10 1993

informatyka

WYDAWNICTWO SIGMA NOT

Cena 25 000 zł

informatyka

nr 10 1993
Październik
Rok wydania XXVIII

Miesięcznik
ISSN 0542-9951
INDEKS 36124

WYDAWCA:
Wydawnictwo Czasopism i Książek Technicznych SIGMA NOT
Spółka z o.o.
ul. Ratuszowa 11
00-950 WARSZAWA
skrytka pocztowa 1004

Redakcja:
01-552 Warszawa,
Pl. Inwalidów 10, p. 104, 105
tel. 39-14-34

Materiałów nie zamówionych redakcja nie zwraca

W numerze:

W najbliższych numerach:

- **Zbigniew Benedykt snuje rozważania na temat korzyści wynikających z obiektowego podejścia do analizy i projektowania systemów informatycznych.**
- **Wojciech Głazek naświetla założenia, projekt i realizację sieci transmisji danych KOLPAK – przygotowanej dla PKP.**
- **Wojciech M. Jaworski, Andrzej Zaliwski i Marian Kuraś popularyzują nową technikę notacyjną infoMapy – mogącą stanowić podstawę rozwoju oprogramowania.**
- **Witold Komorowski i Krzysztof Godula zajmują się analizą stosowania rozkazów mikroprocesora 8086.**
- **Stanisław Kędzierski charakteryzuje modelowanie obiektów dynamicznych.**
- **Jan Ryżko omawia projekt nowych standardów dla zestawu znaków w przetwarzaniu i przesyłaniu informacji.**

Skład i druk: Drukarnia SIGMA NOT Sp. z o.o. z. 355/93

Piąta Jubileuszowa Międzynarodowa Górska Szkoła PTI Szczyrk
21-25.06.1993 r.

WŁODZIMIERZ ADAMSKI
WSK-OBR-PZL-MIELEC

Sieci komputerowe w polskim przemyśle lotniczym

Znaczny rozwój mikrokomputerów, jaki nastąpił w ostatnich czasach, zmienił sposób podejścia do komputeryzacji przedsiębiorstw. W biurach instaluje się coraz więcej sieci lokalnych (NOVELL NETWARE) wypierających duże instalacje komputerowe. Przyjmuje się, że duży komputer jest wygodniejszy wówczas, gdy program sięga do bazy danych o wielkości co najmniej 40 MB.

Budowę sieci mikrokomputerowej w WSK PZL Mielec rozpoczęto w 1987 roku. Obecnie w lokalnej sieci mikrokomputerowej pracuje 65 mikrokomputerów (od PC/AT do PC/486) – rys. 1. W skład sieci wchodzą cztery *file servery*. Pierwszy o nazwie *BDK*1 z dyskami o pojemności 2 × 320 MB i dyskiem roboczym 1 GB obsługuje bazy danych konstrukcyjnych, bazy danych administracyjnych (planowanie, rozliczanie z wykonanych prac, obsługa sekretariatów biur konstrukcyjnych), obliczenia wytrzymałościowe statyczne i zmęczeniowe, aeroelastyczności, aerodynamiczne i obciążenia oraz bazę danych z prób statycznych i dynamicznych a także pomiarową bazę danych z prób związanych z „oblotem" samolotów. Drugi serwer z dyskami o pojemności 2 × 550 MB, o nazwie *BDT*1, obsługuje biura technicznego przygotowania produkcji i biura technologiczne. Technologiczną bazą danych zarządza system *DSS*. Trzeci serwer *GEO* 1 zawiera bazę danych związanych z numerycznym kształtem samolotu oraz dane dotyczące programowania obrabiarek sterowanych numerycznie. Pojemność dysku serwera wynosi 600 MB. Głównymi systemami na tym serwerze są *DAMS* i *SPO*. Ostatni, czwarty serwer *FK*1 zawiera bazę danych kadrowych, płacowych i finansowych, obsługuje księgowość, gospodarkę środkami trwałymi itp. Pojemność dysku serwera wynosi 150 MB.

Wszystkie serwery są połączone między sobą kartami ETHERNET. W serwerach *BDK*1 i *BDT*1 drugi dysk jest „zwierciadlaną" kopią pierwszego, co stanowi zabezpieczenie niektórych ważniejszych danych. W pozostałych serwerach kopie są wykonywane na dyskach lokalnych, które mają wystarczającą pojemność (po 230 MB).

Nowoczesne komputery i ich oprogramowanie odgrywają dominującą rolę w rozwoju i wytwarzaniu w WSK-PZL Mielec. Od wielu już lat stosowana jest technologia CAD/CAM (*Computer Aided Design and Manufacturing*). Eksploatowane są następujące systemy (niektóre od 1977 roku): numeryczne odwzorowanie geometrii NMG (*Numerical Master Geometry*), DAMS (*Design All Manufacturing Surface's*), AUTOCAD (projektowanie konstrukcji wytworu), baza danych specyfikacji konstrukcyjnych DSS (*Design Specification System*), automatyczne programowanie OSN do 2 1/2 osi SPO, automatyczne programowanie OSN do 5 osi APT IV, system obliczeń technicznych AFW (*Aerodynamika, Flatter, Wytrzymałość*).

Systemy mogą być integrowane w różny sposób. Najprostszą metodą jest wprowadzenie translatorów. I tak: dla czterech systemów (rys. 2), zachodzi konieczność zbudowania dwunastu różnych translatorów. Wprowadzenie piątego systemu wymaga dodatkowo dołożenia ośmiu translatorów. Tak więc w miarę rozbudowy systemu liczba translatorów rośnie w sposób geometryczny. Dlatego lepszym rozwiązaniem jest wykorzystanie standardowego formatu wymiany danych, który umożliwia współpracę z wieloma innymi systemami. Do najbardziej znanych standardowych pakietów wymiany danych należą:
– IGES (*Initial Graphics Exchange Specification*) – W. Brytania,
– SET (*Specifications du Standard D'Exchange et de Transfert*) – Francja,
– VDA-FS – RFN.

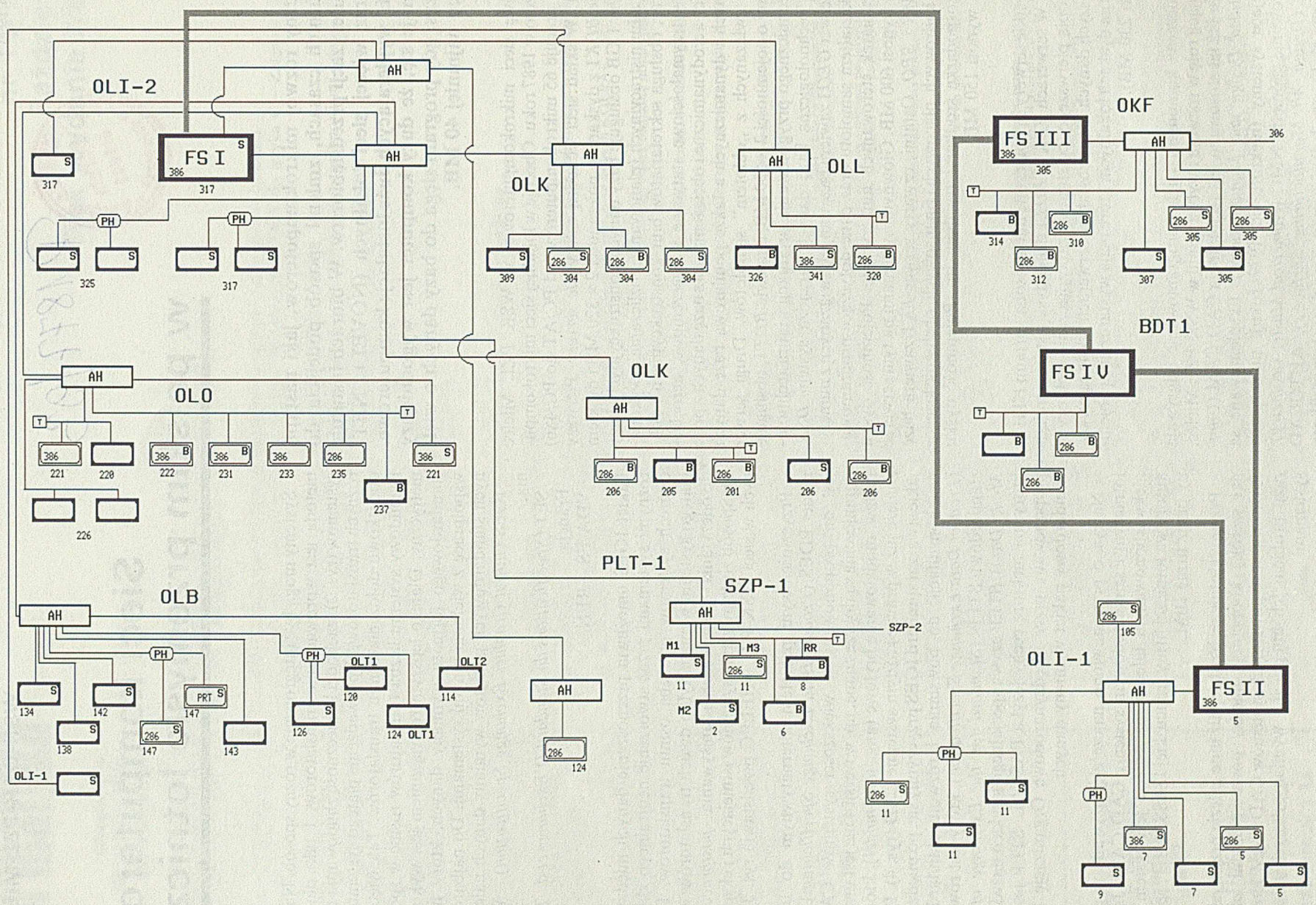
Pakiet IGES umożliwia przenoszenie danych między dwoma różnymi systemami przez stworzenie neutralnego zbioru. Dla czterech systemów potrzeba ośmiu translatorów. Dołożenie piątego systemu wymaga tylko dwóch translatorów: służącego do odczytania – *preprocesora* i zapisywania – *postprocesora* (rys. 2). Sposób zintegrowania systemów istniejących i eksploatowanych w sieci WSK PZL MIELEC przedstawia rys. 3.

