ekranie największego prostokąta o bokach równoległych do osi układu współrzędnych.

● Prostokąt jest wykreślany w kolorze nr 3 palety nr 0.

WYKREŚLANIE ŁUKÓW I OKRĘGÓW

Wykreślanie łuku odbywa się za pomocą procedury Arc, a wykreślanie okręgu odbywa się za pomocą procedury Circle.

Wywołanie procedury Arc ma w ogólnym przypadku postać:

Arc(xCoord,yCoord,Angle,Radius,Color)

W zapisie tym xCoord, yCoord, Angle, Radius i Color są wyrażeniami typu integer. Wykonanie procedury Arc powoduje wykreślenie łuku o promieniu Radius, w kolorze o numerze Color bieżącej palety barw. Wykreślanie łuku rozpoczyna się w punkcie o współrzędnych (xCoord,yCoord). Argument Angle określa kąt łuku wyrażony w stopniach. Jeżeli Angle>0, to łuk jest wykreślany w kierunku zgodnym z ruchem wskazówek zegara. Jeśli Angle<0, to jest wykreślany w kierunku przeciwnym.

Przykład 4. Wykreślanie łuku

```
program Quadrant;
{$i Graph.p}
begin
  HiRes;
  HiResColor(Red);
  Arc(0,199,—180,100,1);
  repeat until KeyPressed;
  TextMode
end.
```

● Wykonanie procedury powoduje wykreślenie półokręgu.

● Półokrąg jest oparty na lewej krawędzi ekranu jako na średnicy i jest wykreślany w kolorze czerwonym.

Wywołanie procedury CIRCLE ma w ogólnym przypadku postać:

Circle(xCoord,yCoord,Radius, Color)

W zapisie tym xCoord, yCoord, Radius i Color są wyrażeniami typu integer. Wykonanie procedury Circle powoduje wykreślenie okręgu o promieniu Radius i środku w punkcie o współrzędnych (xCoord, yCoord). Okrąg jest wykreślany w kolorze o numerze Color. W trybach graficznych o średniej rozdzielczości ma taki sam rozmiar w kierunku pionowym i poziomym. W trybie wysokiej rozdzielczości ma natomiast postać elipsy.

Przykład 5. Wykreślanie okręgu

```
program Circle;
{$i Graph.p}
begin
  GraphMode;
  Circle(99,99,99,1);
  repeat until KeyPressed;
  TextMode
end.
```

● Wykonanie programu powoduje wykreślenie największego okręgu, jaki bez obcięć mieści się na ekranie.

● Okrąg ten jest wykreślany w kolorze nr 1 i jest styczny od lewej oraz górnej krawędzi ekranu.

WYPEŁNIANIE OBSZARÓW

Wypełnianie obszarów jednolitym kolorem odbywa się za pomocą procedur FillShape i FillScreen. Wypełnienie obszaru wzorem odbywa się za pomocą procedury FillPattern. Do definiowania wzoru służy procedura Pattern.

Wywołanie procedury FillShape ma w ogólnym wypadku postać:

FillShape(xCoord,yCoord,Color,Border)

W zapisie tym xCoord, yCoord, Color i Border są wyrażeniami typu integer. Wykonanie procedury FillShape powoduje zlokalizowanie obszaru ograniczonego linią w kolorze o numerze Border, otaczającego punkt o współrzędnych (xCoord,yCoord), a następnie wypełnienie tego obszaru kolorem o numerze Color. Wymaga się, aby kolor wypełniający nie był kolorem tła.

Przykład 6. Wykreślanie wypełnionego prostokąta

```
program FilledSquare;
{$i Graph.p}
procedure Box(xMin,yMin,xMax,yMax,Color : integer);
begin
  Draw(xMin,xMin,xMax,yMax,Color);
  Draw(xMax,yMin,xMax,yMax,Color);
  Draw(xMax,yMax,xMin,yMax,Color);
  Draw(xMax,yMax,xMin,yMax,Color);
  FillShape(trunc((xMin + xMax) / 2,
            trunc((yMin + yMax) / 2,
            Color,Color)
end;
begin
  GraphColorMode;
  Palette(0);
  GraphBackrgound(Red);
  Box(0,0,159,99,1);
  repeat until KeyPressed;
  TextMode
end.
```

● Wykonanie programu powoduje wypełnienie całego ekranu kolorem czerwonym, a następnie wypełnienie lewej górnej ćwiartki ekranu kolorem zielonym.

Wywołanie procedury FillScreen ma w ogólnym przypadku postać:

FillScreen(Color)

W zapisie tym Color jest wyrażeniem typu integer. Wykonanie procedury FillScreen powoduje wypełnienie okienka graficznego kolorem o numerze Color.

Przykład 7. Wypełnianie okienka zadanym kolorem

```
program FillUp;
{$i Graph.p}
begin
  GraphColorMode;
  GraphBackground(Blue);
  Write('JanB');
  repeat until KeyPressed;
  FillCcrean (1);
  Delay(3000);
  TextMode
end.
```

● Wykonanie programu powoduje wyprowadzenie na niebieskim tle brązowego napisu JanB, a po naciśnięciu dowolnego klawisza klawiatury — wypełnienie całego ekranu kolorem zielonym.

● Wobec braku jawnego zdefiniowania okienka graficznego przyjmuje się, że jest nim cały ekran.

Wypełnienie obszaru dowolnym, uprzednio zdefiniowanym, wzorem odbywa się za pomocą procedury FillPattern. Pozostaje ona w ścisłym związku z procedurą Pattern.

Wywołanie procedury Pattern ma w ogólnym wypadku postać:

Pattern(Vector)

W zapisie tym Vector jest nazwą tablicy typu array [0..7] of byte. Wykonanie procedury Pattern powoduje ustalenie wzoru wykorzystywanego do wypełnienia obszaru za pomocą procedury FillPattern. Wzór ma postać tablicy 8×8 bitów, a na ekranie pojawia się jego lustrzane odbicie, zarówno względem osi pionowej, jak i poziomej.

Wywołanie procedury FillPattern ma w ogólnym przypadku postać:

FillPattern(xMin,yMin,xMax,yMax,Color)

W zapisie tym xMin, yMin, xMax i yMax oraz Color są wyrażeniami typu integer. Wykonanie procedury FillPattern powoduje wypełnienie prostokątnego obszaru wyznaczonego przez prostokąt o przeciwległych wierzchołkach (xMin,yMin) i (xMax,yMax) i ograniczonego obrzeżem w kolorze o numerze Color, wzorem zdefiniowanym za pomocą procedury Pattern. Wzór jest umieszczany w lewym dolnym rogu wypełnianego obszaru, a następnie powielany od dołu do góry i od lewej do prawej. Te bity wzoru, które mają wartość 0, nie powodują zmiany koloru punktów ekranu.

Przykład 8. Wypełnianie obszaru wzorem

```
program Arrows;
{$i Graph.p}
```

```
const
  Arrow : array [0..7] ob byte =
    ($00,$02, $04,$08,
     $90,$a0, $c0,$f0);
begin
  HiRes;
  Pattern(Arrow);
  FillPattern(0,0,639,199,1);
  repeat until KeyPressed;
  TextMode
end.
```

● Wykonanie programu powoduje wyświetlenie na ekranie strzałek skierowanych w stronę prawego górnego narożnika ekranu.

POSŁUGIWANIE SIĘ WEKTOREM BARW

Wykreślenie dowolnego obiektu wymaga podania koloru punktów składających się na obiekt. Każdy z kolorów jest określany umownie jako numer pozycji bieżącej palety barw. W szczególności dla palety nr 0 kolorem nr 2 jest kolor czerwony. Ponieważ każda z palet składa się z 4 kolorów, dopuszczalne numery kolorów mogą być tylko liczbami z przedziału 0..3. Tym niemniej w języku Turbo Pascal zezwolono na wyrażenie numeru koloru za pomocą liczby —1. W takim wypadku kolor punktów tworzących obiekt wynika z rozpatrzenia koloru punktów wyświetlanych na ekranie.

Bezpośrednio po aktywowaniu trybu graficznego obowiązuje ustalenie, że wykreślanie w kolorze o numerze —1 odbywa się w kolorze punktów ekranu. Ustalenie to może być zmienione za pomocą procedury ColorTable.

Procedura ColorTable ma w ogólnym przypadku postać:

ColorTable(Hue1,Hue2,Hue3,Hue4)

W zapisie tym Hue1, Hue2, Hue3 i Hue4 są wyrażeniami typu integer. Wykonanie procedury ColorTable powoduje zdefiniowanie wektora barw, branego pod uwagę podczas przyszłego wykreślania punktów w kolorze nr —1. Przyjmuje się, że argument Hue$_i$ określa, na jaki kolor ma być zmieniony kolor piksela wyświetlanego w kolorze i. W szczególności dla trybu HiRes wykonanie procedury:

ColorTable(1,0,2,3)

spowoduje, że wyświetlenie na ekranie dowolnej linii w kolorze o numerze —1 wywoła inwersję koloru pikseli położonych na tej linii. Inwersja polega na zmianie koloru tła na kolor określony przez procedurę HiResColor i odwrotnie.

Przykład 9. Wykreślanie przez wektor barw — tryb średniej rozdzielczości

```
program SwitchColors;
{$i Graph.p}
var
  i : byte;
begin
  GraphColorMode;
  Palette(1);
  for i := 0 to 10 do
    Draw(i,0,319,199 — i,2);
  repeat until KeyPressed;
  ColorTable(3,2,1,0);
  Draw(319,0,0,199,—1);
  Delay(5000);
  TextMode
end.
```

● Wykonanie programu powoduje wykreślenie paska wzdłuż głównej przekątnej ekranu.

● Po naciśnięciu dowolnego znaku klawiatury następuje wykreślenie drugiej przekątnej ekranu.

● Przekątna ta jest wykreślana w kolorze o numerze —1, a więc wykreślanie odbywa się przez wektor barw. Powoduje to, że punkty ekranu wyświetlane w kolorze nr 0 zostają zmienione na punkty wyświetlane w kolorze nr 3, a punkty wyświetlane w kolorze nr 2 zostają zmienione na punkty wyświetlane w kolorze nr 1.

● Ma to ten skutek, że rozpatrywana przekątna jest wyświetlana w kolorze jasnoszarym, a jej przecięcie z paskiem usytuowanym wzdłuż głównej przekątnej jest wyświetlane w kolorze turkusowym.

Przykład 10. Wykreślanie przez wektor barw — tryb wysokiej rozdzielczości

```
program Invert;
{$i Graph.p}
begin
  HiRes;
  GraphWindow(160,48,480,152);
  Write('*');
  GotoXY(40,13);
  Write('JanB');
  repeat until KeyPressed;
  ColorTable(1,0,—1,—1);
  FillScreen(—1);
  Delay(3000);
  TextMode
end.
```

● Wykonanie programu powoduje zdefiniowanie okienka graficznego, a następnie wyprowadzenie dwóch napisów: znaku * (gwiazdka) oraz napisu JanB.

● Napis * (gwiazdka) zostaje wyprowadzony w lewym górnym narożniku ekranu, a więc nie w narożniku okienka.

● Napis JanB zostaje wyprowadzony w pobliżu środka okienka.

● Po naciśnięciu dowolnego klawisza klawiatury następuje inwersja kolorów w okienku. Kolor tła zmienia się na kolor pierwszego planu i odwrotnie.

ANIMACJA

Środki animacyjne języka Turbo Pascal sprowadzają się do procedury GetPic i PutPic. Procedury te umożliwiają zapamiętywanie obrazu znajdującego się na ekranie i późniejsze jego odtwarzanie w innym miejscu. Wywołanie procedury GetPic ma w ogólnym przypadku postać:

GetPic(Buffer,xMin,yMin,xMax,yMax)

W zapisie tym Buffer jest nazwą zmiennej dowolnego typu, a xMin, yMin, xMax i yMax są wyrażeniami typu integer. Wykonanie procedury GetPic powoduje zlokalizowanie na ekranie prostokąta o przeciwległych wierzchołkach znajdujących się w punktach xMin,yMin i xMax,yMax, a następnie przechowanie obrazu wyświetlanego w tym prostokącie w zmiennej Buffer. Wymagany rozmiar zmiennej Buffer jest uzależniony od trybu wyświetlania i rozmiarów prostokąta. Jeśli przyjąć oznaczenia:

Hor = abs(xMin — xMax) + 1
Ver = abs(yMin — yMax) + 1

to minimalny rozmiar zmiennej Buffer wyrażony w bajtach wynosi:

— dla trybów średniej rozdzielczości
((Hor + 3 div 4) * Ver * 2 + 6

— dla trybu wysokiej rozdzielczości
((Hor + 7) div 8) * Ver + 6

Sposób reprezentowania obrazu w zmiennej Buffer jest taki, że pierwsze pary bajtów stanowią nagłówek obrazu, a pozostałe zawierają dane o obrazie. Nagłówek zawiera kolejno: określenie liczby bitów niezbędnych do reprezentowania jednego piksela (2 dla średniej rozdzielczości i 1 dla wysokiej rozdzielczości), szerokość obrazu wyrażoną w pikselach (zaokrągloną do pełnych bajtów) i wysokość obrazu. Kopiowanie zawartości pamięci ekranu odbywa się wierszami, od wiersza najniższego do najwyższego i od lewej do prawej. Skrajne lewe piksele wierszy są przechowywane w najbardziej znaczących bajtach danych.

Przykład 11. Zasada zapamiętywania obrazu

```
program Save;
{$ Graph.p}
var
  Buffer : record
    Width : integer;
  Hor,Ver : integer;
    Data : array[0..199] of byte
  end;
```

```
begin
  HiRes;
  Draw(0,0,0,199,1);
  GetPic(Buffer,0,0,0,199);
  ...
end;
```

● Wykonanie procedury GetPic jest w rozpatrywanym kontekście równoważne wykonaniu instrukcji:

```
with Buffer do begin
  Width := 1;
  Hor := 1;
  Ver := 200;
  for i := 0 to 199 do
    Data[i] := $80
end
```

Po zapamiętaniu obrazu w zmiennej można go przywołać na ekran posługując się procedurą PutPic. Wywołanie tej procedury ma w ogólnym wypadku postać:

```
PutPic(Buffer,xCoord,yCoord)
```

W zapisie tym Buffer jest nazwą zmiennej, w której zapamiętano lub wygenerowano obraz, a xCoord i yCoord są wyrażeniami typu integer. Wykonanie procedury PutPic powoduje umieszczenie na ekranie prostokątnego obrazu zapamiętanego w zmiennej Buffer. Obraz jest umieszczony na ekranie w taki sposób, że jego lewy dolny narożnik zajmuje pozycję o współrzędnych (xCoord,yCoord). Ustalenie koloru pikseli odbywa się zawsze przez wektor barw.