Przyszłościowym pakietem alternatywnym do IGES jest pakiet PDES (*Product Data Exchange Specification*). Obecnie jest on proponowany w większości systemów CAD. Jego ograniczenia są dobrze znane, np. w wersji 4 nie jest uwzględnione tzw. modelowanie bryłowe (ang. *solid model*), które można znaleźć już w prawie każdym systemie CAD (rys. 4). Problemy, jakie stwarzał standard IGES przyczyniły się, po międzynarodowej konsultacji, do powstania i rozwoju standardu PDES (*Product Data Exchange Specification*), który jest równoważny standardowi STEP (*Standard for the Exchange of Product Model Data*). STEP używa specjalnego języka o nazwie *Expass*, który ma niektóre cechy języka PASCAL. STEP jest obecnie proponowany przez Międzynarodową Organizację ISO jako standardowy pakiet wymiany danych.

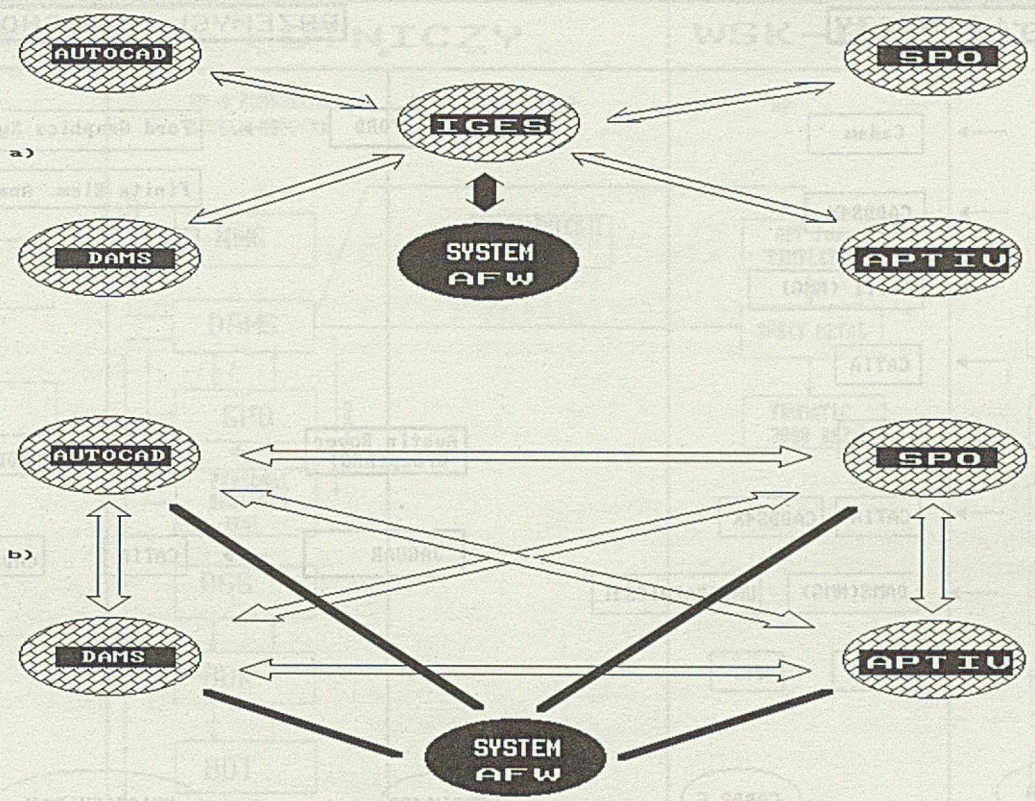
Na rys. 5 przedstawiono najczęściej używane w przemyśle lotniczym i samochodowym systemy CAD/CAM. Jak widać, niekwestionowanym liderem na świecie jest system CATIA, autorstwa lotniczej firmy francuskiej DASSAULT, sprzedawany przez firmę IBM.

Przemysł lotniczy, bardziej niż inne gałęzie przemysłu, rozwinął szeroką kooperację między poszczególnymi zakładami. Różnorodność powstałych systemów CAD/CAM narzuciła od razu problem ich integracji i wymiany danych między tymi systemami.

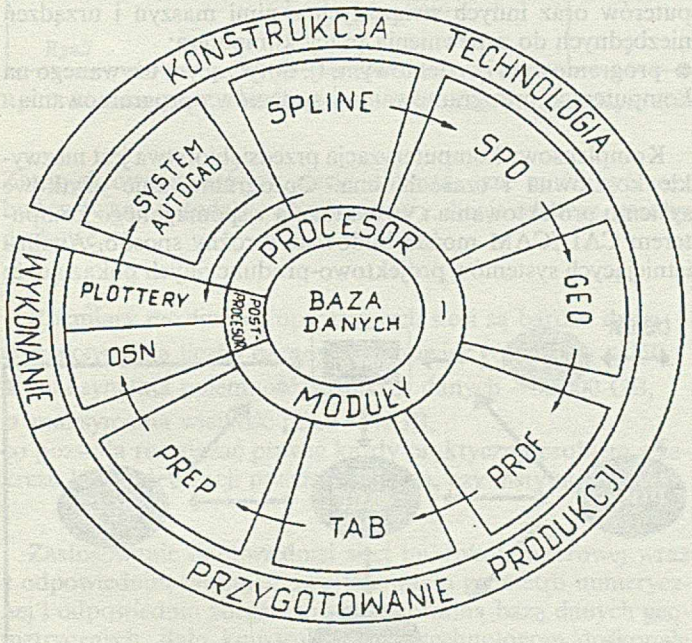
Eksploatowane obecnie w przemyśle europejskim systemy, związane z numerycznym opisem kształtu, takie jak:
- MBB – Cadam,
- Aerospatiale – Computervision, STREAM 100,



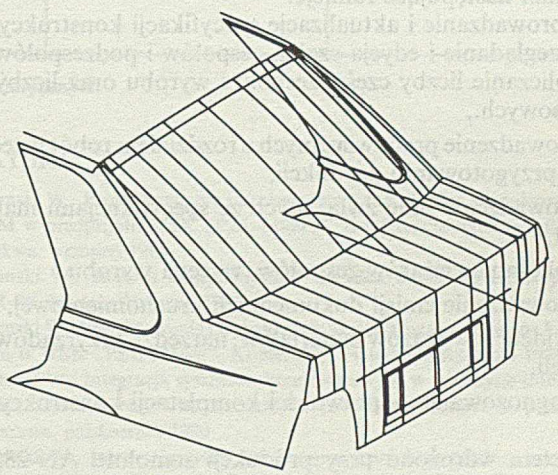
Rys. 1. Sieć mikrokomputerowa NOVELL WSK PZL MIELEC



Rys. 2. Połączenie różnych systemów za pomocą translatorów
a) z standardem za pomocą neutralnego zbioru IGES
b) bez standardu



Rys. 3. System projektowania obrabiarek sterowanych numerycznie SPO



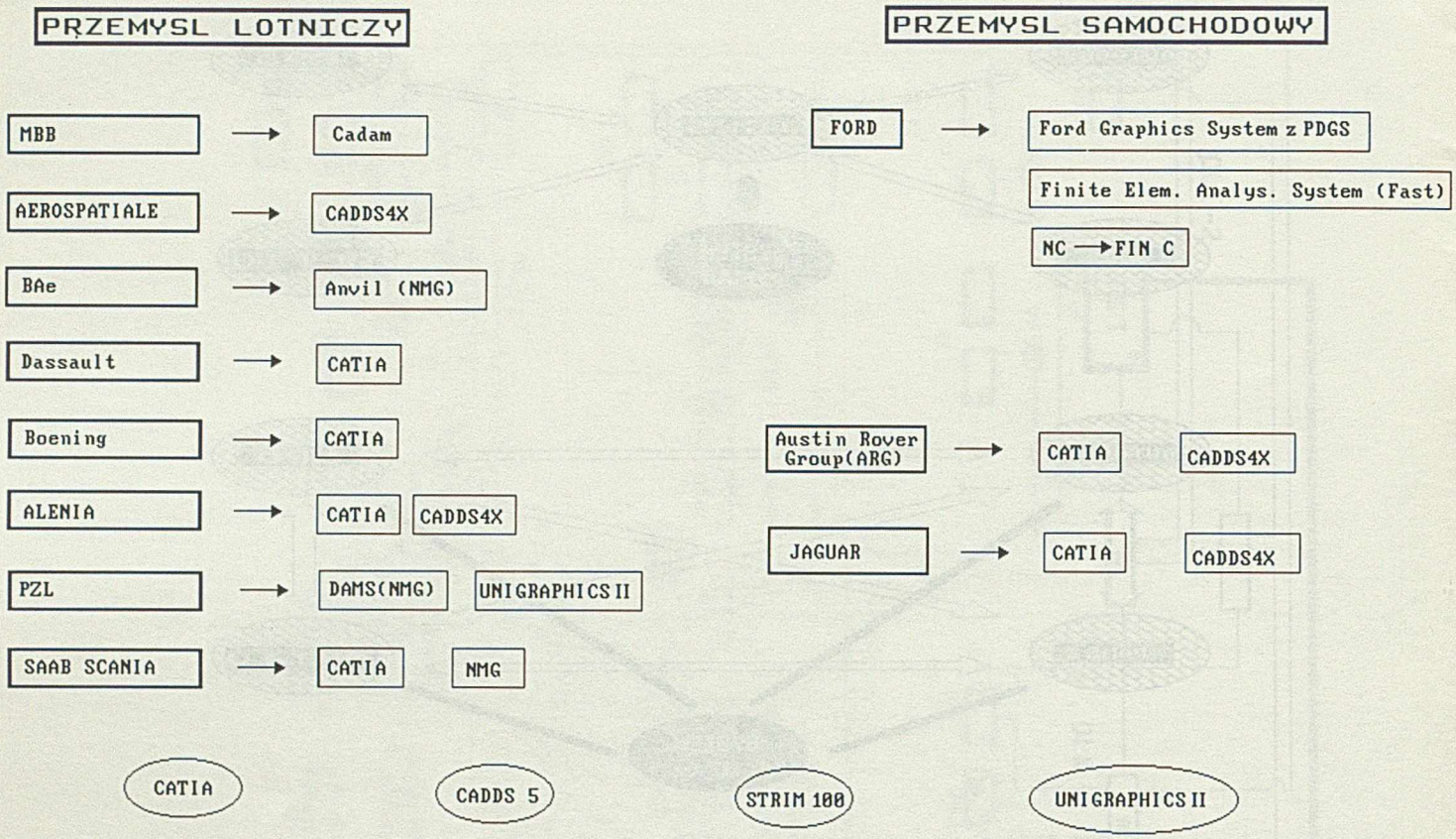
Rys. 4. Geometria nadwozia samochodu

- Aeritalia – Computervision,
- Dassault – Catia,
- British Aerospace – NMG, ANVIL 4000,
- Saab-Scania – NMG, Catia,
- PZL Mielec – NMG, DAMS,

umożliwiają zwiększenie wydajności pracy konstruktora, uzyskanie polepszenia konstrukcji przez zbadanie alternatywnych rozwiązań, skrócenie czasu opracowań oraz opracowanie programów obróbczych na obrabiarkach sterowanych numerycznie.

Dzięki wprowadzeniu sieci mikrokomputerowej bardzo rozwinął się system SPO – automatycznego programowania OSN do 2 1/2 osi. Połączenie tego systemu z systemem AutoCad dało technologowi możliwości szybkiego i dobrego przygotowania programów obróbczych na OSN (rysunki na monitorze w kilku warstwach, razem lub osobno, np. kontur teoretyczny detalu, droga narzędzia w obróbce zgrubnej, wykańczającej, ruchy jałowe, powiększenia itp.). Na rys. 3 przedstawiono schemat przepływu informacji między poszczególnymi modułami systemu SPO. Stosowany w WSK PZL Mielec system CAD/CAM 3D/2D składa się z trzech głównych pakietów (rys. 6):

- DAMS (*Design All Manufacturing Surface's*),
- SPO (*System Programing Operation*),
- DSS (*Design Specification System*).



Rys. 5. Najczęściej stosowane systemy CAD/CAM

System DSS stanowi uzupełnienie dwóch pierwszych pakietów i spełnia następujące funkcje:

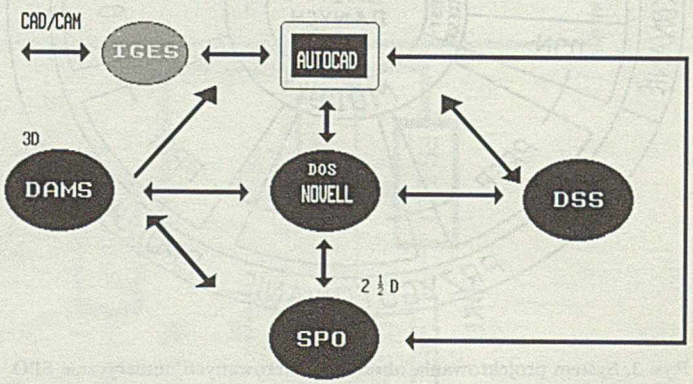
- wprowadzanie i aktualizacje specyfikacji konstrukcyjnej,
- przeglądanie i edycja części, zespołów i podzespołów,
- obliczanie liczby części zespołu i wyrobu oraz liczby części zapasowych.,
- prowadzenie prac związanych z rozdziałem robót w technicznym przygotowaniu produkcji,
- prowadzenie prac związanych ze specyfikacjami materiałowymi,
- obliczanie ciężarów zespołów, wycena wyrobu,
- prowadzenie emisji dokumentacji uruchomieniowej,
- ewidencja wykazów materiałów, narzędzi, przyrządów i technologii,
- diagnozowanie poprawności kompletacji konstrukcyjnych.

System wdrożono przy produkcji samolotu AN-28, który składa się z 17 500 części, 5100 zespołów i 4000 części znormalizowanych. Baza danych wszystkich wersji i kompletacji zajmuje 35 MB. Obecnie znajdują się w niej kompletne informacje o wyrobach produkowanych przez przedsiębiorstwo. Samolot AN-28 jest produkowany w sześciu wersjach i czterech kompletacjach, samolot M-20 w siedmiu wersjach i czterech kompletacjach, samolot IRYDA I-22 w jednej wersji i dwóch kompletacjach, samolot SOCATA w czterech wersjach. Pozostałe wyroby, jak SATURN i M-26 są produkowane na razie w jednej wersji. Baza danych wyżej wymienionych wyrobów zajmuje łącznie około 84 MB. Większość stosowanych systemów jest eksploatowana zarówno na dużym komputerze IBM 4381, ja i w sieci mikrokomputerowej NOVELL (rys. 7). Dlatego też planuje się połączenie tych dwóch sieci w jedną globalną sieć przy wykorzystaniu protokołu SNA.

Rozwój informatyki w przedsiębiorstwach należy rozpatrywać w dwóch aspektach:

- sprzętowym, tj. komputerów, minikomputerów, mikrokomputerów oraz innych związanych z nimi maszyn i urządzeń niezbędnych do zapewnienia pracy komputera;
- programowym i systemowym, tj. dotyczącym używanego na komputerach oprogramowania i systemów oprogramowania.

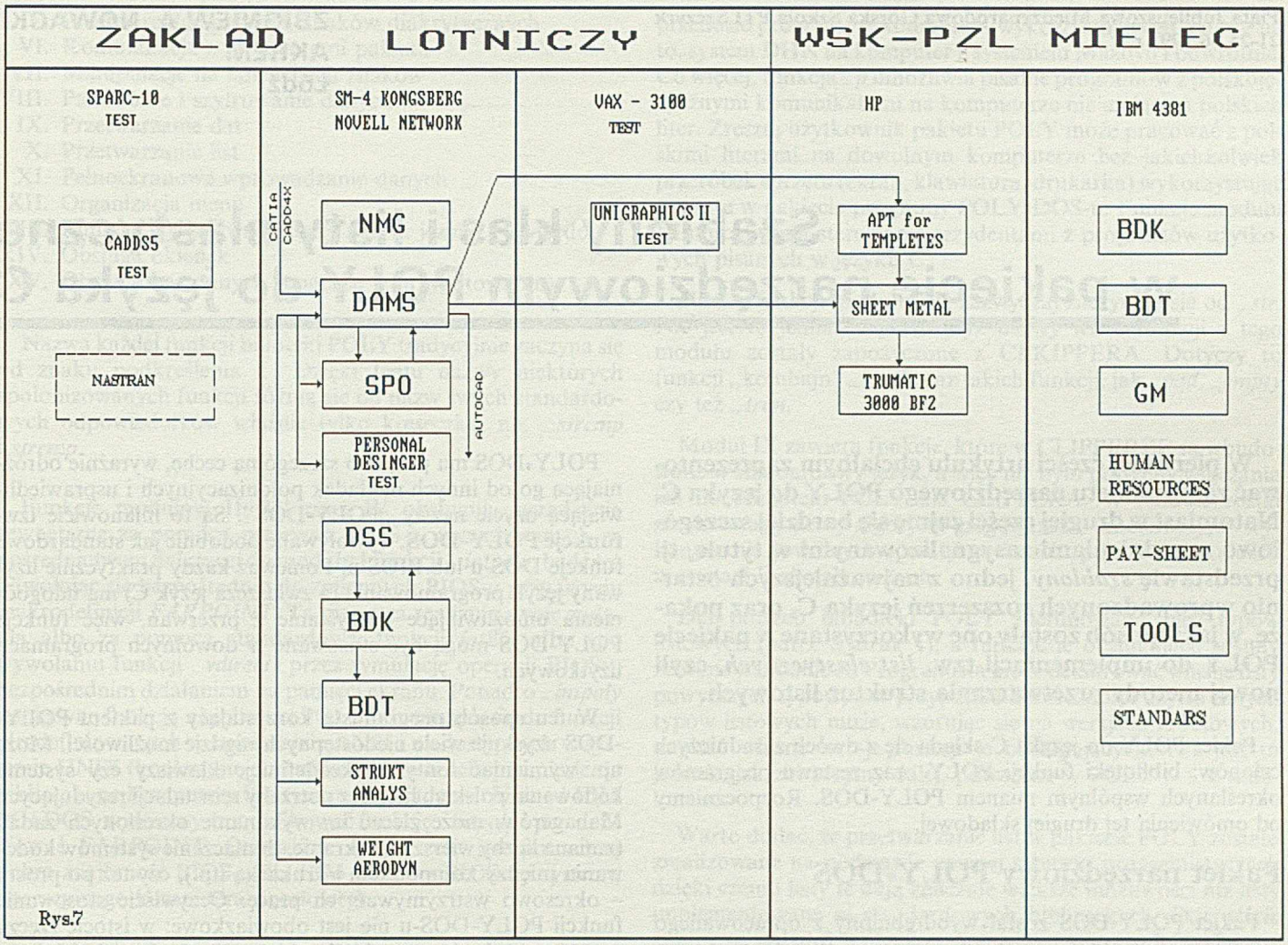
Kompleksowa komputeryzacja przedsiębiorstwa jest niezwykle kosztowna i czasochłonna. Oprogramowanie użytkowe systemu projektowania i wytwarzania wspomaganego komputerem CAD/CAM można budować w różny sposób. Analiza istniejących systemów projektowo-produkcyjnych pokazuje, że