Przykład 12. Przywoływanie obrazu na ekran

```
program Copy;
{$i Graph.p}
var
  Buffer : array[1..206] of byte;
begin
  HiRes;
  Draw(0,0,0,199,1);
  GetPic(Buffer,0,0,0,199);
  PutPic(Buffer,639,199);
  repeat until KeyPressed;
  TextMode
end.
```

● Wykonanie procedury PutPic ma w rozpatrywanym kontekście taki sam skutek jak wykonanie procedury:

```
Draw(639,0,639,199,1)
```

Charakterystyczne dla animacji nakładanie obrazów może wymagać odwołania się do koloru pikseli wyświetlanych na ekranie. Do tego celu służy funkcja GetDotColor, której wywołanie ma w ogólnym przypadku postać:

```
GetDotColor(xCoord,yCoord)
```

W zapisie tym xCoord i yCoord są wyrażeniami typu integer. Rezultatem funkcji GetDotColor jest dana typu integer. Wartością tej danej jest numer koloru piksela wyświetlanego w okienku graficznym, w punkcie o współrzędnych (xCoord,yCoord). Jeśli podane współrzędne dotyczą piksela znajdującego się poza okienkiem, to rezultatem omawianej funkcji jest dana o wartości —1.

Przykład 13. Określanie koloru piksela

```
program Color;
{$i Graph.p}
begin
  GraphColorMode;
  Palette(1);
  ColorTable(2,1,0,3);
  Plot(0,0,—1);
  Delay(5000);
  Palette(0);
  GotoXY(1,1);
Write(GetDotColor(0,0));
  repeat until KeyPressed;
  TextMode
end.
```

● Wykonanie procedury Plot powoduje wyświetlenie piksela w kolorze karmazynowym. Wynika to stąd, że wykreślanie odbywa się przez wektor barw, a więc kolor nr 0 zostaje odwzorowany na kolor nr 2.

● Po upływie 5 s kolor piksela zmienia się na czerwony.

● Wykonanie procedury Write powoduje wyprowadzenie liczby 2.

Listy

Redakcja miesięcznika INFORMATYKA

Słyszymy coraz częściej o zbliżaniu się etapu rozwoju społecznego, w którym czynnikiem dominującym będzie informacja. Tymczasem w praktyce krajowej stwierdzamy, że informacje krążą wyłącznie w zamkniętych kręgach środowiskowych i nie mogą przełamać istniejących barier zawodowych. Dotyczy to również zastosowań informatyki, które, wykorzystując nowe możliwości przekazu i rozpowszechniania informacji, powinny w pierwszej kolejności przełamywać wspomniane bariery.

Na trudności tego typu natrafiła redakcja dwutygodnika „Przegląd Hodowlany", adresowanego do środowiska zootechników. Powszechnie wiadomo, że nie tylko za granicą, ale również w naszym kraju istnieją już liczne programy komputerowe, pomagające w sposób bardziej efektywny rozwiązywać różnorodne problemy technologiczne hodowli zwierząt. Pragnąc zaspokoić szybko rosnące społeczne zapotrzebowanie, redakcja zamierza otworzyć stały dział omawiający tego rodzaju programy. Dlatego za pośrednictwem miesięcznika INFORMATYKA chcielibyśmy dotrzeć do środowiska informatyków i zaapelować o nadsyłanie pod adresem redakcji „Przeglądu Hodowlanego", 00-182 Warszawa, ul. Dubois 9 (tel. 38-91-66) wszelkich informacji na ten temat (artykuły, komunikaty, notki informacyjne).

Sądzimy, że problem dotyczy również innych dziedzin zastosowań informatyki, proponujemy więc, aby podobne działy, informujące o istniejących już programach aplikacyjnych, powstały również w innych czasopismach specjalistycznych. Sądzimy także, że będzie to jedno z najskuteczniejszych działań w kierunku unowocześnienia i wzrostu efektywności naszej gospodarki narodowej.

Redakcja dwutygodnika
„Przegląd Hodowlany"

Ośrodek Postępu Technicznego NOT
zaprasza do zwiedzania
STAŁEJ GIEŁDY ROZWIĄZAŃ TECHNICZNYCH
w Warszawie, ul. Żelazna 51/53
(dawne Zakłady Norblina)
Godziny otwarcia: 9.00—15.00
(oprócz sobót i niedziel)

Ogłoszenia ● *Ogłoszenia* ● *Ogłoszenia* ● *Ogłoszenia*

Programy instrukcje i udoskonalenia techniczne dla komputerów ATARI, COMMODORE, IBM oferuje Agencja Komputerowa 41-200 Sosnowiec, P-157, tel. 699-649.

EO/423/87

Ogłoszenia ● *Ogłoszenia* ● *Ogłoszenia* ● *Ogłoszenia*

JAN BIELECKI
Instytut Informatyki
Politechnika Warszawska

Język C — wykreślanie podstawowych obiektów graficznych

Dowolnie złożony obraz graficzny, przedstawiany na ekranie monitora, składa się z pojedynczych pikseli. Z tego powodu istotę grafiki stanowi wykreślenie piksela. Dysponując podprogramem wykreślającym piksel można konstruować podprogramy do wykreślania obiektów bardziej złożonych.

Na wydruku 1 przedstawiono program zawierający m.in. definicje: podprogramu do wykreślania punktu — **Plot** oraz podprogramu do wykreślania odcinka — **Draw**. Przed wywołaniem dowolnego z tych podprogramów musi być ustanowiony tryb graficzny. Uczyniono to za pomocą nie przedstawionej tu bliżej funkcji **mode**, wywoływanej z funkcji **HiRes**.

Za pomocą funkcji **Plot** i **Draw** sporządzono wykres przedstawiony na rys. 1. Zawarty tam napis **jb** został wykreślony za pomocą wywołań funkcji **Plot**, a odcinki linii prostych ograniczające ekran zostały wykreślone za pomocą wywołań funkcji **Draw**.

Na wydruku 2 przedstawiono program, którego wykonanie powoduje wykreślenie wzoru z rys. 2. Niespodziewany

```
#include <stdio.h>

main()
{
  HiRes();

  jbLogo();

  Delay();

  Text();
}

char *ScreenPtr;

HiRes()
{
  union{
    char *Ref;
    struct{
      int Ofs,Seg;
    } Adr;
  } Ptr;

  mode('H');

  Ptr.Adr.Seg = 0xB800;
  Ptr.Adr.Ofs = 0;

  ScreenPtr = Ptr.Ref;
}

Text()
{
  mode('L');
}

jbLogo()
{
  static char Logo[16][3] =
    { { 0xFF,0xFF,0xFF },{ 0x80,0x00,0x01 },
      { 0xBF,0xFF,0xFD },{ 0xA0,0x00,0x05 },
      { 0xA0,0x8E,0x05 },{ 0xA0,0x06,0x05 },
      { 0xA1,0xC7,0xC5 },{ 0xA0,0xC6,0x65 },
      { 0xA0,0xC6,0x65 },{ 0xAC,0xC6,0x65 },
      { 0xAC,0xCB,0xC5 },{ 0xA7,0x80,0x05 },
      { 0xA0,0x00,0x05 },{ 0xBF,0xFF,0xFD },
      { 0x80,0x00,0x01 },{ 0xFF,0xFF,0xFF } };
  char x,y,i,Byte;
  static int xLogo = 610,
             yLogo = 180;

  for(y = 0; y < 16; y++)
    for(x = 0; x < 3; x++){
      Byte = Logo[y][x];
      for(i = 0; i < 8; i++)
        if(Byte & (128 >> i))
          Plot(xLogo + x * 8 + i, yLogo + y);
    }
  Draw(0,0,639,0);
  Draw(639,0,639,199);
  Draw(639,199,0,199);
  Draw(0,199,0,0);
}

Plot(x,y)
  int x,y;
{
  ScreenPtr[ ((y & 1)  << 13) +
             ((y & -2) <<  3) +
             ((y & -2) <<  5) +
             ( x >> 3) ] ^= (128 >> (x & 7));
}

Abs(x)
  int x;
{
  return (x < 0) ? -x : x;
}

Sign(x)
  int x;
{
  return (x < 0) ? -1 : (x > 0);
}

Swap(x,y)
  int *x,*y;
{
  int Temp;

  Temp = *x;
  *x = *y;
  *y = Temp;
}

Draw(xMin,yMin,xMax,yMax)
  int xMin,yMin,xMax,yMax;
{
  int dx,dy,s1,s2,i,e,x,y;
  char Flag;

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

  if(!(dx | dy))
    Plot(x,y);
  else {
    Flag = dy > dx;
    if(Flag)
      Swap(&dx,&dy);
    e = (dy << 1) - dx;
    for(i = 1; i <= dx; i++){
      Plot(x,y);
      while(e >= 0){
        Flag ? (x += s1) : (y += s2);
        e -= dx << 1;
      }
      Flag ?  (y += s2) : (x += s1);
      e += dy << 1;
    }
  }
}

Delay()
{
  getchar();
}
```

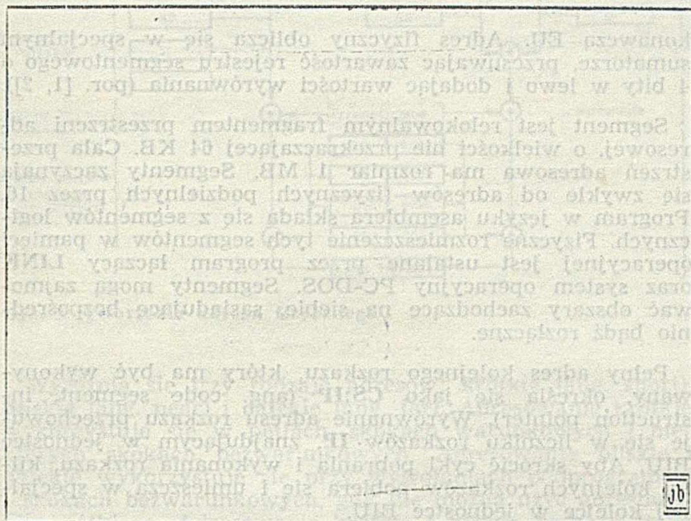
Wydruk 1. Podstawowe funkcje graficzne

```
#include <stdio.h>
#include "PlotDraw"

main()
{
  HiRes();

  DrawPattern();

  Delay();

  Text();
}

DrawPattern()
{
  int i;

  for(i = 0; i < 30; i++){
    Draw(16 * i,0,0,5 * i);
    Draw(639 - 16 * i,0,639,5 * i);
    Draw(0,199 - 5 * i,16 * i,199);
    Draw(640 - 16 * i,199,639,199 - 5 * i);
  }
}

char *ScreenPtr;

HiRes()
{
  union{
    char *Ref;
    struct{
      int Ofs,Seg;
    } Adr;
  } Ptr;

  mode('H');

  Ptr.Adr.Seg = 0xB800;
  Ptr.Adr.Ofs = 0;

  ScreenPtr = Ptr.Ref;
}

Text()
{
  mode('L');
}

Delay()
{
  getchar();
}
```

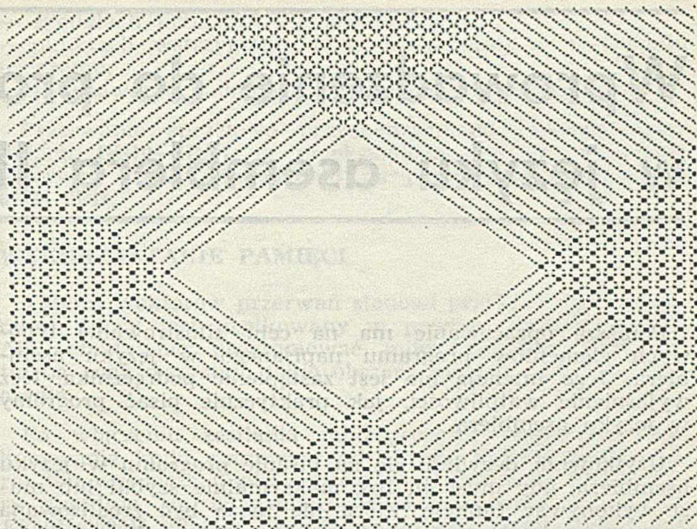
Wydruk 2. Wykreślanie wzoru

skutek przecięcia odcinków linii prostych wynika ze specjalnego zaprogramowania funkcji **Plot**, w której posłużono się operatorem różnicy symetrycznej.

Przytoczone programy dla mikrokomputera IBM PC zostały uruchomione za pomocą translatora Aztec C firmy Manx, wywołanego dla modelu z dalekimi wskazaniami (ang. long pointers). Wykorzystano fakt, iż takie wskazania są reprezentowane przez następujące bezpośrednio po sobie: przemieszczenie [1]) w segmencie **(Ofs)** i numer segmentu **(Seg)**.



Rys. 1. Wykreślanie inicjałów



Rys. 2. Wykreślanie wzoru

[1]) W terminologii stosowanej w miesięczniku Informatyka, angielskiemu terminowi offset odpowiada polski termin wyrównanie, a terminowi displacement — przemieszczenie (przypis redakcji).

LITERATURA

[1] Bielecki J.: Wprowadzenie do języka C, WNT (w druku)

[2] Bielecki J.: Język C — interpretacja standardu, WNT, 1987

[3] Bielecki J.: Uruchamianie programów w języku C, WNT (w przygotowaniu).

MIĘDZYWOJEWÓDZKA SPÓŁDZIELNIA PRACY „SIÓDEMKA”

ŁÓDŹ, AL. KOŚCIUSZKI 93,

ZAKŁAD INFORMATYKI I SYSTEMÓW KOMPUTEROWYCH

poleca po **najniższych cenach** w kraju:

— mikrokomputery 8-, 16-, 32-bitowe najwyższej jakości renomowanych firm z całego świata

— urządzenia peryferyjne:
- drukarki — również 24-igłowe i laserowe
- streamery
- napędy dyskowe 3″, 5.25″
- monitory monochromatyczne, kolorowe, EGA, HEGA, VGA
- karty rozszerzenia pamięci
- kontrolery
- dyski twarde typu Winchester 20 MB, 40 MB, 60 MB, 80 MB

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

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

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

Proponujemy również na wszelkiego rodzaju mikrokomputery programy wspomagające zarządzanie przedsiębiorstwem:

— system finansowo-księgowo-kosztowy

— system zbytu i zaopatrzenia

— system technicznego przygotowania produkcji

— system materiałowy

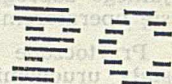
— system kadrowy i kadrowo-płacowy (również dla pracowników akordowych)

Wymienione systemy pracują w wersjach sieciowych i wielodostępnych.

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

EO/1003/87

Wprowadzenie do programowania w języku asemblera IBM PC (I)



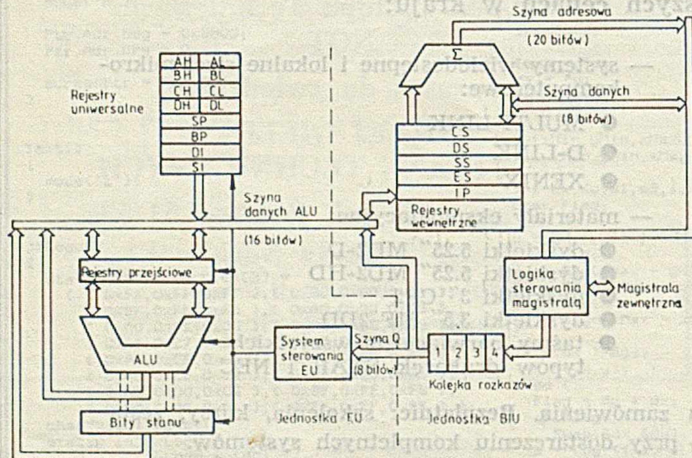
Poniższe opracowanie ma na celu zilustrowanie typowych elementów programu napisanego w języku asemblera. Jego intencją nie jest zastąpienie podręcznika, lecz wyjaśnienie wątpliwości, jak praktycznie pisać programy w języku asemblera.

Użytkownik decyduje się na pisanie programu w języku asemblera, gdy musi on być maksymalnie szybki lub musi wykonywać funkcję, która nie może być zrealizowana za pomocą konstrukcji dostępnych w językach wysokiego poziomu. Znajomość programowania w języku asemblera umożliwia także lepsze zrozumienie funkcjonowania mechanizmów spotykanych w językach wysokiego poziomu.

W pierwszej części artykułu stanowiącej rodzaj repetytorium, przedstawiono specyficzne cechy mikroprocesora Intel 8088 oraz elementy budowy systemu operacyjnego PC-DOS — wszystko znane już czytelnikom INFORMATYKI. W drugiej części omówiono zasady programowania (ilustrując je wywołaniami funkcji systemu operacyjnego PC-DOS oraz wykonywaniem operacji napisowych), a także technikę pisania kodu, który może być wywołany z programu w Basicu.

MIKROPROCESOR 8088

Mikroprocesor 8088 (rys. 1) ma dwa odzielne podukłady realizujące przetwarzanie: jednostka wykonawcza **EU** (ang. execution unit), wykonuje rozkazy, a jednostka sprzężenia z magistralą **BIU** (ang. bus interface unit) służy do komunikowania się mikroprocesora z otoczeniem. Jednostka BIU umożliwia mikroprocesorowi 8088 koordynowanie współpracy wielu jednostek wykonawczych koprocesorów 8087 (koprocesor arytmetyczny) i 8089 (koprocesor wejścia-wyjścia). Jedyna różnica między mikroprocesorami 8088 i 8086 polega właśnie na tym, że posiadają one różne jednostki sprzężenia z magistralą.



Rys. 1. Ogólna budowa mikroprocesora 8088

Jednostka EU dostarcza do jednostki sprzężenia z magistralą adres logiczny, który jest tłumaczony na adres fizyczny. Do obliczania adresu fizycznego, wykorzystuje się 16-bitowy rejestr segmentowy (ang. segment register) oraz tzw. wyrównanie (ang. offset), rozumiane jako adres względny. Adresy logiczne przedstawia się w postaci pary **segment:wyrównanie**. Do rejestrów segmentowych, zawartych w jednostce BIU należą: rejestr segmentu kodu **CS** (ang. code segment), rejestr segmentu stosu **SS** (ang. stack segment), rejestr segmentu danych **DS** (ang. data segment) i rejestr segmentu dodatkowego **ES** (ang. extra segment). Wyrównanie jest zwykle dostarczane przez jednostkę wykonawczą EU. Adres fizyczny oblicza się w specjalnym sumatorze, przesuwając zawartość rejestru segmentowego o 4 bity w lewo i dodając wartości wyrównania (por. [1, 2]).

Segment jest relokowalnym fragmentem przestrzeni adresowej, o wielkości nie przekraczającej 64 KB. Cała przestrzeń adresowa ma rozmiar 1 MB. Segmenty zaczynają się zwykle od adresów fizycznych podzielnych przez 16. Program w języku asemblera składa się z segmentów logicznych. Fizyczne rozmieszczenie tych segmentów w pamięci operacyjnej jest ustalane przez program łączący **LINK** oraz system operacyjny PC-DOS. Segmenty mogą zajmować obszary zachodzące na siebie, sąsiadujące bezpośrednio bądź rozłączne.

Pełny adres kolejnego rozkazu, który ma być wykonywany, określa się jako **CS:IP** (ang. code segment, instruction pointer). Wyrównanie adresu rozkazu przechowuje się w liczniku rozkazów **IP**, znajdującym w jednostce BIU. Aby skrócić cykl pobrania i wykonania rozkazu, kilka kolejnych rozkazów pobiera się i umieszcza w specjalnej kolejce w jednostce BIU.

Jednostka wykonawcza zawiera osiem 16-bitowych rejestrów uniwersalnych. Cztery z nich służą do operowania danymi: akumulator **AX**, rejestr bazy **BX**, rejestr **CX** pełniący najczęściej funkcje licznika (ang. counter) i rejestr danych **DX**. Górne i dolne połowy tych rejestrów funkcjonują w pełni niezależnie jako pary rejestrów 8-bitowych. Para rejestrów akumulatora jest oznaczana jako AH i AL. Analogiczne oznaczenia przyjęto dla pozostałych trzech par rejestrów.

Rejestr indeksu źródłowego **SI** (ang. source index) i rejestr indeksu wynikowego (ang. destination index) tworzą parę 16-bitowych rejestrów wykorzystywanych do indeksowania. Najczęściej używa się ich w operowaniu napisami. Aby wykonać przesyłanie lub porównanie zawartości obszarów pamięci odległych od siebie o więcej niż 64 KB, wymagane jest użycie dwóch rejestrów segmentowych. Oprócz rejestru **DS** wykorzystuje się wtedy rejestr segmentu dodatkowego **ES**, a adres, pod który przesyła się znak, wyznacza para **ES:DI**.

Wskaźnika stosu **SP** (ang. stack pointer) i wskaźnika bazy **BP** (ang. base pointer) używa się do wykonywania operacji na stosie i przekazywania parametrów do podprogramu. Przy wywoływaniu podprogramu, para rejestrów **SS:SP** pozwala na zapamiętanie na stosie adresu powrotu (stos „rośnie" w kierunku adresów malejących).

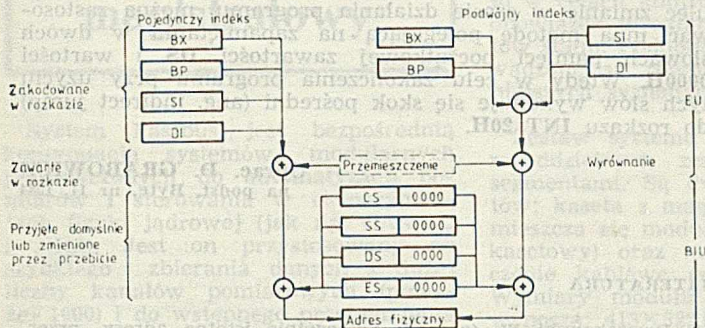
ADRESOWANIE

Jednostka wykonawcza **EU** generuje wyrównanie (określane także jako adres efektywny, ang. effective address) stosując kilkanaście różnych trybów adresowania. Wyrażenie określające wyrównanie składa się z następujących elementów: bazy (ang. base), indeksu i przemieszczenia (ang. displacement). Bazą jest zawartość rejestru **BX** lub **BP**, indeksem — zawartość rejestru **SI** lub **DI**, a przemieszczeniem — liczba 8- lub 16-bitowa ze znakiem.

Do utworzenia adresu fizycznego potrzebne jest także określenie rejestru segmentowego, a jeżeli je pominięto, to jako rejestr segmentowy przyjmuje się rejestr segmentu danych **DS**. Wyjątek stanowią sytuacje, gdy w wyrażeniu określającym wyrównanie występuje wskaźnik bazy **BP**. Wtedy jako domyślny rejestr segmentowy przyjmuje się rejestr segmentu stosu **SS**. Ponadto, przy wykonywaniu operacji napisowych (ang. string operations) jako rejestr segmentowy argumentu wynikowego przyjmuje się rejestr **ES**. Jawne określenie rejestru segmentu nie pokrywającego się z domyślnym nosi nazwę przebicia segmentu. Osiąga się to poprzedzając kod instrukcji dodatkowym prefiksem (ang. segment prefix override). Przebiciem nie można zastąpić:

- rejestru **CS** przy obliczaniu adresu następnego rozkazu do wykonania,
- rejestru **SS**, gdy używany jest rejestr **SP**,
- rejestru **ES** w odniesieniu do argumentów wynikowych przy operacjach napisowych.

Na rysunku 2 przedstawiono sposób tworzenia adresu fizycznego (por. [1]).



Rys. 2. Tworzenie adresu fizycznego

Wyróżnia się trzy rodzaje adresów: **krótkie** (ang. short), **bliskie** (ang. near) i **dalekie** (ang. far). Adresowanie krótkie wykorzystuje się w pętlach, skokach warunkowych i niektórych skokach bezwarunkowych. Adresowanie bliskie i dalekie wykorzystuje się przy wywołaniach podprogramów i skokach bezwarunkowych, gdy nie można użyć adresowania krótkiego. Adresowanie krótkie i bliskie wpływa tylko na zawartość wskaźnika rozkazów **IP** i jest zawsze względne. Adresowanie dalekie wpływa na zawartość zarówno licznika rozkazów **IP**, jak i rejestru, segmentu kodu **CS** i jest bezwzględne. Jeżeli zawartość rejestru segmentu kodu nie ulega zmianie, to mamy do czynienia z **adresowaniem wewnątrzsegmentowym** (ang. intrasegment addressing), a w przeciwnym przypadku z **adresowaniem międzysegmentowym** (ang. intersegment addresing). Bliskie wywołanie podprogramu powoduje ułożenie na szczycie stosu zawartości licznika rozkazów **IP**, a wywołanie dalekie — włożenie zawartości rejestru **CS** i licznika rozkazów **IP**. Powrót z podprogramów powoduje więc usunięcie ze szczytu stosu jednego albo dwóch słów. Brak konsekwencji w adresowaniu przy wywoływaniu podprogramu i przy powrocie z niego stanowi potencjalne źródło błędów o poważnych skutkach.

Zasady adresowania z użyciem segmentów w mikroprocesorze 8088 różnią się znacznie od tradycyjnego adresowania linearnego. Zastosowanie segmentów skraca długość rozkazów i zmniejsza rozmiary danych potrzebnych przy adresowaniu. Pozwala to na oszczędniejsze gospodarowanie pamięcią oraz przyspiesza wykonywanie programów, ponieważ wymaga mniejszej liczby odwołań do pamięci w celu pobrania rozkazu. Umożliwia także łatwą relokację segmentów logicznych. Natomiast adresowanie linearne jest lepiej dostosowane do przetwarzania danych o rozmiarach przekraczających 64 KB.

PRZERWANIA

Mikroprocesor 8088 nie odróżnia przerwań spowodowanych rozkazem INT od przerwań sprzętowych. Łącznie istnieje 256 przerwań, którym odpowiadają tzw. wektory umieszczono w tablicy począwszy od adresu 0000H::0000H. Każda pozycja tablicy (dwa kolejne słowa) stanowi adres (zapisany w postaci **CS:IP**) początku podprogramu obsługi tego przerwania (por. [3, 4]).

Podprogram obsługi przerwania jest wykonywany analogicznie jak dalekie wywołanie podprogramu, przy czym przed włożeniem na stos zawartości pary rejestrów **CS:IP** zostaje tam umieszczone dodatkowo słowo stanu procesora (rejestr **FLAGS)**. Oprócz rozkazów bliskiego i dalekiego powrotu z podprogramu, istnieje dodatkowy rozkaz powrotu z podprogramu obsługi przerwania.

Mikroprocesor 8088 wykorzystuje przerwania o numerach od **00H** do **04H** do obsługi przerwań występujących przy:
— próbie dzielenia przez zero,
— pracy krokowej,
— sygnałach przerwań niemaskowalnych,
— punktach wstrzymania (ang. breakpoint) w testowaniu programu,
— powstawaniu nadmiaru w operacjach artymetycznych.

Przerwanie sprzętowe przychodzące do mikroprocesora liniami **IRQ0—IRQ7** są przekształcane przez sterownik przerwań **8259A** odpowiednio na przerwania o numerach **08H—0FH**.

WYKORZYSTANIE PAMIĘCI

Tablica wektorów przerwań stanowi przykład obszaru pamięci, który jest użytkowany w pewien ustalony sposób. Aby efektywnie programować mikrokomputer IBM PC, trzeba znać wszystkie takie obszary mające określone znaczenie.

Po włączeniu zasilania mikroprocesor rozpoczyna wykonywanie kodu począwszy od adresu **FFFFH:0000H**. Adres ten wyznacza punkt wejścia do podsystemu BIOS, który znajduje się w pamięci ROM począwszy od adresu **FE00H:0000H** (dlatego określa się go jako ROM BIOS). Jak wiadomo, zawiera on kod umożliwiający współpracę jednostki centralnej ze wszystkimi urządzeniami znajdującymi się w standardowej konfiguracji. Odpowiednie podprogramy podsystemu ROM BIOS są dostępne dla użytkownika przez przerwania programowe o numerach od **10H** do **1FH** (por. [5, 6]). BIOS wykorzystuje także obszar pamięci zawarty między adresami **0040H:0000H** i **0040H:00FFH**. W obszarze tym znajduje się m.in. parametr określający bieżący tryb pracy monitora oraz bufor na znaki wprowadzane z klawiatury (por. [7]).

W obszasze zaczynającym się od komórki **F600H:0000H** pamięci ROM jest zawarty Basic. ROM Basic, a także dyskowe wersje Basica i niektóre funkcje systemowe wykorzystują obszar od **0050H:0000H** do **0050H:00FFH**.

W drugiej wersji systemu operacyjnego PC-DOS, pamięć od adresu **0060H:0000H** do **0060H:00FFH** jest zarezerwowana na przechowywanie różnych zmiennych dyskowych. Gdy PC-DOS jest sprowadzany do pamięci, jego pliki **IBMBIO.COM** i **IBMDOS.COM** są umieszczane bezpośrednio za tym obszarem. Funkcje systemu PC-DOS wywołuje się przez przerwania o numerach od **20H** do **27H**. DOS inicjuje urządzenia zewnętrzne, ładuje kod interpretera poleceń zapisany na pliku COMMAND.COM (do obszaru leżącego bezpośrednio nad obszarem zajętym przez plik IBMDOS) i przekazuje mu sterowanie. Część tego pliku zajmuje górny obszar pamięci, tak aby jak największy jej fragment pozostawał wolny dla programów użytkowych.

Kod zawarty w pliku COMMAND.COM jest odpowiedzialny za wykonywanie poleceń systemu operacyjnego. Część poleceń, np. **DIR** itp. (tzw. polecenia wewnętrzne), ma kod zawarty bezpośrednio w pliku COMMAND.COM. Inne polecenia np. **CHKDSK** itp. (tzw. zewnętrzne) mają kod umieszczony w samodzielnych plikach. Na przykład makroasemblerowi (zawartemu w pliku **MASM.EXE**) odpowiada polecenie zewnętrzne **MASM**.

WYKONANIE PROGRAMU

Zanim interpreter COMMAND rozpocznie wykonywanie polecenia zewnętrznego, musi najpierw utworzyć tzw. prefiks segmentu programu **PSP** (ang. program segment prefix). Prefiks ma długość 256 bajtów i zawiera dane istotne dla ładowanego programu. Idea prefiksu **PSP** wywodzi się z systemu operacyjnego CP/M. Na wydruku przedstawiono prefiks PSP zapisany w postaci odpowiednich struktur danych w języku asemblera (por. [8]). Jeżeli ładowany jest program z pliku o rozszerzeniu nazwy COM, to umieszcza się go począwszy od adresu **CS:0100H**, przy czym adres **CS:0000H** wskazuje na początek prefiksu **PSP**. Plik typu **COM** zawiera dokładnie jeden segment logiczny. Gdy rozpoczyna się wykonywanie programu, wszystkie rejestry segmentowe zawierają tę samą wartość i wskazują (przy wyrównaniu równym 0000H) na początek prefiksu segmentu programu **PSP (CS=DS=ES=SS=PSP)**. Licznik rozkazów ma wartość **IP=0100H**, a w wskaźnik stosu **SP=00FEH;** na szczycie stosu znajduje się słowo zerowe.