Rys. 6. System CAD/CAM w sieci NOVELL

jedynie 5% oprogramowania jest rzeczywiście oryginalnym wkładem autorów systemu danej generacji. Ponad 80% oprogramowania jest kopią oprogramowania poprzedniej wersji, a około 10% oprogramowania stanowi lekko zmodyfikowane opracowanie.

Tak więc szybkiemu rozwojowi komputeryzacji nie towarzyszą wielkie skoki technologiczne w dziedzinie oprogramowania.



Rys. 7. Stosowany w Zakładzie sprzęt do eksploatacji poszczególnych systemów informatycznych

Zaprojektowana sieć jest siecią otwartą, tzn. może być rozbudowywana o dodatkowe *file serwery,* stacje robocze, zasoby dyskowe.

Rozmiary możliwości operacyjnych sieci są bardzo duże:

- maksymalna liczba czynnych stanowisk roboczych – 250,
- maksymalna pojemność nośników danych – 32 000 GB,
- maksymalna wielkość pliku – 4 GB,

co pozwala rozwiązać prawie każdy praktyczny problem z zakresu komputeryzacji przedsiębiorstwa, czy instytucji.

Zastosowanie odpowiedniej sieci mikrokomputerowej wraz z odpowiednim systemem projektowania geometrii numerycznej i odpowiednio zorganizowaną centralną bazą danych geometrycznych, dało konstruktorom i technologom doskonałe i sprawne narzędzie w ich pracy. Sieć mikrokomputerowa z powodzeniem zastąpiła duże i drogie instalacje komputerowe, które są stosowane przez wszystkie wielkie firmy lotnicze. Umożliwia ona każdemu upoważnionemu użytkownikowi natychmiastowy dostęp do banku danych geometrycznych.

LITERATURA

[1] Adamski W.: Projektirowanie i proiswodstwo samoliotów s ispolzowaniem EWM w predprijatii WSK „PZL-Mielec". Konferencja naukowa RWPG, Moskwa, wrzesień 1986

[2] Adamski W.: Rozwój komputeryzacji Ośrodka Badawczo-Rozwojowego OBR-SK Mielec. Wrzesień 1987

[3] Adamski W.: Projektowanie i wytwarzanie samolotów wspomagane komputerem w WSK „PZL-Mielec". Konferencja Naukowa, Rydzyna 1988

[4] Adamski W.: Integracja systemów komputerowych w przemyśle lotniczym. III Konferencja Naukowa aktualnych problemów lotnictwa polskiego, Warszawa, październik 1990

[5] Adamski W.: Integracja systemów komputerowych w polskim przemyśle lotniczym. VII Krajowa Konferencja Automatyzacji Procesów Przemysłowych, Kozubnik, wrzesień 1992

[6] Adamski W.: Stosowane systemy CAD/CAM w polskim przemyśle lotniczym. XV Międzynarodowe Sympozjum Naukowe, Zielona Góra, IV 1993

[7] Cesky Vybor Strojnicke Spolecnosti CSVTS: Stav a Vyuziti Vypocenti Techniky Pri Konstruovani a Vyrobe Letadel, Duben 1984

[8] Elliot W.S.: Computer-aided mechanical engineering: 1958 to 1988. Computer-Aided Design, Volume 21, Number 5, June 1989

[9] Mayer R.J.: One answer to the problems of CAD database exchange. Byte, No. VI/1987

[10] Saab Aircraft Division: CAD/CAM and Geometry. Saab Aircraft Division application of advanced technology, II 1991.

Piąta Jubileuszowa Międzynarodowa Górska Szkoła PTI Szczyrk
21-25.06.1993 r.

ZBIGNIEW A. NOWACKI
AKREM
Łódź

Szablony klas i listy elastyczne w pakiecie narzędziowym POLY do języka C

W pierwszej części artykułu chciałbym zaprezentować zarys pakietu narzędziowego POLY do języka C. Natomiast w drugiej części zajmę się bardziej szczegółowo zagadnieniami zasygnalizowanymi w tytule, tj. przedstawię *szablony*, jedno z najważniejszych ostatnio wprowadzonych rozszerzeń języka C, oraz pokażę, w jaki sposób zostały one wykorzystane w pakiecie POLY do implementacji tzw. *list elastycznych*, czyli nowej metody przetwarzania struktur listowych.

Pakiet POLY do języka C składa się z dwóch zasadniczych członów: biblioteki funkcji POLY oraz zestawu programów określanych wspólnym mianem POLY-DOS. Rozpoczniemy od omówienia tej drugiej składowej.

Pakiet narzędziowy POLY-DOS

Pakiet POLY-DOS został wyodrębniony z opracowanego wcześnie pakietu POLY5+87 (dla języka CLIPPER), do którego sukcesywnie były włączane programy rezydentne i narzędziowe o uniwersalnym zakresie działania. Jądro POLY-DOS-u stanowią trzy programy rezydentne: *Video Manager POLYFONT, Keyboard Manager POLPRESS* oraz *Printing Manager POLPRINT*. Uzupełniają je programy narzędziowe: edytor fontów dla monitora i drukarki PFONTED, edytor makrodefinicji klawiatury KEYEDIT, program definicji „download" POLDRUK oraz inne.

Pakiet POLY-DOS rozwiązuje, rzecz jasna, wszelkie problemy związane z polskimi literami (dowolny system kodowania, opcja zmiany systemu w „locie", rozmaite polskie klawiatury nieznikające polskie znaki na ekranie i drukarce), ale daje też, a może przede wszystkim, kilka udogodnień rozszerzających możliwości standardowego PC. Przykładowo, Video Manager pozwala pracować w trybie tekstowym z zestawem 512 fontów, z których wszystkie lub niektóre mogą być zdefiniowane przez użytkownika, umożliwia natychmiastowe przejście z 25 wierszy ekranu na 28, 43 lub 50 wierszy i odwrotnie, zaś Keyboard Manager jest w stanie przyporządkować ponad 1000 makrodefinicji o dowolnej długości do poszczególnych klawiszy (modyfikowanych, być może, klawiszami sterującymi).

Programy POLPRINT i POLDRUK pozwalają drukować dowolne fonty w trybie tekstowym na drukarce wyposażonej w opcję „download" (wg standardu Epson lub IBM). POLDRUK ma bardzo duże możliwości (włącznie z trybem NLQ), natomiast POLPRINT – niezwykłą zdolność automatycznego definiowania fontów po każdym włączeniu drukarki. W pakiecie są dostarczane również pliki zawierające gotowe fonty, możliwe do wykorzystania zarówno na drukarce, jak i na ekranie.

POLY-DOS ma ponadto szczególną cechę, wyraźnie odróżniającą go od innych nakładek polonizacyjnych i usprawiedliwiającą użycie nazwy „POLY-DOS". Są to mianowicie tzw. **funkcje POLY-DOS**, wywoływane podobnie jak standardowe funkcje DOS-u lub BIOS-u. Ponieważ każdy praktycznie używany język programowania (a zwłaszcza język C) ma udogodnienia umożliwiające korzystanie z przerwań, więc funkcje POLY-DOS mogą być stosowane w dowolnych programach użytkowych.

W ten sposób programista korzystający z pakietu POLY-DOS uzyskuje wiele niedostępnych nigdzie możliwości. Może np. wymieniać fonty, makrodefinicje klawiszy czy systemy kodowania polskich liter bez potrzeby reinstalacji rezydujących Managerów, może zlecać im wykonanie określonych zadań (zmiana liczby wierszy na ekranie, tłumaczenie systemów kodowania między komputerem i drukarką itp.), czy też po prostu – okresowo wstrzymywać ich pracę. Oczywiście stosowanie funkcji POLY-DOS-u nie jest obowiązkowe; w istocie rzeczy programy wchodzące w skład pakietu same wykonują je często w sposób automatyczny.

Rezydentne programy pakietu POLY-DOS zostały napisane w języku ASSEMBLER, za nierezydentne – w języku C. Zapewniło to odpowiednią efektywność systemu, a w szczególności stosunkowo niewielką zajętość pamięci. Przykładowo, jeśli oszczędzanie pamięci jest konieczne, to rezygnując z niektórych mniej ważnych funkcji można spowodować, iż POLYFONT po instalacji zajmie tylko ok. 4 KB, zaś POLPRESS i POLPRINT – po 2 KB pamięci.