Jeżeli program jest zapisany na pliku o rozszerzeniu nazwy **EXE**, to może składać się z kilku segmentów logicznych, a plik zawiera dodatkowe dane wykorzystywane przy relokacji. Gdy rozpoczyna się wykonywanie programu, zawartość rejestrów **DS** i **ES** (przy wyrównaniu rów-

nym 0000H) wskazuje na początek prefiksu **PSP** (tzn. **DS=** **=ES=PSP**), natomiast adresy **CS:IP** i **SS:SP** zależą od warunków określonych na etapie asemblacji i łączenia. Zwykle zawartość rejestru segmentu kodu **CS** wskazuje na początek obszaru za prefiksem **PSP** (tzn. **CS=DS+** **+0010H**), a licznik rozkazów **IP** zawiera **0000H**.

```
;***************************
; PREFIKS SEGMENTU PROGRAMU
;***************************
                    DB   2 DUP(?)   ;int20
TOP_OF_MEMORY       DW   ?
                    DB   ?          ;ioctl
                    DB   5 DUP(?)   ;dos_jump
TERMINATE           DD   ?
CTRL_BREAK          DD   ?
CRITICAL_ERROR      DD   ?
                    DB   22 DUP(?)  ;używane przez dos
ENVIRONMENT         DW   ?
                    DB   46 DUP(?)  ;używane przez dos
FORMATTED_AREA_1    DB   16 DUP(?)
FORMATTED_AREA_2    DB   16 DUP(?)
                    DB   4 DUP(?)
UNFORMATTED_AREA    DB   128 DUP(?)
```

Fragment programu oddający strukturę prefiksu segmentu programu PSP

Aby zakończyć wykonywanie programu sprowadzanego z pliku o rozszerzeniu nazwy **COM**, używa się rozkazu bliskiego powrotu. Na szczycie stosu umieszcza się słowo zerowe, a pod adresem PSP równym **0000H** (tj. w pierwszym słowie prefiksu) kod rozkazu **INT 20H**, który powoduje zakończenie pracy i powrót sterowania do systemu operacyjnego. Skok do adresu **00000H** lub wykonanie rozkazu **INT 20H** również powoduje zakończenie wykonywania programu. Ten ostatni wariant ma tę zaletę, że nie zależy od stanu stosu.

Żadnej z podanych wyżej metod zakończenia programu nie można zastosować w wypadku programów sprowadzanych z pliku o rozszerzeniu nazwy **EXE**, bowiem wykonanie rozkazu **INT 20H** zakłada, że zawartość rejestru **CS** wskazuje na początek prefiksu segmentu programu. Założenie to wywodzi się stąd, że w pierwotnej wersji systemu operacyjnego polecenia zewnętrzne występowały jedynie w postaci plików z rozszerzeniem nazwy **COM**. Aby zakończyć wykonywanie programu typu **EXE** trzeba przekazać sterowanie do rozkazu **INT 20H** przy czym zawartość rejestru **CS** musi wskazywać na początek prefiksu **PSP**. Aby to zrealizować, zwykle umieszcza się na stosie początkową zawartość rejestru **DS** i słowo zerowe oraz wykonuje rozkaz dalekiego powrotu, korzystając z tych dwóch słów na stosie. Ponieważ zawartość stosu może ulec zmianie w czasie działania programu, można zastosować inną metodę polegającą na zapamiętaniu w dwóch słowach pamięci początkowej zawartości **DS** i wartości **0000H**. Wtedy w celu zakończenia programu przy użyciu tych słów wykonuje się skok pośredni (ang. indirect jump) do rozkazu **INT 20H**.

Oprac. D. GRABOWICZ
na podst. Byte, nr 11, 1984

LITERATURA

[1] Dworakowski W. (oprac.): Szczególnie istotne adresy, przerwania i porty IBM PC/XT. Informatyka, nr 4, str. 21—22, 1987

[2] Grabowicz R. (oprac.): Podprogramy obsługi przerwań dla IBM PC. Informatyka, nr 7—8, str. 48—50, 1986

[3] Grabowski J.: Przegląd mikroprocesorów 16-bitowych (2). Intel 8086. Informatyka, nr 4, str. 21—23, 1983

[4] Rażnowiecki A., Syfert A.: Funkcje systemu BIOS dla PC/XT, Informatyka, nr 4, str. 15—17, 1987

[5] Syfert A., Rażnowiecki A., Zalewski J.: Struktura systemu operacyjnego PC-DOS. Informatyka, nr 7—8, 1987

[6] Tabak D.: Intel 80386 i nowe mikroprocesory 32-bitowe (1). Informatyka, nr 1, str. 1—4, 1987

[7] Zalewski J. (oprac.): Porównanie systemów BIOS dla rodziny komputerów IBM PC. Informatyka, nr 4, str. 18—21, 1987

[8] Zalewski J. (oprac.): Obszar komunikacyjny systemu BIOS Informatyka, nr 7, str. 20, 1987

MIKROKOMPUTER '88

Projektowanie—Praktyka—Nauczanie

III Międzynarodowa szkoła pn. MIKROKOMPUTER '88, która odbędzie się w **Bierutowicach** w **dniach 20—23 września 1988 r.** jest organizowana przez Instytut Cybernetyki Technicznej Politechniki Wrocławskiej.

Cele szkoły: Wymiana wiedzy i doświadczeń praktycznych z zakresu grafiki komputerowej, prezentacja wyników wdrożeń, przegląd najnowszego dorobku badawczego w dziedzinie techniki mikroprocesorowej.

Tematyka obrad i sekcje: Matematyczne aspekty grafiki komputerowej, rozwiązania sprzętowe i programowe w systemach graficznych, użytkowe systemy graficzne, postępy techniki mikroprocesorowej.

Organizacja obrad obejmuje cykl wykładów wygłaszanych przez zaproszonych specjalistów oraz referaty uczestników, prezentujące dorobek badawczy z zakresu obrad Szkoły.

Zgłoszenia referatów. Streszczenia o obj. 0,5—1 str. maszynopisu należy nadsyłać do 15 lutego 1988 r. Natomiast do 20 kwietnia 1988 — pełne teksty referatów (w jęz. angielskim lub rosyjskim) o objętości do 6 str. maszynopisu. Dopuszcza się wygłaszanie referatów w jęz. polskim.

Warunki uczestnictwa: W terminie do 1 czerwca 1988 nadesłanie deklaracji i wpłacenie kwoty 20 150 zł na konto nr 93057-3418-131, NBP V O/M Wrocław. Na przekazie należy podać imię i nazwisko uczestnika oraz skrót ICT-MIKROKOMPUTER '88.

Zakład Usług Informatycznych
R. Brykajło

ul. J. Bojki 6/22, 30-612 Kraków
tel. 34-50-93

p o l e c a

ODRA-1305

- system redagowania i uruchamiania z monitorów lokalnych zadań George-2,
- interfej ODRA — AMSTRAD 6128,
- pomoc przy wdrożeniu systemów George 2 i 3.

MIKROKOMPUTERY

- system kosztorysowania,
- system płac,
- procedury dostępu do plików dBase z poziomu Pascala Turbo i MT+,
- procedury grafiki dla AMSTRAD 6128 w Pascalu Turbo i MT+.

ORAZ

- usługi w zakresie projektowania i programowania systemów przetwarzania danych.

EO/901/87

FASTBUS modularny system magistralowy

System Fastbus jest bezpośrednią kontynuacją systemów modularnych przeznaczonych do automatyzacji pomiarów i sterowania w eksperymentach fizyki jądrowej (jak np. CAMAC i NIM). Jest on przystosowany do szybkiego zbierania danych z dużej liczby kanałów pomiarowych (powyżej 1000) i do wstępnego przetworzenia informacji.

W systemie zastosowano szereg nowoczesnych rozwiązań przy zachowaniu niektórych tradycyjnych. Protokół komunikacji po magistrali stanowi w pewnym sensie etap przejściowy między systemami starszego typu, np. CAMAC, VME i najnowocześniejszymi propozycjami np. Multibus II. Główną zaletą systemu Fastbus jest bardzo duża szybkość działania umożliwiająca — przy wprowadzeniu dodatkowych mechanizmów (transmisja blokowa, potokowa) — bardzo szybką wymianę informacji. Istotną cechą systemu jest przystosowanie protokołu do budowy zestawów wielokasetowych bez dodatkowych urządzeń, jak np. magistrale gałęziowe w systemie CAMAC.

Innym ciekawym rozwiązaniem jest metoda arbitrażu zbliżona w założeniach do szeroko rozpowszechnionego systemu VME i zawierająca jednocześnie pewne mechanizmy rozbudowane w systemie Multibus II. Podobnie wygląda sprawa transmisji blokowej i potokowej oraz operacji rozgłaszania, wprowadzanych tylko w najnowszych rozwiązaniach i nie stosowanych wcześniej. Dalszą innowacją wprowadzoną w systemie Fastbus (rozbudowaną w Nubus i Multibus II), jest wykorzystanie tych samych 32 linii magistrali do przesyłania danych i adresów. Multipleksowanie linii adresów i danych pociąga za sobą podział operacji na dwie fazy, adresową i przesyłania danych, co ułatwia wprowadzenie transmisji blokowych, potokowych i rozgłaszania, a więc przesyłanie ogromnych ilości danych.

Ze względu na przeznaczenie systemu jak i jego szybkość niezbędne jest sprawdzenie poprawności transmisji. Dlatego zastosowano bardzo rozbudowaną metodę wykrywania błędów, umożliwiającą ich sprecyzowanie. Protokół pracy systemu Fastbus wymaga uzyskania potwierdzenia odbioru adresów i danych, co zwiększa prawdopodobieństwo wykrycia błędów transmisji. W systemie przewidziano również sygnalizację różnorodnych usterek (np. brak adresowania bloku, niezgromadzenie odpowiednich danych, nieprawidłowość adresowania, zbyt wolna praca urządzenia itp.) uniemożliwiających wykonanie lub zakończenie operacji. Pozwala to sprecyzować źródło błędu. Blok wykonawczy ma możliwość powrotnego wysłania nierozpoznanej części adresowej przesłanej informacji. Po zakończeniu cyklu adresowego może on również poinformować blok sterujący o niemożności przyjęcia lub wysłania danych. Ponadto w celach kontrolnych wprowadzono obowiązek wysyłania dodatkowych bitów stanu. Podobne mechanizmy wprowadzone są również w najnowocześniejszych systemach.

Zestaw systemu Fastbus składa się z oddzielnych zespołów nazywanych segmentami. Są dwa rodzaje segmentów: kaseta z magistralą, w której umieszcza się moduły systemu (segment kasetowy) oraz międzykasetowe połączenie kablowe (segmenty kablowe). Wymiary modułu są dość znaczne i wynoszą: 413×322,6×16,51 mm. W kasecie może znajdować się 26 modułów. Przewidywane są dwa typy wykonania kaset: z wymuszonym chłodzeniem powietrznym (typ A) i z chłodzeniem wodnym (typ W). Komunikacja między kasetami jest realizowana za pomocą specjalizowanych bloków. W segmencie kasetowym znajdują się dwa rodzaje urządzeń: sterujące i wykonawcze. Każda operacja na magistrali jest wywoływana przez urządzenie sterujące, które oczekuje odpowiedzi od podległych bloków wykonawczych. W wypadku konieczności porozumienia się jednego sterownika z innym sterownikiem segmentu, sterownik wywoływany reaguje analogicznie jak blok podległy. Blok podległy nie może uzyskać samodzielnego dostępu do magistrali, ale może żądać obsługi przez blok sterujący w tym samym segmencie. Bloki sterowników są wyposażone w bardziej wyrafinowany mechanizm przerwań, pozwalający im uzyskać dostęp do magistrali i przesłać po niej informacje do innego wybranego bloku.

Synchronizacja adresów i danych jest uzyskiwana parami sygnałów. W zależności od rodzaju operacji (zapis, odczyt) w cyklu danych, sterownik wysyła odpowiednie sygnały informacyjne. Po uzyskaniu potwierdzenia (o przyjęciu lub wysłaniu danych) sterownik kończy operację usuwając z magistrali wszystkie wysyłane przez siebie sygnały.

Protokół pracy magistrali przewiduje pracę asynchroniczną, z potwierdzeniem transmisji, i synchroniczną bez potwierdzenia. Podstawowym rodzajem pracy jest praca asynchroniczna. Oprócz tego, przewidziano rozbudowany system przerwań oraz zastosowano niezależny protokół transmisji szeregowej.

Ze względu na możliwość włączenia kilku bloków sterujących do jednego segmentu, w systemie Fastbus obowiązuje procedura arbitrażu. Każdemu sterownikowi przypisany jest poziom arbitrażu, przydzielony przez sterownik arbitrażowy i określający, który z bloków sterujących ubiegających się o dostęp do magistrali uzyska go jako następny. Proces arbitrażu jest dokonywany w czasie trwania poprzedniej operacji na magistrali przeprowadzanej przez jeden ze sterowników. Bezpośrednio po jej zakończeniu sterowanie przejmuje kolejny sterownik. Możliwa jest również komunikacja między sterownikiem z jednego segmentu a blokiem wykonawczym umieszczonym w innym segmencie. Jest to uzyskiwane za pomocą urządzenia nazywanego blokiem międzysegmentowym (ang. segment interconnet). Ze względu na szybkość pracy systemu przewidziano w nim ścisłe przedziały czasowe na dokonywanie tego rodzaju połączeń.

W systemie Fastbus można w pewnych zastosowaniach dołączyć komputer zewnętrzny za pomocą specjalnego bloku, tzw. sprzęgu procesorowego. Może on stanowić odrębne urządzenie, łączone za pomocą segmentów kablowych, lub może być umieszczony w kasecie zależnie od wybranej konfiguracji. Komputer powinien być tak zaprogramowany, aby dostępna mu była cała struktura systemu, tzn. by mógł on pełnić rolę najwyższego w hierarchii komputera nadrzędnego, kierującego pracą całości systemu.

Korzystając z metody adresowania geograficznego powinien on docierać do każdego urządzenia systemu indywidualnego, a przy adresowaniu logicznym — do wszystkich potrzebnych urządzeń niezależnie od ich umieszczenia w systemie.

Szybkość pracy systemu jest uzależniona od szybkości propagacji sygnałów na magistralach oraz opóźnień wynikających z przeprowadzenia operacji logicznych. Czas trwania pojedynczej operacji magistrali wynosi 0,1 μs. Istnieje możliwość zrównoleglenia pracy segmentów jak również możliwość tworzenia różnych konfiguracji. Aby umożliwić stosowanie różnorodnych elementów elektronicznych, w systemie przewidziano kilka napięć zasilających.

Mimo szeregu zalet, system Fastbus zawiera również pewne rozwiązania, z których wielu konstruktorów systemów wycofało się, np. połączenie łańcuchowe, tzw. Daisy Chain. Szczegółowe właściwości systemu są uzależnione w dużym stopniu od jego oprogramowania, które może być bardzo skomplikowane.

System Fastbus jest przeznaczony przede wszystkim do badań naukowych. W związku z tym jego właściwości technologiczne (duża płyta montażowa, specjalne elementy) i użytkowe (duży pobór mocy) ograniczają jego zastosowanie w innych dziedzinach.

K. RZYMKOWSKI

LITERATURA

[1] FASTBUS. A modular high-speed data acquisition system for high energy phisics and other application. Report ESONE/FB/ /01, Luxembourg, May 1983

[2] Rzymkowski K.: Modularny system magistralowy do automatyzacji zbierania danych eksperymentalnych — FASTBUS. Centrum Informatyki Energetyki, Warszawa, 1987.

Ada — konwencje kodowania

Adę stworzono z inicjatywy Departamentu Obrony USA, aby pomóc w przezwyciężaniu problemów związanych z tworzeniem i pielęgnowaniem oprogramowania. W tym celu również przygotowano zestaw zaleceń wspomagających tworzenie zrozumiałego, zwięzłego i jednolitego zapisu w Adzie. Ważną rzeczą jest, żeby twórcy dużych lub bardzo odpowiedzialnych programów próbowali dostosować się do norm przedstawionych w tym artykule. Twórcy oprogramowania o mniejszym stopniu ryzyka powinni również w miarę możliwości podejmować próby stosowania poniższych reguł.

WPROWADZENIE

Celem artykułu jest przedstawienie konwencji kodowania w Adzie do tworzenia szczególnie odpowiedzialnego oprogramowania. Celem tych konwencji jest ukierunkowanie implementatorów na tworzenie czytelnego, strukturalnego i jednolitego kodu. Konwencje kodowania powinny być sztywne, ale nie ograniczające. Powinny zabraniać wykorzystania niebezpiecznych właściwości języka, jednocześnie nie zmniejszając produktywności programistów. Konwencje kodowania powinny zapewnić:

1) minimalizację prawdopodobieństwa wystąpienia błędów kodowania [5];

2) zwiększenie zrozumiałości kodu, szczególnie dla osób nie związanych bezpośrednio z projektem [5];

3) uproszczenie nanoszenia mniejszych poprawek w tekście programu [5];

4) ułatwienie tworzenia dokumentacji.

W poszczególnych częściach artykułu opisano konwencje tworzenia identyfikatorów, struktury graficznej programu (wcięcia, dosunięcia, odstępy, komentarze), instrukcji sterujących, jednostek programowych Ady, jak również struktury programu. Opis dotyczy cech języka uniemożliwiających kodowanie w sposób nieczytelny i niestrukturalny.

IDENTYFIKATORY

Poszczególne punkty opisują preferowane zasady tworzenia identyfikatorów.

1) identyfikatory słów zastrzeżonych należy pisać małymi literami.

Przykład:

```
package Standard;
```

2) identyfikatory typów i podtypów należy pisać dużymi literami. Identyfikatory typów i podtypów powinny zawierać człon "_TYPE".

Przykład:

```
type COLOR_TYPE is (RED, WHITE,
BLUE);
```

3) identyfikatory zmiennych pisze się dużymi literami.

Przykład:

```
FIRSTELEMENT: RECORDPTR_
                    TYPE;
```

4) wszystkie stałe są identyfikowane małymi literami.

Przykład:

```
qsize: constant INTEGER := 10;
```

5) nazwy podprogramów, pakietów, pętli i bloków zaczynają się dużą literą, natomiast pozostała część jest pisana małymi. W nazwach wieloczłonowych, przedzielonych znakami podkreślenia, każdy człon zaczyna się dużą literą. Identyfikatory nie powinny różnić się obecnością (lub brakiem) wtrąconych znaków podkreślenia.

Przykład:

```
function Dotproduct;
procedure Insertrecord;
procedure Output_A_Character;
```

6) nazwy typów, podtypów, stałych, zmiennych, procedur, bloków, pętli, pakietów, funkcji i zadań są opisowe. Identyfikator nie może być skrótem mnemonicznym piszącego program, lecz powinien stanowić element dokumentacji dla przypadkowego czytelnika. Każdy identyfikator powinien mieć jednoznaczne znaczenie i wymowę. Literowanie identyfikatora o znanym znaczeniu powinno dać się przewidzieć. W większości wypadków długość identyfikatora zawiera się w przedziale od 6 do 16 znaków. Odstępstwem od powyższej reguły mogą być zmienne sterujące pętli i indeksy tablicy, dla których dopuszcza się identyfikatory jednoznakowe.

Przykład:

W lewej kolumnie podano poprawnie, a w prawej niepoprawnie utworzone identyfikatory.

```
STARTFREQ      STFRQ
JAIL_MENUS     JAIMS
JEWELCENTER    JECE
LAWREPORTS     LAWRE
```

7) identyfikatory wewnętrzne nie powinny przesłaniać identyfikatorów zewnętrznych o tych samych nazwach, szczególnie wtedy, gdy w zasięgu identyfikatora występuje odwołanie do identyfikatora zewnętrznego;

8) nie należy używać skrótów, chyba że skrócenie w stosunku do wyrazu pierwotnego jest znaczne. Oczywiście, znaczenie skrótów musi być jasne.

WCIĘCIA, DOSUNIĘCIA I ODSTĘPY

W celu sformatowania tekstu programu w sposób standardowy można posłużyć się formaterem lub superprinterem (ang. pretty printer). Większość środowisk programowych Ady zawiera formater kodu.

Poniższe zalecenia znajdują zastosowanie w przypadku, gdy system nie zawiera formatera.

1) zaleca się stosowanie wcięć co trzy znaki.

Przykład:

```
if SUCCESS then
   Run_Worked;
else
   Run_Failed;
end if;
type EMPLOYEE_TYPE is
   record
      AGE: INTEGER;
      SALARY : INTEGER;
   end record;
```

2) wielokrotne warunki instrukcji if powinny być zgrupowane, umieszczone w odpowiednich liniach i dosunięte tak, aby zapis był zrozumiały [3].

Przykład:

```
if (TALL and STRONG and AGGRES-
   SIVE) or
   (QUICK and GOOD_HANDS) then
   Basketball_Player;
end if;
```

3) instrukcje umieszcza się w oddzielnych liniach.

Przykład:

```
PLAYER.AGE := 25;
PLAYER.SALARY :=100000;
-- zamiast
PLAYER.AGE
:= 25;PLAYER.SALARY :=100000;
```

4) słowa zastrzeżone begin, exception i end używane w specyfikacji podprogramu powinny być dosunięte do marginesu. Kod zawarty między nimi powinien być wcięty o trzy znaki.

Przykład:

```
procedure Build_Queue is
begin
   Load_Records;
   Initialize_Ptrs;
exception
   when QUEUE_ERRORS =>
   Initialize_Queue;
   Put_Line     (ERROR_LOG,"QUEUE
   REINITIALIZED");
end Build_Queue;
```

5) inicjalizatory i terminatory pętli powinny być dosunięte do marginesu. Treść pętli powinna być wcięta o trzy znaki.

Przykład:

```
loop
   Shoot (POINTS_SCORED);
   exit when POINTS_SCORED = 2;
end loop;
while TOTAL_POINTS < max_points
loop
   Shoot (POINTS_SCORED);
   TOTAL_POINTS := TOTAL_
         POINTS + POINTS_SCORED;
end loop;
```

6) odstępy (np. spacje) [1]) powinny obejmować każdy operator (np. strzałkę, symbol przypisania). Wyjątkiem może być potęgowanie lub negacja.

[1]) W Adzie odstępami (ang. blank) są spacje (ang. space) i znaki tabulacji poziomej — przyp. red.

Przykład:

```
TOTAL_FRUIT := APPLE_COUNT +
              ORANGE_COUNT;
when BAD_APPLES =>Spoil_Case;
VOLUME_DISPLECED :=
— LENGTH**3;
```

KOMENTARZE

1) komentarze występujące w tekście programu służą zwiększeniu czytelności;

2) instrukcja else powinna zawsze zawierać komentarz wyjaśniający.

Przykład:

```
-- OKREŚLENIE GRANICY WIEKU
-- PRZY SPRZEDAŻY NAPOJÓW
-- ALKOHOLOWYCH
if AGE>= cut_off_age then
  -- cut_off_age = 18
  Allow_Purchase LIQUOR;
else -- WIEK JEST MNIEJSZY OD
  -- cut_off_age
  Disallow_Purchase;
end if;
```

3) każda jednostka kompilacji powinna zawierać nagłówek w postaci komentarza. W nagłówku należy umieścić następujące informacje:

- nazwisko projektanta,
- nazwisko programisty,
- datę wykonania projektu,
- numer segmentu kodu,
- opis celu tej jednostki kompilacji,
- dodatkowe komentarze;

Blok nagłówka umieszcza się pod nazwą pakietu lub procedury. Pole opisujące zadania w nagłówku komentarza powinno wyjaśniać związki między procedurami i funkcjami każdej jednostki kompilacji. Poniżej pokazano postać bloku nagłówka:

package Scoring is;

4) nagłówek ogólny powinien być umieszczany na początku każdego podprogramu, pakietu lub bloku w obrębie jednostki kompilacji. Ten nagłówek znajduje się jedną linię przed identyfikatorem podprogramu, pakietu lub bloku. Jego postać jest następująca:

```
Swap:
declare
  TEMP : INTEGER;
begin
  TEMP := LOWVALUE;
  LOWVALUE := HIVALUE;
  HIVALUE := TEMP;
end Swap;
```

5) wszystkie komentarze należy pisać dużymi literami;

6) komentarz powinien objaśniać względnie dużą grupę instrukcji, wstawiony segment kodu lub instrukcje wymagające specjalnej uwagi;

7) linia odstępu powinna poprzedzać komentarz i występować po komentarzu, za wyjątkiem komentarzy tworzących bloki nagłówka;

8) zaleca się stosowanie komentarzy wbudowanych;

9) instrukcje uruchomieniowe powinny być opatrzone komentarzami, co uprości późniejszą pielęgnację programu.

IMPLEMENTACJA INSTRUKCJI STERUJĄCYCH

Poniżej opisano sposób przekształcania języka PDL (ang. program design language) na Adę. W Adzie jest do dyspozycji wiele wszechstronnych konstrukcji, które upraszczają konwersję z PDL na Adę. W większości konwersja może odbywać się linia po linii.

Instrukcja if

Przykład:

```
PDL:
IF (ODEBRANO POTWIERDZENIE
ACK) THEN
  WRITE (ODEBRANO POTWIER-
  DZENIE Z MIEJSCA PRZEZNA-
  CZENIA)
ELSE IF (ODEBRANO ZAPRZECZE-
NIE NAK) THEN
  WRITE (ODEBRANO ZAPRZECZE-
  NIE Z MIEJSCA PRZEZNACZENIA)
ELSE (WYSTĄPIŁ BŁĄD)
  INCLUDE (PRZETWARZANIE BŁĘ-
  DU)
  WRITE (MELDUNEK O BŁĘDZIE)
ENDIF
Ada:
if MESSAGE = ack then
  Put_Line (PROCESS_LOG,"ODE-
  BRANO POTWIERDZENIE");
elsif MESSAGE = nak then
  Put_Line (PROCESS_LOG,"ODE-
  BRANO ZAPRZECZENIE");
else -- WYSTĄPIŁ BŁĄD
  Process_Error (MESSAGE,ERROR_
  STRING);
Put_Line (PROCESS_LOG,ERROR_
  STRING);
endif;
```

Instrukcja case

Przykład:

```
PDL:
CASE (TYP_REKORDU)
  TYP_REKORDU = AAA;
  PRZETWÓRZ REKORD TYPU
  AAA
  TYP_REKORDU = BBB:
    ZWIĘKSZ LICZNIK BBB_COUN-
    TER
  TYP_REKORDU =CCC:
    ZAPISZ REKORD NA PLIKU
  TYP_REKORDU = DDD:
    ZWIĘKSZ LICZNIK DDD_COUN-
    TER
END CASE
Ada:
case RECORDS is
  when AAA => Process_Records;
  when BBB => BBB_COUNTER :=
  BBB_COUNTER + 1;
  when DDD => DDD_COUNTER
  CESS_LOG, "ZNALEZIONO RE-
  KORD TYPU CCC");
  when others => raise ERROR;
  := DDD_COUNTER + 1;
  when others => raise ERROR;
end case;
```

Instrukcja pętli

W Adzie dopuszcza się pętle o różnorodnej strukturze. Są to:

- pętla „do while" (sprawdzanie warunku przed rozpoczęciem wykonywania pętli),
- pętla „do until" (sprawdzenie warunku na końcu pętli, dlatego pętla jest wykonywana co najmniej jeden raz),
- pętla „for" (wykonywana dla każdego elementu z dyskretnego zakresu),
- pętla bez podania schematu interacji.

Ostatnia pętla nie powinna być używana. Pierwsze trzy pętle są dopuszczalne i powinny być wykorzystywane w sposób właściwy. We wszystkich wypadkach z pętli jest tylko jedno wyjście.

Przykład:

```
PDL:
DO WHILE (BID < CUT OFF PRICE)
  INCLUDE GET_BID (BID)
END DO
Ada:
while BID.PRICE < CUT_OFF.PRICE
loop
  Get_Bid (BID.PRICE);
end loop;
```

Przykład:

```
PDL:
DO UNTIL (BIEŻĄCY_ZNAK = *)
  GET (BIEŻĄCY_ZNAK)
END DO
Ada:
loop
  Get_Char (SCREENCHAR);
  exit when SCREENCHAR = '*';
end loop;
```

Przykład:

```
PDL:
DO FOR (DLA WSZYSTKICH ZNA-
KÓW W NAPISIE 80-ZNAKOWYM)
  (OBLICZ LICZBĘ GWIAZDEK *)
END DO
Ada:
for J in 1..max_string_size
-- max_string_size = 80
  if CHARSTRING (J) = '*' then
    ASTERISK_COUNT :=
    ASTERISK_COUNT + 1;
  end if;
end loop;
```

Instrukcja goto

Instrukcja skoku goto zezwala na jawne przekazanie sterowania od tej instrukcji do instrukcji docelowej wskazanej etykietą [7]. Korzystanie z instrukcji skoku często wiąże się ze złą techniką programowania i dlatego stosowanie jej jest zabronione — z wyjątkiem uzyskania aprobaty głównego programisty.

PAKIETY

Pakiet, podobnie jak podprogram, zadanie lub jednostka rodzajowa jest jednostką programową. Pakiety umożliwiają specyfikowanie grup logicznie powiązanych bytów. W najprostszej postaci pakiety określają zbiór wspólnych obiektów i deklaracji typów. Ogólnie pakietu można użyć do specyfikowania grup powiązanych bytów zawierających podprogramy; podprogramy mogą być wywoływane spoza wnętrza pakietu, natomiast ich zawartość pozostaje ukryta i zabezpieczona przed użytkownikami zewnętrznymi [7].