Biblioteka funkcji POLY dla języka C

Biblioteka POLY dla C powstała z opracowanej wcześniej biblioteki POLY dla CLIPPERA, której większość funkcji była napisana właśnie w języku C. Zostały one zmodyfikowane zgodnie z filozofią C, a ponadto dołączono do nich kilka dodatkowych funkcji, bądź niepotrzebnych w CLIPPERZE, jako że wchodzących tam do podstawowego języka, bądź też niemożliwych w nim do wykorzystania.

Biblioteka może być wykorzystywana zarówno w tradycyjnym C wg standardu ANSI, jak i w C++. W tym drugim przypadku programista ma do dyspozycji wiele dodatkowych udogodnień, których zestaw określany jest mianem POLY++. Wszystkie funkcje są dostarczane dla użytkownika również w postaci źródłowej. Ponad 300 funkcji biblioteki POLY można podzielić na następujące zasadnicze moduły:

I. Realizacja przerwań BIOS-u
II. Obsługa ekranu
III. Obsługa klawiatury
IV. Obsługa drukarki

V. Przetwarzanie polskich znaków diakrytycznych
VI. Komunikacja z rezydentami pakietu
VII. Manipulacje na łańcuchach znaków
VIII. Pakowanie i szyfrowanie danych
IX. Przetwarzanie dat
X. Przetwarzanie list
XI. Pełnoekranowe wprowadzanie danych
XII. Organizacja menu
XIII. Funkcje wywoływane przez makra języka NewOrder
XIV. Obsługa okienek
XV. Obsługa baz danych (upgrade w przygotowaniu).

Nazwa każdej funkcji bilioteki POLY tradycyjnie zaczyna się od znaku podkreślenia _. Dzięki temu nazwy niektórych spolonizowanych funkcji różnią się od nazw swych standardowych odpowiedników właśnie tylko kreseczką, np. *_strcmp* i *strcmp*.

Funkcje modułów II, III oraz IV obsługują urządzenia zewnętrzne za pomocą przerwań BIOS-u, wywoływanych za pośrednictwem należącej do modułu I funkcji *_intpoly*, bądź też odwołując się bezpośrednio do zmiennych BIOS-u przy użyciu makrodefinicji *FARPOINT*. Ta pierwsza realizuje swoje zadania albo za pomocą standardowej funkcji *int*86, albo (po wywołaniu funkcji *_vdirect*) przez symulację operacji BIOS-u bezpośrednim działaniem na pamięci ekranu. Ponadto *_intpoly* jest jedyną funkcją z biblioteki POLY odwołującą się do funkcji niezdefiniowanych w standardzie ANSI i niedostępnych w systemie UNIX (konkretnie do *int*86). Zatem w celu przeniesienia biblioteki na komputery lub systemy operacyjne różne od IBM PC i DOS, wystarczy zmodyfikować funkcję *_intpoly* i przedefiniować FARPOINT. Oczywiście w tym celu jest potrzebna dobra znajomość BIOS-u dla IBM PC oraz BIOS-u i systemu operacyjnego komputera docelowego.

Moduł V nie jest określony w sposób „czysty"; wiele znajdujących się tu funkcji można by zaliczyć także do innych modułów, a w szczególności do modułu VI i VII. Funkcje z modułu V zostały opracowane w ten sposób, aby przetwarzanie z polskimi literami maksymalnie upodobnić do zwykłego, wykorzystującego alfabet międzynarodowy. Np. funkcje tradycyjnie realizowane jako makra mają w POLY spolonizowane odpowiedniki, również zrealizowane jako makra.

Warto zaznaczyć, iż biblioteka POLY jest dostosowana do dowolnego systemu kodowania polskich znaków, nawet wymyślonego *ad hoc* przez użytkownika. Poszczególne standardy są definiowane bądź przez zmienne środowiskowe DOS-u o nazwach *POL, MAZ, DHN,* itd., bądź przez zbiory dyskowe zakładane pomocniczym programem *POLZNAK*. Zmienne te lub zbiory zawierają tylko 18 polskich liter; wszystkie inne tablice tworzone są automatycznie przez funkcję *_pldef*, która musi wystąpić na początku każdego programu korzystającego z polskich liter. Funkcje z biblioteki POLY umożliwiają tłumaczenie danych z jednego systemu na drugi, a także pozwalają przenosić programy z komputera wykorzystującego, dajmy na to, system DHN na komputer z systemem Mazovii i odwrotnie. Co więcej, funkcja *_p* umożliwia pisanie programów z polskojęzycznymi komunikatami na komputerze nie mającym polskich liter. Zresztą użytkownik pakietu POLY może pracować z polskimi literami na dowolnym komputerze bez jakichkolwiek przeróbek sprzętu (ekran, klawiatura, drukarka) wykorzystując zawarte w pakiecie programy POLY-DOS-u. Funkcje modułu VI umożliwiają sterowanie rezydentami z programów użytkowych pisanych w języku C.

Nazwy funkcji modułu VII zazwyczaj zaczynają się od *_str*. Nie jest to jednak regułą, gdyż nazwy wielu funkcji z tego modułu zostały zapożyczone z CLKIPPERA. Dotyczy to funkcji „kombajn" *_stuff* oraz takich funkcji, jak *_pad, _empty* czy też *_trim*.

Moduł IX zawiera funkcje, które w CLIPPERZE są wbudowane w standardowy język, a więc nie było potrzeby włączania ich do biblioteki POLY dla CLIPPERA. W module został zrealizowany pełny algorytm gregoriański, umożliwiający wykonywanie dowolnych operacji na datach i przechowywanie ich w postaci liczb całkowitych.

Dla potrzeb biblioteki POLY zdefiniowano pięć typów listowych (patrz wydruk 1), a funkcje je obsługujące zostały zebrane w module X. Programista może deklarować własne listy powyższych typów, a w przypadku konieczności użycia innych typów listowych może, wzorując się na wersjach źródłowych, napisać własne wersje funkcji je obsługujących. Nie jest to jednak, jak zobaczymy dalej, potrzebne w POLY++ z szablonami.

Warto dodać, że przetwarzanie list w pakiecie POLY zostało zrealizowane na podstawie pewnej sztuczki programistycznej, dzięki czemu listy te dają znacznie większe możliwości niż listy implementowane w standardowych bibliotekach. Oczywiście istnieje tu również ściśle określony algorytm, w pełni widoczny dla użytkownika POLY, ale algorytm ten ma pewną charakterystyczną cechę: dość trudno go sformułować zajmując się listami do strony teoretycznej, zaś znacznie łatwiej – pisząc programy przetwarzające listy, najlepiej w języku C.

Pełnoekranowe wprowadzanie danych wykorzystujące obiekty *GET* i tzw. *formaty izometryczne* jest zrealizowane w module XI. Obiekty *GET* są dostępne już w bibliotece POLY używanej w zwykłym C, ale POLY++ umożliwia realizację pełnoekranowego strumienia danych za pomocą obiektów klasy *Getstream*, wg przykładowego schematu przedstawionego na wydruku:

```
Getstream gs;
gs << Get(...)
<< Get(...)
 ...
<< Get(...)
>> SCREEN;
```

```
typedef struct anylist
 {struct anylist *prev; void *any; } *anylist;                    (1)
typedef struct wordlist
 {struct wordlist *prev; int word;} *wordlist;
typedef struct numlist
 {struct numlist *prev;long num;} *numlist;
typedef struct namelist
 {struct namelist *prev; char* name;} *namelist;
typedef struct keylist
 {struct keylist *prev;int key;intfun funkcja;} *keylist;
```

Wydruk 1. Typy listowe zdefiniowane dla potrzeb biblioteki POLY

Funkcje *Get* zwracają wskaźniki do obiektów *GET* podających dla każdej danej następujące informacje: komentarz wprowadzający, format, nazwa funkcji sprawdzającej poprawność, nazwa funkcji zezwolenia na wprowadzanie i inne.

Ważne miejsce w bibliotece POLY zajmują funkcje z modułu XII umożliwiające zorganizowanie menu dwuwymiarowego, rolowanego lub nierolowanego, z dowolną liczbą kolumn i wierszy. Dwuwymiarowość jest często naturalną strukturą opcji, a zawsze pozwala zaoszczędzić powierzchnię ekranu. Dzięki wprowadzeniu obiektów *PROMPT*, menu może mieć w razie konieczności dowolny nieregularny kształt. Istnieje również możliwość konstruowania menu logicznego polegającego na wyborze nie jednej opcji, lecz określonego podzbioru opcji prezentowanych. Termin „menu logiczne" pochodzi stąd, że taki wybór musi być zapisany w tablicy o elementach traktowanych jako wartości logiczne i funkcja *_logmenu* zwraca tablicę określającą dokonany wybór. Ponadto w POLY++ jest zdefiniowany operator << umożliwiający konstruowanie menu strumieniowego.

W module XIII zostały zgrupowane funkcje, które nie powinny być przez programistę wywoływane bezpośrednio, a jedynie przez makra należące do tzw. **języka NewOrder.** Język ten wykorzystuje preprocesor C w celu maksymalnego ułatwienia procesów tworzenia helpu kontekstowego, sformatowanego wprowadzania danych oraz menu wielopoziomowych, powiązanych ze strukturą programu. Jest on trochę podobny do języka NewOrder dla CLIPPERA 5, aczkolwiek wiele różnic zostało wymuszonych przez odmienne możliwości preprocesorów.

```
#include "NEWORDER.H"
void PROTOTYPE(help,(void))
PROTOTYPE(fun,(char* par))
void cfun(char*);
GENESIS
   _pldef(0);
   ASSIGN(EXTEND(F1),help)
   fun(ARGUMENT(1));
STOP(0)
FUNCTION(fun,(char* par))
   static DATE dat={1978,4,23};
   int log=TAK;
   double d=-1;
   static char t[]="ALFA21";
   MAINMENU(2,2)
      THISMENU << Prompt(2,4,"Kopiowanie")
      << Prompt(2,24,"baza Danych")
      << Prompt(4,4,"Indeksy")
      << Prompt(4,24,"kaSowanie");
      SELECT
      SUBMENU(3,2,1,1)
         THISMENU << Prompt(12,15,"Przeszukiwanie")
         << "Testowanie"
         << "Kopiowanie"
         << FREE
         << par
         << "Wyniki";
         SELECT
         BEGIN(WITHOUT,CHOICE)
            cout << "BEZ WYBORU\n";
         END
      ENDMENU
      BEGIN(0,0)
         cout << _p("WYBRANO OPCJE' 0,0\n");
      END
      BEGIN(OTHER,WISE)
         cout << "WYBRANO INNE\n";
      END
   ENDMENU
   GETSTREAM
      << Get(4,0,"Kalendarz",it,CALENDAR,&dat,"DD/MM/RRRR",
            INVERSE,sprawdz_dat,0,TAK)
      << Get(6,40,"Warunek",it,LOGICAL,&log,"......T......")
      << Get(12,0,_p("Wielkos'c'"),it,NUMERIC,&d,
            "-9,999,990.09")
      << Get(12,40,"Tekst",it,TEXT,&t[0],"XxxxCC");
   READ
STOP(1)
void FUNCTION(help,(void))
   SAVE_SCREEN
   cfun(CONTEXT(1));
   REST_SCREEN
FINISH
```

Wydruk 2. Przykładowy fragment programu w języku NewOrder

Wydruk 2 zawiera przykładowy fragment programu w języku NewOrder (a właściwie NewOrder++ ze względu na użycie operatora <<). Jak widać, komendy języka NewOrder mogą dowolnie przeplatać się z instrukcjami i deklaracjami C.

Podejście okienkowe pozwala tworzyć programy i procedury niezależne od parametrów aktualnie stosowanego okna. Za pomocą funkcji *_okno* oraz funkcji pokrewnych (należących do modułu XIV) jest możliwe utworzenie listy okien, przy czym okno znajdujące się w danym momencie na końcu listy jest nazywane oknem aktywnym. Podejście okienkowe nie ogranicza się tylko do funkcji biblioteki POLY, ale może być również wykorzystywane w wywołaniach funkcji użytkownika oraz funkcji z innych bibliotek – wszędzie tam, gdzie jako parametry stosuje się współrzędne okna lub kursora.

W czasie przygotowywania artykułu funkcje modułu XV były testowane. Przewiduje się, że upgrade pakietu POLY zawierający moduł obsługi baz danych wejdzie do sprzedaży na jesieni 1993 r.

Szablony – najnowsze rozszerzenie języka C

Pojęcie **szablonów**, zwanych również *sparametryzowanymi typami*, zostało wprowadzone do standardu Cfront 3.0, czyli najnowszego wydania obiektowo zorientowanego nadzbioru języka C, opracowanego przez Bjarne Stroustrupa z firmy AT&T Bell Laboratories. Pierwsza kompletna komercyjna implementacja szablonów pojawiła się w kompilatorze Borland C++ 3.1, dostępnym również w Polsce.

Najogólniej mówiąc, jeśli tylko stwierdzimy konieczność napisania wielu bardzo podobnych procedur, powinniśmy pomyśleć o szablonach. Jednym z najprostszych, lecz mimo to interesujących przykładów jest funkcja *swap(x, y)*, której zadaniem jest zamiana zawartości dwóch parametrów *x* i *y* identycznego typu. Konieczność wykonania operacji tego rodzaju jest często spotykana w informatyce, wiadomo jak ją przeprowadzić, ale dotychczas trzeba było taką funkcję pisać oddzielnie dla każdego typu.

Koncepcja szablonów rozwiązuje powyższy problem raz na zawsze. Aby ją zastosować, trzeba przede wszystkim zadeklarować szablon za pomocą deklaracji postaci:

```
template <argument_1,
          ...,argument_k>    (2)
```

gdzie $k >= 1$, a każdy argument szablonu może być jednej z następujących postaci:

```
class nazwa      (3)
nazwa_typu nazwa1     (4)
nazwa_typu nazwa1 = wyrażenie stałe
                                (5)
```

Bezpośrednio po deklaracji szablonu powinna wystąpić deklaracja lub prototyp funkcji, bądź też definicja klasy, w których można używać identyfikatorów *nazwa* z argumentów postaci (3) dla oznaczenia dowolnych typów (oczywiście ten sam identyfikator użyty wielokrotnie reprezentuje ten sam typ), zaś identyfikatorów *nazwa1* z argumentów (4) lub (5) dla oznaczenia wielkości stałych (argumenty te są nazywane *pozatypowymi*).

Deklaracja funkcji, poprzedzona deklaracją szablonu, jest określana mianem **szablonu funkcyjnego** lub **funkcji ogólnej** (ang. *generic function*). W szablonie funkcyjnym nie wolno stosować argumentów pozatypowych. Natomiast konkretny przypadek użycia szablonu funkcyjnego nazywa się funkcją **szablonową** i formalnie rzecz biorąc, nie różni się niczym od wywołania zwykłej funkcji; kompilator sam zadba o to, aby został wykorzystany odpowiedni szablon.

Powracając do naszego przykładu funkcji *swap* możemy zastosować następujący szablon funkcyjny:

```
template <class T>
void swap(T & x, T & y)
{
 T robocza;
 robocza=x;
 x=y;
 y=robocza;
};
```

Dla każdego typu *T* takiego, że w programie znajduje się przynajmniej jedno wywołanie postaci *swap(x, y)*, gdzie *x* i *y* są typu *T*, kompilator wygeneruje dokładnie jeden egzemplarz funkcji szablonowej przeprowadzającej żądaną wymianę. Zatem jest to dokładnie to, co musieliśmy do tej pory robić „ręcznie". Widać też, że bez straty efektywności można teraz wykonać w C wiele rzeczy możliwych dotąd tylko w językach takich, jak CLIPPER, gdzie typ zmiennej jest znany dopiero w czasie realizacji programu.

Klasa, której definicja jest poprzedzona deklaracją szablonu, nazywa się **szablonem klas**, **generatorem klas** albo też **klasą ogólną** (ang. *generic class*). Jeśli *X* jest nazwą takiej klasy, to użycie jej konkretnej wersji musi mieć postać:

```
X<T1, ..., Tm>                  (6)
```

gdzie *T*1, ..., *Tm* powinny być aktualnymi argumentami odpowiadającymi formalnym argumentom z (2). Dokładniej mówiąc, pod argumenty postaci (3) należy podstawić konkretne typy, zaś pod argumenty pozatypowe – stałe wyrażenia typu zgodnego z podanym w (4) lub (5). Liczba argumentów aktualnych może być mniejsza od liczby argumentów formalnych tylko wtedy, gdy wszystkie brakujące argumenty były postaci (5); występujące w nich wyrażenia są traktowane jako wartości domyślne. Zaleca się zastępowanie (6) odpowiednią nazwą wprowadzoną przez *typedef*.

Listy elastyczne

Aczkolwiek nie jestem w stanie zaprezentować tu całego algorytmu przetwarzania list w bibliotece POLY, to jednak przedstawiam jego najistotniejszą część, a mianowicie strukturę danych tworzących listę. Niektórzy Czytelnicy mogą poczuć się rozczarowani, ponieważ struktura ta jest niezwykle prosta (ale właśnie dlatego to działa). Definicja podstawowego szablonu klas o nazwie *polylist* w pliku POLYTEMP.H zawiera tylko dwie składowe danych i zaczyna się następującym fragmentem:

```
template <class T>
class polylist
{
anylist lista;
static T* (*copy)(T*);
 ...
```

Dalej występują już tylko definicje operatorów i składowych funkcji. Najistotniejszą składową danych jest oczywiście *lista*, która jest typu *anylist* – dokładnie takiego, jaki został zdefiniowany przez (1), z tym że w C++ musimy użyć:

```
typedef
 struct Anylist {Anylist
      *prev; void *any; } Anylist;
typedef Anylist *anylist;
```

Pomimo formalnych różnic powyższa definicja znaczy to samo co (1), a zatem funkcje obsługi list typu *anylist*, zawarte w zwykłym POLY, są również wykorzystywane w POLY++.

Aby przetwarzać listy w POLY++, należy włączyć do programu plik POLYTEMP.H oraz zadeklarować używane typy listowe za pomocą makro:

```
POLYLIST(typ_listy,
              typ_obiektu, kopia)
```

gdzie *typ_listy* jest unikalną (dotychczas nieużywaną) nazwą typu, *typ_obiektu* jest nazwą dowolnego już istniejącego typu, zaś *kopia* jest nazwą funkcji o podanym wcześniej prototypie:

```
*typ_obiektu kopia(*typ_obiektu);
```

lub nazwą specjalną *NOCOPY*. Od tego momentu w programie mogą występować deklaracje takie, jak przykładowo:

```
typ_listy a, b, c;
```

powodująca utworzenie trzech początkowo pustych list *a*, *b*, i *c*, których elementami mogą być wskaźniki do obiektów typu *typ_obiektu*. Parametr *kopia* określa, czy obiekty dołączane do listy mają być kopiowane (wyjaśniamy to dokładniej w dalszej części artykułu).