1. Pakiety powinny być wykorzystywane przy specyfikowaniu grup powiązanych bytów.
2. Stałe i typy globalne odnoszące się do specyficznych zagadnień powinny znaleźć się w pakiecie.
3. Stosowanie zmiennych globalnych jest zabronione, z wyjątkiem uzyskania aprobaty głównego programisty.
4. Zmienne anonimowe są zabronione. Wszystkim typom anonimowym należy nadać określony typ.
5. Informacje nieistotne dla użytkownika pakietu powinny być utajnione przez typ prywatny z klauzulą limited.
6. Pakiety nie powinny być bezpośrednio widoczne w podprogramach; dlatego w podprogramach nie powinna występować klauzula use.
7. Specyfikacja pakietu i jego ciało są kompilowane rozłącznie. Ciało pakietu musi być kompilowane po specyfikacji.

PODPROGRAMY

Podprogram jest jednostką programową, której wykonanie odbywa się dzięki wywołaniu. Istnieją dwa rodzaje podprogramów: procedury i funkcje. Wywołanie procedury jest instrukcją, a wywołanie funkcji jest wyrażeniem udostępniającym wartość. Definicja podprogramu może występować w dwóch częściach: w deklaracji podprogramu definiującej zasady wywołania i w ciele podprogramu definiującym jego wykonanie [7].

1. Zmienne są przekazywane jako parametry.
2. Operator może być przeciążony, o ile nowa operacja jest logicznie podobna do starej operacji.

Przykład:
Operator „+" odnoszący się do dodawania może zostać przeciążony, co pozwoli na konkatenację dwóch napisów tego samego typu. Operator „**" zwykle oznaczający potęgowanie nie powinien być przeciążony do wykonywania konkatencji napisu.

3. Podprogram powinien zawierać nie więcej niż 100 wykonywalnych linii programu w Adzie.
4. Zaleca się wykorzystywanie atrybutów.
5. Posługiwanie się zdefiniowanymi pierwotnie wyjątkami w celu wykrycia niezwykłych, lecz przewidywanych sytuacji, jest złym zwyczajem, ponieważ nie ma gwarancji, że dany wyjątek został stworzony z powodu przewidywanej sytuacji. Użycie klauzuli others w obsłudze wyjątków jest zabronione.

JEDNOSTKI RODZAJOWE

Jednostka rodzajowa jest jednostką programową będącą podprogramem albo pakietem rodzajowym. Jednostka rodzajowa jest szablonem (parametryzowanym lub nie), według którego można otrzymać odpowiednie nierodzajowe podprogramy lub pakiety. O powstałych jednostkach programowych mówi się, że są konkretami pierwotnej jednostki rodzajowej [7]. Podprogramy napisane dla więcej niż jednego typu dyskretnego powinny być zamienione na jednostki rodzajowe.

ZADANIA

Zadanie jest jednostką programową składającą się ze specyfikacji i ciała. Specyfikacja zadania zaczyna się od słowa zastrzeżonego task type i deklaruje typ zadaniowy. Wartość obiektu typu zadaniowego oznacza zadanie z wejściami zadeklarowanymi w specyfikacji zadania; te wejścia są również nazywane wejściami obiektu. Wykonanie zadania jest zdefiniowane przez odpowiednie ciało zadania [7].

Zadania są bytami, których wykonanie odbywa się równolegle w określonym sensie. Można przyjąć, że każde zadanie wykonuje się na swoim własnym procesorze logicznym. Różne zadania są przetwarzane niezależnie, za wyjątkiem punktów synchronizacji.

Niektóre zadania mają wejścia. Wejście zadania może być wołane przez inne zadania. Zadanie przyjmuje wywołanie jednego ze swoich wejść instrukcją accept. Synchronizację osiąga się przez spotkanie zadania wywołującego wejście z zadaniem przyjmującym wywołanie. Niektóre wejścia mają parametry — wywołania wejść i instrukcje accept dla takich wejść są podstawowym środkiem przesyłania wartości między zadaniami [7].

1. Sprzężenia zadań powinny być jak najprostsze.
2. Ciała zadań powinny być oddzielone od ich specyfikacji.

Konwencje kodowania przedstawione w tym artykule stanowią reguły mające dopomóc w tworzeniu strukturalnego, czytelnego i jednolitego kodu. Powstały z myślą o implementatorach, mając ułatwić im tworzenie kodu standardowego i łatwego w pielęgnacji. Proponowane konwencje są na tyle elastyczne, że nie ograniczają produktywności programistów.

Oprac. M. KUC
na podst. Journal of Pascal, Ada and Modula-2, September-October, 1985

LITERATURA

[1] Barnes J. G. P.: Programming in Ada. Second Edition. Addison-Wesley, Reading (MA), 1984
[2] Cohen N. H.: Guidelines for the Selection of Identifiers in Ada Programs. Softech Inc., Huntingdon Valley (PA), 1983
[3] Gardner M. et al.: Intellimac Ada Style Manual. Intellimac Inc. Rockville (MD), 1983
[4] Haberman A. N., Perry D. E.: Ada for Experienced Programmers, Addison-Wesley, Reading (MA), 1983 (tłumaczenie polskie w druku)
[5] Krecker D. K. et al.: Software Documentation and Development Conventions. BDM Corporation, McLean (VA), 1979
[6] Nissen J., Wallis P.: Portability and Style in Ada. Press Syndicate of the University of Cambridge, New York, 1984
[7] United States Department of Defence (Ada Joint Program Office): Reference Manual for the Ada Programming Language. U.S. Government Printing Office, Washington (DC), 1983
[8] Young S. S.: An Introduction to Ada. Wiley and Sons, New York, 1983.

W skrócie • W skrócie

■ **Znany wytwórca urządzeń zewnętrznych, firma Centronics, przewidywał na rok 1986 obrót w wysokości 180 mln dolarów i drugi kolejny rok z zyskiem. Akcje przedsiębiorstwa w ciągu półtora roku zdrożały 3-krotnie i w październiku 1986 r. kosztowały 7 dolarów. Firma współpracuje (w dziedzinie mechaniki) z japońskim wytwórcą Sharp i zamierza w 1987 r. osiągnąć 15% światowego rynku drukarek o szybkości drukowania rzędu 8 stron na minutę; planuje się wyprodukowanie we Francji 3 tysięcy takich drukarek.**
Centronics zawarł umowę z IBM dotyczącą wymiany technologii i opracowuje drukarki o innych szybkościach. Model Page Printer 8 jest drukarką laserową o szybkości drukowania 8 stron na minutę i rozdzielczości 300 punktów na cal, zgodną z Proprinter, Epson FX i Diablo 630/630ESC. Ma sprzęgi Centronics, RS 232 C i IEEE, a także dodatkową pamięć 1,5 MB, umożliwiającą przechowanie strony A4. Może drukować na formacie A4, B4, na kopertach, etykietkach i foliach. Sprzedawana jest we Francji za 26 200 franków, przy czym toner na 5000 arkuszy kosztuje 490 franków, zestaw wywołujący na 30 tys. arkuszy — 2350 franków, a pas fotoprzewodzący na 15 tys. arkuszy — 580 franków. Różne emulacje można zakupić po 1700 franków.
Centronics proponuje też drukarkę mozaikową PS 220, umożliwiającą wydruk NLQ z szybkością 180 lub 45 znaków na sekundę, w cenie 5200 franków. We Francji wyroby Centronicsa sprzedają firmy Asap, Facen i Metafax. (JR)

•

■ **Minisuperkomputery wypełniają lukę między supermikrokomputerami a superkomputerami klasy Cray. Według oceny C. Ledbettera, wiceprezesa do spraw programów naukowych firmy Prime Computer Inc., największą przyszłość będą miały minisuperkomputery o szybkości obliczeniowej 2—6 milionów operacji zmiennoprzecinkowych na sekundę. Optymizm swój opiera on na obserwacji, że największe zapotrzebowanie jest na komputery do obliczeń numerycznych — inżynierskich i naukowych. Sektor ten powinien rozwijać się w następnej dekadzie w tempie 2—3-krotnie wyższym od całego przemysłu komputerowego.**
To rosnące zapotrzebowanie skłoniło wielu wytwórców do działań w tym kierunku. Firma Alliant Computers już sprzedaje takie komputery. Wiele innych, podobnie jak Prime, szybko opracowuje nowe wyroby. Niektóre z nich budują maszyny o niekonwencjonalnych architekturach, przeznaczone do określonych zastosowań. Zdaniem Ledbettera jednak największą popularność uzyskają minisuperkomputery uniwersalne o stałych cechach obliczeniowych, przydatne do wielu różnych zastosowań.
Około 40% rozkazów wykonywanych przez superkomputery dotyczy symulacji i modelowania przy wspomaganiu komputerowym projektowania i wytwarzania. Dlatego Prime zabiega o rozwój w tych dziedzinach. (JR)

W skrócie • W skrócie

Kto jest kim w IFIP

Graham Morris

Graham Morris, wiceprezydent IFIP, jest Walijczykiem, ale od kiedy wstąpił do Imperial College Uniwersytetu Londyńskiego mieszka w Anglii. Pracuje w przemyśle komputerowym od 1951 roku, stale w tej samej firmie, która po kilkakrotnych zmianach nazwy jest dzisiaj znana jako ICL. Ma ogromne doświadczenie zawodowe, które zdobył będąc programistą, analitykiem systemowym, handlowcem, planistą, szkoleniowcem i publicystą.

G. Morris jest członkiem rzeczywistym Brytyjskiego Towarzystwa Komputerowego, w którym pełnił wiele funkcji — m.in. był jego prezydentem w 1972 r. Co ciekawsze, jego młodszy brat, który też poświęcił się komputerom, pełni tę funkcję w roku bieżącym.

Związki Grahama Morrisa z IFIP datują się od jego uczestnictwa w inauguracyjnym Światowym Kongresie Komputerowym w 1959 r. G. Morris jest jednym z nielicznych, którzy uczestniczyli we wszystkich kongresach. Był również członkiem Komitetu Organizacyjnego Kongresu IFIP w 1968 roku w Edynburgu.

Pan Morris reprezentuje Brytyjskie Towarzystwo Komputerowe w IFIP od Zgromadzenia Ogólnego w Toronto, w 1977 roku, gdzie został wybrany członkiem Zarządu. Pełnił tę funkcję do 1985 roku, kiedy został wybrany wiceprezydentem. Podczas swojej pracy dla IFIP był przewodniczącym Komitetu Informacji i grupy ds. reorganizacji.

Prywatnie, Graham Morris jest zaangażowany w sprawy Kościoła i społeczności lokalnej.

M.K.
na podst. IFIP Newsletter

Ceny ogłoszeń

Od 1 stycznia br. obowiązują następujące ceny ogłoszeń publikowanych na naszych łamach:

ogłoszenia duże (zależnie od objętości):
cała strona — 35 tys. zł, 3/4 — 30 tys., 1/2 — 25 tys., 1/4 — 20 tys., 1/8 — 15 tys.
ogłoszenia drobne (zależnie od liczby słów)
jedno słowo — 30 zł

Dodatki do ceny podstawowej:
- za dodatkowy kolor (na okładce) +30%
- za zamieszczenie ogłoszenia na czwartej stronie okładki +100%
- za zamieszczenie ogłoszenia na trzeciej stronie okładki +50%

Zniżki:
- za ogłoszenie 3—5-krotne —5%
- za ogłoszenia 6—10-krotne —10%
- za ogłoszenia 11-krotne i powyżej —20%
- za artykuły reklamowe i wkładki wykonane przez zleceniodawcę —40%
- za bloki i biuletyny wykonane przez zleceniodawcę — maks. —60%

Najstarsze przedsiębiorstwo polonijne branży elektronicznej

02 658 warszawa, ul. filona 16, tel. 43 03 84, 43 75 66, 43 93 41, tlx 817218

Specjalizuje się od 5 lat w dostawach sprzętu mikrokomputerowego i oprogramowania:

√ imp 85 W+ rewelacyjnie tani mikrokomputer z 4 stanowiskami i pamięcią masową 20 MB

√ imp 85 m+ mikrokomputer z pamięcią operacyjną 0,5 MB i masową 1,5 - 21 MB, najlepszy w swojej klasie

√ pc 8/16 przystawka zmieniająca Twój 8-bitowy mikrokomputer w 16-bitowy odpowiednik PC za ułamek jego ceny

√ master najtańszy wielodostęp do PC/AT (4 stanowiska)

√ imp 86 standard XT Turbo

√ imp 8500 terminale ekranowe, odpowiedniki VT-52 i ICL 7182/2 (ODRA, ICL)

√ Oprogramowanie użytkowe na dowolne mikrokomputery

Gwarancja do 1,5 roku. Dostawa i instalacja u klienta. Pełna dokumentacja. Szkolenie techniczne. Doradztwo.

impol II posiada własne opracowania konstrukcyjne dostosowywane do potrzeb odbiorcy oraz wyspecjalizowane biuro programowania. Serwis naszego sprzętu prowadzą punkty w Warszawie, Bydgoszczy, Gliwicach, Szczecinie i Krakowie.

EO/544/87

Metakody znaków alfanumerycznych

Obecnie w większości mikro- i minikomputerów powszechnie stosowany jest kod zachowujący standard ASCII w zakresie wartości od 32 do 127 oraz w podstawowym zestawie znaków sterujących. Pozostałe wartości kodu, jeżeli są wykorzystywane (szczególnie wartości powyżej 128), w różnych typach komputerów oznaczają różne symbole. Zgodnie z tradycją przyjmuje się, że standardem jest tablica kodów Latin-1 stosowana w IBM PC. W tablicy tej istnieje kilkanaście znaków alfabetów narodowych — dokładniej zachodnioeuropejskich. Brakuje w niej znaków alfabetów słowiańskich oraz cyrylicy. Należy jednak nadmienić, że oprócz tego istnieje tablica kodów Latin-2 zawierająca dokładnie przemieszane znaki polskie i naszych sąsiadów oraz tablica Cyrillic zawierająca znaki alfabetu rosyjskiego, bułgarskiego, macedońskiego itp. Tablice te jednak nie są szerzej wykorzystywane.

Brak standardu wartości kodów znaków narodowych powoduje, że w produkowanych w Polsce mikrokomputerach — Elwro 800 i Mazowia oraz innych — przyjmowane są różne rozwiązania, które mają stać się w przyszłości normą. Przeprowadzenie uzgodnień będzie trudne, kiedy ruszy już produkcja i wszelkie zmiany w rastrach znaków nie będą już sprawą prostą — przynajmniej organizacyjnie.

Podobnie wygląda sprawa z drukarkami. Sprowadzane z zagranicy z różnych firm mają przeważnie dodatkowe znaki narodowe zamiast innych, podobno mniej potrzebnych. Oczywiście nie mają one znaków alfabetu polskiego, ale takich znaków nie mają również drukarki produkowane w Błoniu. Równocześnie większość lepszych drukarek, pracując w trybie graficznym lub z programowalnymi rastrami znaków, może drukować dowolne symbole. Lokalne rozwiązania już istnieją — niestety każde różni się czymś od innego, równie oryginalnego.

Dodatkowych problemów przysparzają różnego rodzaju edytory tekstów oraz inne programy, z różnym skutkiem przystosowywane do akceptacji dodatkowych znaków języka polskiego. Odbywa się to kosztem innych znaków o kodach z zakresu 32..127 lub też kosztem dodatkowych znaków innych alfabetów (o kodach powyżej wartości 128). W rezultacie albo nie można napisać tekstu programu z polskimi identyfikatorami oraz nawiasami kwadratowymi, albo nie można wpisać nazwiska z literą ü itd. Adaptacja tych programów do potrzeb polskich wiąże się też z koniecznością przeprogramowania obsługi klawiatury, aby można wprowadzać te dodatkowe znaki, oraz — ewentualnie — zmiany sposobu współpracy z ekranem, aby można je wyświetlić.

Reasumując, stan ten można określić mianem „Big Chaos", czyli „Wielki Bałagan". Co więcej, argumenty przytoczone za danym rozwiązaniem, a przeciw wszystkim innym, są dla każdego rozwiązania tak mocne, że uzyskanie stanowisk zbieżnych staje się niemożliwe.

Negatywne właściwości kodu ASCII (ISO)

Kod ASCII ma wiele niekorzystnych właściwości z polskiego (i nie tylko) punktu widzenia.

Rozmieszczenie znaków o kodach w zakresie 0..127 musi pozostać niezmienione. Warunek ten wynika z faktu, że suma alfabetów stosowanych w różnych językach programowania i systemach pokrywa cały ten zakres.

Rozmieszczenie znaków semigraficznych o kodach w zakresie 179..222 musi pozostać niezmienione. Warunek ten wynika z wykorzystywania tych znaków przez oprogramowanie IBM PC do rysowania ramek. Niektóre drukarki zostały już przystosowane do specjalnego pogrubiania i ciągłego rysowania tych ramek.

Wprowadzając dodatkowe znaki polskich liter: ą, ć, ę, ł, ń, ó, ś, ź, ż, kierujemy się nieco złudną pod względem praktycznym zasadą, aby były one ułożone alfabetycznie, oraz aby kod dużej litery różnił się od kodu małej o 16, a najlepiej o 32. Propozycja wprowadzania, przechowywania i wyświetlania oraz drukowania polskich znaków jako pary dwóch znaków (np. ą drukowane przez nakładanie przecinka na a) jest nie do przyjęcia, bowiem otrzymujemy wtedy na ekranie tekst całkiem nieczytelny, a w przechowywaniu jest on nieokreślonej długości. Co więcej, nie każda drukarka lubi być tak traktowana. Moim zdaniem, ten jeden pomysł na pewno musi być skazany na zagładę.

Dotychczasowy układ kodu ASCII (ISO) nie zapewnia możliwości prostego sortowania leksykograficznego bez uwzględniania podziału na duże i małe litery (tak jak w skorowidzach, leksykonach czy słownikach). Dodanie znaków liter dodatkowych jeszcze bardziej utrudnia to zadanie oraz inne operacje przetwarzania tekstu — zmianę liter dużych na małe i odwrotnie, konwersje znaków o kodach większych od 127 na znaki dopuszczalne w prostszych drukarkach.

Obecnie istnieją już tysiące tekstów zapisanych na nośnikach magnetycznych z wykorzystaniem różnego rodzaju edytorów tekstów oraz innych programów, np. do obsługi baz danych. Już teraz lub w najbliższej przyszłości będzie istniała potrzeba skopiowania tych tekstów na inny typ komputera, z innym typem drukarki oraz wykorzystywanie ich w innym rodzaju programu. Co więcej, większość nowych tekstów powinna być zapisana z pełnym wykorzystaniem zestawów liter alfabetów narodowych, a często równocześnie w kilku alfabetach (np. słowniki, dokumentacje, instrukcje użytkowania).

Reasumując, pilną potrzebą jest znalezienie metody mogącej pogodzić różne sposoby wykorzystania kodu ASCII oraz pozwalającej rozwiązać postawione problemy.

Propozycja rozwiązania

Moim zdaniem jedynym rozwiązaniem jest zaprojektowanie metakodu znaków zawierającego zestawy wszystkich możliwych symboli znaków z kodu ASCII oraz różnych alfabetów narodowych. Zasada jego wykorzystania polegałaby na konwersji istniejącego tekstu na zapis w metakodzie, z którego można by następnie dokonywać konwersji na nową, żądaną postać zapisu. Inaczej mówiąc, przekształcenie tekstu z jednym zestawem znaków określonym przez program X na tekst z zestawem znaków wymaganych przez program Y odbywałoby się pośrednio za pomocą zapisu w metakodzie. Jednocześnie, przy odpowiednim określeniu sekwencji porządkujących kody znaków w metakodzie, powinno być możliwe przetwarzanie tych tekstów polegające na poprawnym sortowaniu, zmianie wielkości liter czy transkrypcji pomiędzy alfabetami różnych typów.

Postać metakodu dla spełnienia tych wymagań musi opisywać każdy symbol znaku dwoma wartościami — bajtem oznaczającym typ zestawu znaków oraz bajtem kodu znaku. Jednocześnie, w celu ułatwienia kopiowania i przeglądania tekstów zapisanych w metakodzie, wszystkie wartości kodów powinny być większe od 31. W sumie jest do dyspozycji 50 176 różnych wartości kodów. Oczywiście nie wszystkie muszą zostać wykorzystane.

Teksty zapisane w metakodzie mogą być przechowywane w plikach o organizacji strumieniowej lub rekordowej. Dla ułatwienia operowania tymi tekstami, kody zestawów mogą być zapisane w innym pliku niż kody znaków — lecz o tej samej organizacji. W ten sposób teksty po konwersji na metakod zachowują swoją długość oraz mogą zachować dotychczasową organizację pliku.

Zasady opracowania metakodu

Przy opracowaniu metakodu przyjąłem następujące zasady:

- minimalna i prosta konwersja z kodu ASCII,

- brak w metakodzie wartości mniejszych od 32,
- uporządkowanie alfabetyczne liter bez względu na ich wielkość (litery duże i małe są tożsame),
- uporządkowanie dodatkowych liter alfabetu tak, aby w sortowaniu miały kod odpowiadających im liter zestawu podstawowego lub kod właściwie je porządkujący (dotyczy to przede wszystkim alfabetu polskiego),
- uwzględnienie symbolu waluty w zestawach kodów znaków dodatkowych z alfabetów narodowych,
- kod spacji powinien być mniejszy, a kod podkreślenia większy od kodów liter, co pozwala uzyskać uporządkowanie: A A, AAA, A_A,
- uwzględnienie praktycznie wszystkich alfabetów europejskich.

Tabela 1. Metakody podstawowego zestawu znaków. Wyjaśnienia: a) zestawowi znaków sterujących (funkcyjnych) odpowiadają w metakodzie kody odpowiednich liter; b) sposób przypisania znakom sterującym drukarką wartości kodów musi być przedyskutowany po analizie porównawczej najczęściej spotykanych drukarek; c) symbolami sterującymi są symbole o wartościach 0..31 z IBM PC; d) separatorami są cyfry oraz wszystkie znaki symboliczne; e) każdej literze przypisano dwa opisy w metakodzie — np. A = (64,65) oraz (64,66), a = (65,65) oraz (65,66); f) symbol waluty $ lub £ jest traktowany też jako dodatkowa litera — ostatnia w alfabecie (p. tabela 2).

Zestaw znaków	Kod ASCII	Metakod	
		Kod zestawu	Kod znaku
Sterujące	0..31	32	64..95
Sterujące drukarką	0..127	34	32..255
Symbole sterujące	0..31	40	32..63
Separatory, cyfry	32..64 92..96 123..127	42	32..64 191..196 223..227
Numeryczne symbole liczb	43..59 68..69 100..101	48	43..59 68..69 100..101
Litery duże Litery małe	65..90 97..122	64 65	65..118 65..118
Symbol waluty	36	64 65	131 131
Alfanumeryczne	32..127	68	32..127
Spacje		96	dowolne
Puste		98	dowolne

Propozycję metakodu podano w tabelach. Na początku należy zaznaczyć, że nie jest to propozycja ostateczna. Konieczne jest bowiem dokładne przeanalizowanie funkcjonalności rozwiązania, szczególnie pod względem prostoty konwersji z metakodu i na metakod. Z drugiej jednak strony, już teraz możliwe jest przygotowanie odpowiednich programów konwersji i korzystanie z tej metody transformacji tekstów. Trzeba bowiem zaznaczyć, że pliki z zapisem tekstu w metakodzie powinny być traktowane jako tymczasowe — tylko na okres dokonywania konwersji lub przekazywania tekstu do innego ośrodka.

Można teraz skomentować tę propozycję. Przede wszystkim, przypisanie każdej literze dwóch kolejnych wartości kodów przy rozróżnieniu liter dużych od małych kodem zestawu ułatwia sortowanie alfabetyczne. Druga wartość kodu litery jest przeznaczona dla znaków liter dodatkowych — tabela 2. Przy okazji okazuje się, że tylko w języku polskim dodatkowe litery są umieszczone pomiędzy literami podstawowego alfabetu łacińskiego, tzn. **ą** między **a** i **b**, **ę** między **e** i **f**, itd., natomiast w innych alfabetach są utożsamiane z nimi — **ä** z **a**, **ü** z **u**, lub są tylko literami z oznaczeniem akcentu **é**, **è**, **ê**. W alfabecie duńskim dodatkowe litery są umieszczone na końcu.

Jak wynika z tabel, podstawowe uporządkowanie kodu ASCII zostało zachowane (wartości kodów separatorów i cyfr oraz ich uporządkowanie), zmieniono jedynie kody znaków: lewy nawias kwadratowy, ukośnik, prawy nawias kwadratowy, grot, podkreślenie, apostrof, lewy nawias klamrowy, kreska pionowa, prawy nawias klamrowy, tylda, ze względu na zmiany w kodach liter. Kody znaków sterujących odpowiadają uporządkowaniu liter dużych zgodnie z właściwością funkcjonowania znaku klawisza CTRL.

Zestaw numeryczny o kodzie 48 obejmuje wszystkie kody znaków koniecznych do zapisania wartości liczbowych wraz ze znakami specjalnymi. Zestaw ten jest przeznaczony do konwersji plików przechowujących tylko wartości liczbowe — bez potrzeby tworzenia skojarzonego z nimi pliku z kodami zestawów. Podobnie, zestaw alfanumeryczny o kodzie 68 obejmuje wszystkie kody podstawowego zestawu liter i typowych znaków interpunkcyjnych.

W zestawach kodów przewidziano kody alfabetów innych niż łacińskie, tj. greckiego, rosyjskiego (bułgarskiego), macedońskiego i in. W tym wypadku kody poszczególnych znaków są przypisane zgodnie z ich uporządkowaniem alfabetycznym i zgodnie z tożsamością liter dużych i małych. Takie uporządkowanie pozwala bowiem i dla tych alfabetów wykorzystać właściwości metakodu przez standardowy program sortujący.

W zestawach metakodów przewidziano też zestaw kodów znaków — semigraficznych, matematycznych (określony na

Tablica 2. Metakody alfabetów narodowych, kod zestawu dla liter dużych jest o 1 mniejszy od kodu zestawu dla liter małych

Kod zestawu — alfabet	Francuski	Niemiecki	Duński	Polski	Czeski	Słowacki
Kod znaku	130	135	141	161	163	165
65	á	ä			á	ä
66	â			ą		á
69					č	č
70				ı		
72	é					
73	è				é	é
74	ê			ę	ě	
81					í	í
87						
88				ł		
91						
92				ń	ň	ň
93	oe	ö				
94				ó		
99					ř	
100						
101		ß			š	š
102				ś		
105	ù	ü				
106	û				ů	
113					ú	ú
115					ž	ž
116				ź		
117			oe			
118				ż		
119			Ø			
121			ae			
131	Fr	Dm	Ør	zł	kr	kr
132						

podstawie układu przyjętego w IMB PC) oraz zestaw standardowych kodów sterujących dla drukarek. Zestaw ten powinien umożliwiać transformację kodów występujących przeważnie po kodzie ESC (27), a sterujących pracą drukarki.

Działania na zapisie w metakodzie

Podstawowym przeznaczeniem metakodu jest ujednolicenie sposobu konwersji tekstów z jednej postaci na inną. Przyjęcie jako wspólnego odniesienia zapisu tekstu w metakodzie pozwala ograniczyć liczbę programów konwersji. Wystarczy bowiem dla każdego zmodyfikowanego edytora tekstów napisać program konwersji na zapis w metakodzie i program konwersji z zapisu w metakodzie. Co więcej, program ten może napisać zespół lub osoba najlepiej znająca dany produkt bez potrzeby poznawania postaci zapisu stosowanej w innych programach.

Jednocześnie zgodnie z pierwotnymi postulatami, zapis w metakodzie pozwala na opracowanie efektywnego programu sortującego teksty zgodnie z przyjętym porządkiem alfabetycznym. Wystarczy bowiem zastosować jedną ze znanych metod sortowania rekordów w pliku — stosując je dla pliku z zapisem w metakodzie. Należy oczywiście pamiętać o równoczesnym przestawianiu rekordów w pliku z kodami zestawów. Ten program sortowania może być wykorzystywany w różnych programach operujących bazami danych, gdyż prostsze jest dokonanie konwersji pliku na postać w metakodzie, posortowanie go i następnie powtórne dokonanie konwersji na postać używaną w programie, niż pisanie dość trudnego algorytmicznie i niepowtarzalnego w tym wypadku programu sortowania na podstawie kodu ASCII z kodami liter dodatkowych. Z innych zastosowań można wymienić programy:

- zmiany wielkości liter z dużych na małe lub odwrotnie przez zwiększenie lub zmniejszenie o jeden wartości kodu zestawu liter,
- transkrypcji alfabetu rosyjskiego na alfabet łaciński, gdy nie ma odpowiedniej drukarki z cyrylicą,
- zmiany wszystkich kodów znaków nie występujących w podstawowym zestawie znaków (od wartości 127) przez zamianę ich na kody znaków liter do nich zbliżonych (wystarczy tylko zmienić — na przykład — kod zestawu 160 na 64, a 161 na 65) lub na kod znaku spacji czy braku informacji (wykorzystuje się wtedy kod zestawu 96 lub 93); program taki jest użyteczny, gdy trzeba wydrukować tekst na drukarce D-100.

* * *

Zaprezentowane rozwiązanie problemów transformacji tekstów przy przenoszeniu ich między różnymi komputerami i oprogramowaniem jest tylko propozycją wymagającą dalszych dyskusji. Nie ulega jedynie wątpliwości, że tego rodzaju ustalenie powinno zapaść jak najszybciej, jeżeli nie chcemy poruszać się w najbliższej przyszłości w gąszczu różnych programów — użytecznych tylko w ściśle konkretnych sytuacjach. Powtarzam, że już teraz można dokonywać prób z metakodem doskonaląc to rozwiązanie i poszukując dla niego również innych zastosowań.

Pozostaje jeszcze do rozstrzygnięcia sprawa rozmieszczenia polskich znaków na klawiaturze, rozwiązanie ich wyświetlania na ekranie oraz sposobu drukowania na drukarce. Ze względów użytkowych istotne jest tylko wspólne rozstrzygnięcie pierwszego problemu. Pozostałe dwa mogą być rozwiązane różnie w różnych instalacjach.