W programie makro *POLYLIST* może wystąpić dowolnie wiele razy, a utworzone w ten sposób różne typy listowe mogą być wykorzystywane jednocześnie. Jak łatwo można się domyślić, każde makro *POLYLIST(x,y,z)* m.in. nadaje za pomocą *TYPEDEF* nazwę *x* dla typu *polylist<y>* (tj. konkretyzacji typu *polylist* wg (6), zaś *z* służy do inicjacji członu statycznego *copy*. Po zadeklarowaniu zmiennych listowych można poddawać je działaniu kilku operatorów. I tak operator:

```
list <= pointer
```

dodaje wskaźnik *pointer* na końcu listy *list*, zwracając referencję do *list*, co umożliwia wykonywanie operacji wielokrotnych, np.:

```
POLYLIST(textlist, char, NOCOPY)
 ...
textlist li;
li <= "CD" <= "EF" <= "GH" <= "IJ";
```

W przypadku, gdy parametr *kopia* makra *POLYLIST* jest różny od *NOCOPY*, to do listy *list* dodawany jest nie *pointer*, ale *kopia(pointer)*. Na ogół oczekuje się przy tym, że funkcja *kopia* skopiuje obiekt wskazywany przez *pointer* do nowego obszaru przydzielonego za pomocą standardowego operatora *new* i zwróci adres kopii. Zatem ten wariant może być interpretowany jako wstawianie do listy obiektów zamiast wskaźników.

Należy podkreślić, iż dzięki wykorzystaniu szablonów istnieje tu pełna kontrola typów. Na przykład sekwencja:

```
POLYLIST(intlist, int, NOCOPY)
 ...
intlist inli;
inli <= "CD";
```

spowoduje zasygnalizowanie błędu już na etapie kompilacji. Zatem chociaż listy te sprowadzają się, jak widzieliśmy, do list typu *anylist*, programista może uważać, iż dysponuje listami nieskończenie wielu rodzajów.

Zastosowane do listy unarne operatory + *i* −− zwracają ostatni (ostatnio wprowadzony) wskaźnik na liście. Jedyną różnicą między nimi jest to, że + pozostawia listę bez zmiany, a −− usuwa ostatni wskaźnik z listy. W przypadku, gdy lista jest pusta, oba operatory zwracają 0.

Również i w tym przypadku użycie szablonów daje nam istotne korzyści, polegające na tym, iż zwracane wskaźniki są odpowiedniego typu. Przykładowo, sekwencja:

```
POLYLIST(textlist, char, kopiuj)
 ...
textlist li;
 ...
cout << --li;
```

spowoduje wyprowadzenie tekstu, podczas gdy przy bezpośrednim korzystaniu z list typu *anylist* byłby niezbędny dodatkowy operator rzutowania.

Usuwanie wskaźników z listy może być także przeprowadzone za pomocą operatora −=. Wyrażenie:

```
list -= number
```

kasuje *number* ostatnich wskaźników z listy *list*. Dodatkowo, jeśli została określona funkcja kopiująca (parametr *kopia* był różny od NOCOPY), to do wszystkich usuwanych wskaźników jest stosowany operator *delete*. Operator −= zwraca referencję do list.

Operator = służy do kopiowania list. Ściśle mówiąc, wyrażenie:

```
list1 = list2
```

powoduje skasowanie wszystkich wskaźników listy *list*1, a następnie przyporządkowanie do zmiennej *list*1 kopii listy *list*2. Jeśli funkcja kopiująca została określona, to będzie ona wykorzystana do kopiowania obiektów wskazywanych przez wskaźniki listy *list*2, a ponadto obiekty wskazywane przez pierwotne wskaźniki z *list*1 będą kasowane operatorem *delete*.

Zachowywane są wszystkie intuicyjne aspekty operatora =. Przykładowo, we fragmencie programu:

```
POLYLIST(textlist,char,copy)
 ...
textlist a;
 ...
textlist b = a, c;
a = 0;
c = a = b;
```

lista *b* już w momencie zadeklarowania zostaje utworzona jako dokładna kopia listy *a*, następnie lista *a* jest czyszczona do chwili, gdy listy *c* i *a* stają się dokładnymi, acz różnymi kopiami listy *b*.

Chociaż funkcja *Len* zwraca długość listy, nie powinno się jej używać do testów na to, czy lista jest pusta. Wynika to z faktu, iż *Len* po prostu czyta całą listę, zliczając jej elementy. Jednakże operator ! został zdefiniowany tak, aby wyrażenie *!list* było różne od zera tylko wtedy, gdy *list* jest puste. Tym samym w warunkach można stosować wyrażenie *!!list*, ale nie można równoważnego pod względem logicznym wyrażenia *list* (dlaczego?).

Omówię teraz ostatni, ale najważniejszy operator list elastycznych, oznaczany symbolem |. Warto w tym miejscu przypomnieć, że oparte na teoretycznych algorytmach funkcje zawarte w bibliotekach standardowych, w celu przetworzenia elementów znajdujących się wewnątrz listy, wykorzystują tzw. iteratory.

Aczkolwiek producenci oprogramowania starają się pisać funkcje obsługujące iteratory w sposób jak najbardziej ogólny, to jednak zawsze muszą wystąpić ograniczenia wynikające z faktu, iż iteratory są dodatkowymi obiektami, nie dającymi się opisać w terminach samej listy. Otóż operator | umożliwia przetwarzanie list bez pośrednictwa iteratorów, dając programiście możliwość pełnej kontroli i optymalizacji tego procesu. Wyrażenie:

```
list | number
```

gdzie *list* jest listą, a *number* liczbą całkowitą nieujemną, definiuje się jako adres podlisty listy *list powstałej przez pominięcie number* ostatnich (ostatnio wprowadzonych) elementów. Należy podkreślić, że operator | zwraca tylko adres, a więc żadna nowa lista nie jest tworzona.

Skoro mamy już adres, to natychmiast przychodzi na myśl operator wyłuskania. Oczywiście, zgodnie z podstawowymi regułami C, również unarny operator * nie tworzy żadnego nowego bytu. Zatem *(*list*|*number*) jest listą będącą fizyczną częścią listy *list* (identyczną z *list* w przypadku *number* = 0). W konsekwencji widzimy natychmiast, w jaki sposób odczytać zawartość elementu listy, za którym znajduje się *number* innych elementów; wystarczy ewaluować wyrażenie:

```
+ * (list | number)                (7)
```

Rzecz jasna, czas ewaluacji zależy od parametru *number*. Jednak podróżując po liście od elementu do elementu nie musimy za każdym razem odwoływać się do wyrażenia (7). Możemy natomiast znacznie uoptymalizować przetwarzanie, jeśli zauważymy, iż podlista podlisty jest znów podlistą całej listy a ściśle mówiąc zachodzi następujący wzór:

```
*(list | number1) | number2 =
          list | number1 + number2
```

Wynika stąd, że optymalna pętla wyprowadzająca elementy listy na standardowe wyjście może wyglądać następująco:

```
textlist li, *a;
 ...
for(a = li|0; !! *a; a = *a|1)
                    cout << + *a ;
                                  (8)
```

Tutaj wyrażenia z operatorem | są wykonywane z maksymalną prędkością (oczywiście zamiast *li*|0 można by było napisać *&li*). Ponieważ zmiany adresu o jeden element zdarzają się bardzo często, wprowadzono specjalną funkcję *Inc* działającą na adresach list. Jeśli *adr* jest zmienną typu „wskaźnik do listy", to *Inc(adr)* wykonuje podstawienie:

```
adr = *adr | 1
```

oraz wyłuskuje i zwraca listę poprzednio wskazywaną przez *adr*. Stosując *Inc* możemy zapisać pętlę (8) następująco:

```
a = li|0;
while(!! *a) cout << +Inc(a);    (9)
```

Odczytywanie elementów jest proste z intuicyjnego punktu widzenia. Jednak skoro podlista jest normalną listą, to za pomocą operatorów −− *i* −= można kasować elementy znajdujące się na końcu podlisty. Powstaje zatem pytanie, co w tym przypadku stanie się z całą listą? Otóż nie ma powodów do obaw, bo w liście nie powstanie dziura. Po prostu można w ten sposób kasować środkowe elementy listy. Podobne uwagi dotyczą dodawania elementów do podlisty. Przykładowa instrukcja:

```
*(li | k) <= "A" <= "B" <= "C"
```

spowoduje wstawienie trzech elementów do środka listy *li* (po elemencie „C" będzie *k* już przedtem istniejących elementów). Zatem lista *li* zostanie, obrazowo mówiąc, rozepchnięta niczym rajstopy elastyczne w celu przyjęcia nowych elementów. Ta właśnie cecha list implementowanych w pakcie POLY uzasadnia nazwę *listy elastyczne*.

Oczywiście w tym samym wyrażeniu można usunąć niektóre elementy, a na ich miejsce wstawić nowe. Taki sposób pracy jest nawet zalecany, ponieważ pozycja wprowadzenia zmian jest znajdowana tylko raz. W razie potrzeby adres odpowiedniej podlisty może być zapamiętany w zmiennej wskaźnikowej.

Warto tu zaznaczyć, że listy wykorzystujące pojęcie iteratora (implementowane w standardowych bibliotekach) nie są elastyczne. W przypadku list jednokierunkowych, nowe elementy mogą być dodawane tylko na końcu listy, zaś dla list dwukierunkowych – na końcu lub początku.