Obecnie w dyskusjach rysują się dwie metody wprowadzania i układu klawiatury polskich liter. Pierwsza z nich zwana umownie „klawiaturą programisty" polega na wprowadzaniu polskich (i innych dodatkowych) znaków klawiszami ALT+litera_podobna oraz ALT+SHIFT+litera_podobna, np.: ą = **ALT+a**, Ą = **ALT+SHIFT+a** oraz ż = **ALT+x**. Rozwiązanie to zachowuje dotychczasowy zestaw znaków na klawiaturze, do którego jest już przyzwyczajona większość programistów. Drugą metodę można nazwać „klawiaturą maszynistki" z układem QWERTY, odpowiadającym normie układu klawiszy maszyny do pisania. W tym rozwiązaniu część dotychczas istniejących znaków zostaje zastąpiona znakami polskimi lub też dla tych dodatkowych znaków wprowadza się dodatkowe klawisze (jak na przykład w klawiaturze mikrokomputera Elwro Junior). W każdym razie i tutaj należy dążyć do ustalenia co najwyżej dwóch norm — dla programisty i dla maszynistek (przepraszam, chyba komputerzystek?).

WACŁAW ISZKOWSKI

WARUNKI PRENUMERATY NA 1988 R.

Prenumeratorzy zbiorowi — jednostki gospodarki uspołecznionej, instytucje i organizacje społeczne zamawiają prenumeratę dokonując wpłaty wyłącznie na blankiecie „wpłata-zamówienie" (jest to „polecenie przelewu" rozszerzone dla potrzeb Wydawnictwa o część dotyczącą zamówienia). Blankiety te będą dostarczane dotychczasowym prenumeratorom przez Zakład Kolportażu. Nowi prenumeratorzy otrzymają je po zgłoszeniu zapotrzebowania (pisemnie lub telefonicznie) w Zakładzie Kolportażu.

Prenumeratorzy indywidualni — osoby fizyczne zamawiają prenumeratę dokonując wpłaty w UPT lub NBP na 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 Oddział 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 ulgowej jest taki sam jak prenumeraty indywidualnej. W prenumeracie ulgowej można zamówić tylko po 1 egzemplarzu każdego czasopisma.

Uwaga! Miesięcznik „Aura" może być zamawiany w prenumeracie ulgowej również przez uczniów szkół ogólnokształcących.

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.

Wpłaty na prenumeratę przyjmowane są w terminach:
- do 10 listopada na każdy kwartał, I i II półrocze oraz cały rok następny,
- do 28 lutego na II, III i IV kwartał oraz II półrocze,
- do 31 maja na III i IV kwartał oraz II półrocze,
- do 31 sierpnia na IV kwartał.

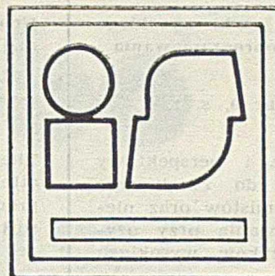
Zmiany w prenumeracie można zgłaszać pisemnie tylko w wyżej wymienionych terminach.

Informacji o prenumeracie udziela — Zakład Kolportażu Wydawnictwa NOT SIGMA (ul. Bartycka 20, 00-716 Warszawa), skr. poczt. 1004, 00-950 Warszawa, tel. 40-00-21 wew. 248, 249, 293, 297, 299 lub 40-30-86 i 40-35-89.

Egzemplarze archiwalne czasopism — można nabyć za gotówkę w Klubie Prasy Technicznej, Warszawa ul. Mazowiecka 12 (tel. 27-43-65), lub zamówić pisemnie. Zamówienia na egzemplarze archiwalne czasopism przyjmuje: Zakład Kolportażu, Dział Handlowy, 00-950 Warszawa, skr. pocz.t 1004 (tel. 40-37-31), na rachunek dla instytucji lub za zaliczeniem pocztowym dla osób fizycznych.

Cena prenumeraty wg cennika na 1988 rok					
kwartalna		półroczna		roczna	
normalna	ulgowa	normalna	ulgowa	normalna	ulgowa
600,—	150,—	1200,—	300,—	2400,—	600,—

INTERSOFT



Sp. z o.o.

Al. Ujazdowskie 18 m. 14, 00-478 Warszawa
Punkt handlowy ul. Zamenhofa 4 m. 32, 00-160 Warszawa, tel.: 316-322 lub 280-176

oferuje:

WYBRANE POZYCJE DOKUMENTACJI W JĘZYKU POLSKIM DO KOMPUTERÓW IBM

— Przewodnik programisty IBM
— Sidekick
— Wstęp do grafiki komputerowej (Basic, Turbo Pascal, TG, aut. J. Bielecki) w komplecie 4 dyskietki z przykładami i oprogramowaniem pomocniczym
— Autocad (opr. Computex)
— Symphony (opr. Computex, tylko łącznie z programem)
— Lotus 1-2-3
— Framework
— System operacyjny DOS 2,0 (2 tomy, opr. Computex)
— System operacyjny DOS 3,1 (1 tom)
x — System operacyjny DOS 3,2 (3 tomy)
x — GW-Basic, interpreter (opr. Computex)
— GW-Basic, kompilator (opr. Computex)
x — GW-Basic, kompilacja i konsolidacja programów (opr. Computex)
x — Język "C", opis (opr. Computex)
— Język "C", kompilator (Lattice, opr. Computex)
— Język "C" (aut. J. Bielecki)
— Fortran 77, opis języka (opr. Computex)
— Fortran 77 (aut. J. Bielecki)
— Fortran 77, kompilator (opr. Computex)
— Fortran 77, podstawowa biblioteka numeryczna (opr. Computex)
— Agraph, biblioteka graficzna dla języków Fortran i Pascal (opr. Computex)
— Turbo Graphics (opr. Computex)
— Turbo Pascal, podręcznik użytkownika (opr. Computex)
— Turbo Pascal (aut. J. Bielecki)
— dBase III, programowanie i opis komend (2 tomy)
— Clipper, kompilator do dBase III
x — dBase III, opis komend z przykładami
— Przewodnik zaawansowanego programisty do dBase II/III
— dBase III+, programowanie
— dBase III+, samouczek
x — dBase III+, zastosowania
x — Instrukcja obslugi PC1512 (2 tomy)
x — Instrukcja obslugi dysku twardego do PC1512
x — PC1512, opis techniczny
x — Instrukcja do drukarki Star Gemini 160
x — Chi writer
x — PC writer
— Turbo "C"

Uwaga: pozycje oznaczone "x" są w przygotowaniu. Firma INTERSOFT jest oficjalnym dostawcą oprogramowania i dokumentacji firmy Computex

WYBRANE POZYCJE DOKUMENTACJI W JĘZYKU POLSKIM DO KOMPUTERÓW AMSTRAD

— Instrukcja do komputera CPC 464
— Instrukcja do komputera CPC 6128
— Instrukcja do komputerów PCW 8256/8512, wstęp
— Instrukcja do komputerów PCW 8256/8512, Locoscript
— Instrukcja do komputerów PCW 8256/8512, CP/M
— Instrukcja do komputerów PCW 8256/8512, Mallard Basic
— CPC Basic
— Systemy operacyjne CPC
— Intern 664/6128
— DDI-1 Firmware
— CPC 464 Firmware
x — Ins&Outs
— AutoCad (CPC, PCW)
— Instrukcja do drukarki DMP-2000
— Wordstar
— Protext
— dBase II
x — Praktyczne zastosowania dBase II
— Dr Draw
— Pascal MT+
— Turbo Pascal
— Profi Painter
x — C Basic (3 tomy)
— Fortran 80

Uwaga: pozycje oznaczone "x" są w przygotowaniu.

EO/792/87

Goos G.: Nowoczesne języki wysokiego poziomu a produkcja oprogramowania

INFORMATYKA 1987, nr 9, s. 1

Historia, stan obecny i perspektywy działań zmierzających do zwiększenia produktywności programistów oraz niezawodności oprogramowania przy użyciu współczesnych języków wysokiego poziomu.

Goos G.: Modern high level languages and software development

INFORMATYKA 1987, No. 9, p. 1

The past, present and future of attempts to increase programmer's productivity and the reliability of software within contemporary high level languages.

Goos G.: Moderne höhere Programmiersprachen und Entwicklung der Software

INFORMATYKA 1987, Nr. 9, S. 1

Geschichte, Gegenwart und Zukunft der Bestrebungen nach Steigerung der Programmiererleistung und der Softwarezuverlässigkeit mit Anwendung der heutigen höheren Programmiersprachen.

Zielczyński P.: MULTICOMP — system rozwiązywania zadań metodą przeszukiwania drzew (1)

INFORMATYKA 1987, nr 9, s. 5

Pierwsza część artykułu na temat języka Multicomp przeznaczonego do rozwiązywania zadań metodą przeszukiwania drzew. Omówiono metody przedstawiania zadań w postaci drzew oraz podstawowe strategie przeszukiwania drzew.

Zielczyński P.: MULTICOMP — a task solving system through tree search (1)

INFORMATYKA 1987, No. 9, p. 5

First part of the paper on the Multicomp language for task solving through tree search. Methods for task presentation in tree form and basic strategies for tree search are discussed.

Zielczyński P.: MULTICOMP — ein System für Aufgabenlösung mit Anwendung von Baumsuchverfahren (1)

INFORMATYKA 1987, Nr. 9, S. 5

Erster Teil des Artikels über die Multicomp-Sprache für Aufgabenlösung mit Anwendung von Baumsuchverfahren. Es wurden Methoden der Aufgabenvorstellung in der Baumform sowie Grundstrategien für Baumsuchverfahren besprochen.

Ossowska S.: Wordstar 3.30 — zasady działania i sposób użytkowania (1)

INFORMATYKA 1987, nr 9, s. 9

Pierwsza część charakterystyki zasad działania i sposobu użytkowania pakietu Wordstar 3.30.

Ossowska S.: Wordstar 3.30 — operating principles and handling (1)

INFORMATYKA 1987, No. 9, p. 9

First part of Wordstar 3.30 characteristics, which presents operating principles and handling of the package.

Ossowska S.: Wordstar 3.30 — Betriebsgrundsätze und Handhabung (1)

INFORMATYKA 1987, Nr. 9, S. 9

Erster Teil einer Charakteristik von Grundsätzen und Handhabung des Wordstar 3.30 — Programmpakets.

Grudziński W.: Zastosowanie Prologu w bazach danych (2)

INFORMATYKA 1987, nr 9, s. 11

Druga część artykułu, zawierająca omówienie sposobów wykorzystania języka Prolog w systemach baz danych.

Grudziński W.: Prolog application in data base systems (2)

INFORMATYKA 1987, No. 9, p. 11

Second part of the paper, which contains discussion of Prolog language application in data base systems.

Grudziński W.: Prolog-Anwendung in Datenbanksystemen (2)

INFORMATYKA 1987, Nr. 9, S. 11

Zweiter Teil des Artikels, der eine Besprechung von Lösungen der Prolog-Anwendung in Datenbanksystemen umfasst.

Bielecki J.: Podstawy grafiki w języku Turbo Pascal (2)

INFORMATYKA 1987, nr 9, s. 14

Druga część artykułu na temat grafiki w języku Turbo Pascal zawierająca omówienie zasad oraz przykłady wykreślania punktów, odcinków i prostokątów, łuków i okręgów, wypełniania obszarów oraz animacji.

Bielecki J.: Graphics essentials in Turbo Pascal (2)

INFORMATYKA 1987, No. 9, p. 14

Second part of the paper on graphics in Turbo Pascal, which contains discussion of principles and examples for drawing points, segments and rectangles, arces and circules, area filling and animation.

Bielecki J.: Grafikgrundlagen in Turbo Pascal (2)

INFORMATYKA 1987, Nr. 9, S. 14

Zweiter Teil des Artikels über Grafik in Turbo Pascal, der eine Besprechung von Grundlagen und Beispiele der Darstellung von Punkten, Abschnitten und Rechtecken, Bogen und Umkreisen, Bereichvollfüllen und Animation umfasst.

Bielecki J.: Język C — wykreślanie podstawowych obiektów graficznych

INFORMATKA 1987, nr 9, s. 18

Podprogramy oraz przykłady wykreślania podstawowych obiektów graficznych w języku C na komputerze IBM PC.

Bielecki J.: C language — basic graphic objects drawing

INFORMATYKA 1987, No. 9, p. 18

Subprograms and examples of basic graphic objects drawing in C language on IBM PC computer.

Bielecki J.: C-Sprache — Darstellung von grafischen Grundobjekte

INFORMATYKA 1987, Nr. 9, S. 18

Unterprogramme und Beispiele zur Darstellung der grafischen Grundobjekte in C-Sprache auf einem IBM PC-Computer.

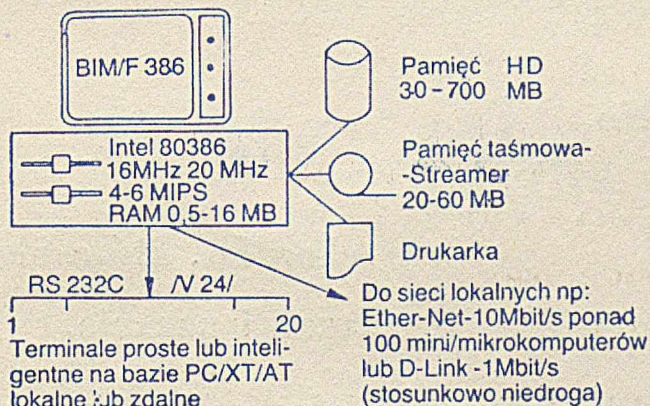
PZ globo®

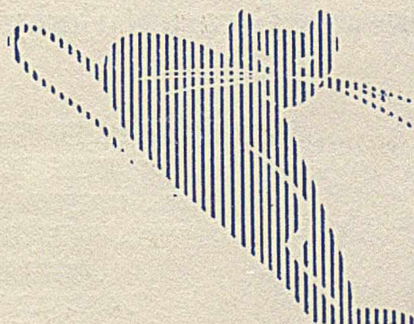
Licencjonowany producent firmy BIM
Koplin 73 200 Choszczno Tel. 7550 Telex 0445413

Oferuje nowoczesne, niezawodne, bardzo wydajne:

- Minikomputery 32-bitowe **BIM/F 386** – kompatybilne z **IBM PC/AT**
- Mikrokomputery 16-bitowe **BIM PC/XT/AT** – kompatybilne z **IBM PC/XT/AT**
- Terminale, modemy, koncentratory transmisji danych.
- Systemy użytkowe: **F-K**, materiałowy, kadrowo-płacowy, kosztorysowania.
- Sieci mikrokomputerowe (Ether-Net, D-Link/ oraz systemy wielodostępne na bazie **BIM/F 386, BIM PC/XT/AT** realizowane „pod klucz" – z oprogramowaniem systemowym i użytkowym, terminalami, modemami z instalowaniem i szkoleniem

Przykład sieci opartej o minikomputer 32-bitowy BIM/F 386





BIM®

Komputery z Przyszłością

BIM jest zastrzeżonym znakiem towarowym firm BIM Technologies AG i PZ Globo
IBM jest zastrzeżonym znakiem towarowym International Business Machines Corporation

EO/614/87

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

EO/74/K/87