Na podlistach list elastycznych można wykonywać naprawdę dowolne operacje. Wszystkie poniższe przykłady są poprawne i mogą być stosowane w sposób celowy:

```
*(li|k) = 0;
// lista li będzie zawierać k
// elementów
*(li|n) = li2; // konkatenacja list
*(li|k) = *(li|n);
li <= (--*(li|3));
```

Ostatnia operacja powoduje przestawienie elementów w liście. Jeśli została określona funkcja kopiująca, to takie wyrażenie dobrze jest umieszczać wewnątrz makra REFER, jak to pokazuje poniższy przykład:

```
REFER(textlist, li <=
                    (-- *(li | 3) ) )
```

Wszystkie operatory umieszczone tym sposobem w makro REFER są wykonywane bez kopiowania i kasowania obiektów, a więc przestawienie zostanie wykonane szybciej.

Jak już stwierdziliśmy wcześniej, wyniki działania operatora | mogą być przechowywane w zmiennych wskaźnikowych. Natomiast po wyłuskaniu nie powinny być one przesyłane operatorem = do innych zmiennych listowych, ponieważ w ten sposób otrzymujemy kopie podlist (chyba, że jest to specjalnie wymagane). Jednak można napisać, przykładowo:

```
{
 textlist &b = *(li | 4);
 ...
}
```

Tutaj *b* będzie w dalszym ciągu podlistą listy *li* i wszystkie zmiany *b* zostaną odzwierciedlone w *li*.

Pętle (8) i (9) odczytywały elementy listy w porządku odwrotnym do tego, w jakim elementy te zostały do listy wstawione. Nie znaczy to jednak, że listy elastyczne są tylko listami typu LIFO. Poniższy przykład wskazuje, jak należy wstawiać elementy do listy, aby pętle (8) i (9) odczytały elementy listy w tej samej kolejności, w jakiej te elementy zostały do listy wstawione:

```
textlist li, *a;
a = li|0;
 ...
a= *a <= p | 1;                    (10)
...
```

Dla list FIFO, wyrażenie (10) powinno być stosowane zamiast zwykłego:

```
li <= p;                           (11)
```

Oczywiście działa to dlatego, że (10) powoduje zawsze wstawienie nowego elementu na początku listy. Zauważmy, iż dzięki wykorzystaniu zmiennej adresowej *a* (10) działa niemal tak samo szybko jak (11).

Na pierwszy rzut oka, listy elastyczne wydają się być listami jednokierunkowymi. Prawdą jest (zob. (1)), iż struktury list elastycznych zawierają tylko jeden wskaźnik (*prev*) do innej struktury listy, co rzekomo wyklucza możliwość poruszania się w dwóch kierunkach. Dlatego jedną z najciekawszych własności list elastycznych jest fakt, że to wrażenie jest fałszywe. Pokaże bowiem za chwilę, w jaki sposób zorganizować poruszanie się w dwóch kierunkach na liście elastycznej. Ważną rolę, przynajmniej dla elegancji wyrażenia algorytmu, odgrywają tu znów szablony klas.

Poglądowo mówiąc, zastosowaną tu metodę można porównać ze strategią wytrawnego trapera, który poruszając się po prerii zostawia sobie tylko widome znaki ułatwiające powrót. Skoro w makro *POLYLIST* można wstawiać dowolne typy, więc nic nie stoi na przeszkodzie, aby jako argument aktualny szablonu klas wykorzystać typ, który sam został utworzony za pomocą szablonu klas. Rzeczywiście, kompilator Borlanda przyjmuje to bez zastrzeżeń i możemy napisać następujące przykładowe deklaracje:

```
POLYLIST(textlist, char, kopia)
POLYLIST(overlist,    textlist,
NOCOPY)
textlist li,*a;
overlist ov,*av;
```

Zatem lista *ov* może być nazwana „listą list" (określenie to brzmi groźnie, ale w rzeczywistości *ov* jest zwykłą listą wskaźników). Teraz wystarczy tylko poruszając się po liście *li*, przy użyciu pętelek takich, jak (8) lub (9), zapełniać listę *ov* instrukcją *ov*< = *a*. To zabezpieczy nam powrót za pomocą takich pętelek jak:

```
// w przeciwnym kierunku
for(av  =  ov|0;  !!  *av;  Inc(av))
cout << +* +* av ;
// kasowanie w przeciwnym kierunku
while(!! ov) cout << + * -- ov ;
```

Należy podkreślić, że lista *ov* nie może być zastąpiona listą typu *textlist*, ale odwrotnie uporządkowaną, bowiem wtedy poruszając się do tyłu nie wiedzielibyśmy, jak wrócić na listę podstawową. Innymi słowy, zastosowane tu podejście wymaga posiadania umiejętności stosowania adresów podlist. Z drugiej strony, jest ono bardziej efektywne od klasycznych list dwukierunkowych, ponieważ nie jest potrzebna ciągła aktualizacja dwóch wskaźników (tworzenie listy list inicjujemy dopiero wtedy, gdy może być potrzebne poruszanie się w dwóch kierunkach).

* * *

Istnieje szeroki wachlarz problemów, w których zastosowanie list wydaje się być celowe. Listy mogą zmniejszyć wykorzystanie pamięci, a jednocześnie pozwalają wyeliminować ograniczenia nakładane w wyniku beztroskiego stosowania tablic. Sądzę, że listy elastyczne z pakietu POLY tworzą bardzo efektywny i wygodny w użyciu mechanizm, który w takich przypadkach może być z powodzeniem wykorzystany.

PIOTR PORWIK
Zakład Elektroniki i Informatyki
Uniwersytet Śląski
Katowice

Multimedialne bazy wiedzy – rekomendacje

Zalety jakie płyną ze stosowania w życiu codziennym klasycznych baz danych są tak oczywiste, że nie wymagają specjalnych uzasadnień i komentarzy. Na naszym rynku z powodzeniem konkurują ze sobą różne systemy zarządzania i projektowania baz danych. Samo założenie bazy w systemach dBase, Paradox (lub innych podobnych) nie jest sprawą trudną. Dla ludzi zajmujących się projektowaniem i eksploatacją baz danych nie są także obce umiejętności prawidłowego rozkładu relacji i szybkiego wyszukiwania różnych kolekcji informacji z dostępnych baz (formularze, raporty, zapytania itp). Nowoczesne programy projektowania złożonych systemów informatycznych nie wymagają też od użytkownika dobrej znajomości technik programowania (np. Microsoft Access, Clarion Professional czy Paradox), co wydaje się być szczególnie atrakcyjne dla osób bez przygotowania informatycznego. Bardziej zaawansowani użytkownicy mogą także wykorzystywać specjalizowane języki manipulowania danymi.

Szybki rozwój technologii elektronicznej i informatycznej sprawił, że pojawiły się także nowe obszary zastosowań baz danych, dotychczas nieznane w klasycznych systemach zarządzania. Obecnie obrazy graficzne (wideo, rysunki, wykresy itp.) oraz dźwięk, mogą być także umieszczane w bazach wiedzy na zasadzie wykorzystywania mechanizmów dołączania i wbudowywania obiektów OLE (ang. *Object Linking and Embedding*). Nie trzeba chyba dowodzić, że grafika wydatnie podnosi walory prezentacyjne dokumentów, a w niektórych przypadkach jest wręcz niezbędna (np. zdjęcia lub głos ludzki w bazach kartotek policyjnych).

Wykorzystywanie i samodzielne kształtowanie obiektów OLE w bazach danych nie jest jednak w niektórych przypadkach takie proste, jakby mogło się wydawać na podstawie niektórych publikacji i opisów pakietów. Stawia także przed użytkownikiem dosyć wygórowane wymagania sprzętowe i potrzebuje nowego spojrzenia na wyposażenie, jakie powinno znajdować się w dyspozycji systemów zarządzania bazami.

Zasady, którymi należy kierować się przy opracowywaniu i rejestracji danych multimedialnych powinny być również wskazówką ułatwiającą zakupy odpowiedniego sprzętu.

Osadzanie obiektów OLE w bazie danych

Zewnętrzna reprezentacja pliku w postaci ciągu bajtów jest niewygodna dla bezpośredniego użytkownika. Stosuje się więc specjalizowane oprogramowanie narzędziowe, dokonujące konwersji na postać bardziej odpowiednią dla poszczególnych zastosowań. Jedną z często spotykanych i bardzo ważnych takich reprezentacji wysokiego poziomu jest baza danych. Logiczny model relacyjnej bazy danych (najbardziej rozpowszechnionej w praktyce) można opisać następująco. Bazę danych przedstawić można w postaci tabeli, w której wiersze i kolumny stanowią pewną relację. Każda kolumna tabeli jest nazywana polem, zawiera pewien zadeklarowany atrybut mogący przyjmować wartości ze ściśle określonego zbioru (dziedziny). Poszczególne wiersze, (inaczej krotki), są pewnym niepowtarzalnym zbiorem wartości poszczególnych atrybutów. Pola mają zwykle stałą długość, czasem jednak dopuszcza się pola o zmiennej długości.

Aby przyspieszyć odnajdywanie poszczególnych krotek, określa się często klucz wyszukiwania (indeks), bądący pojedynczym atrybutem lub ich kontaktenacją. Taka metoda eliminuje potrzebę czasochłonnego, sekwencyjnego przeglądania poszczególnych krotek w celu znalezienia właściwej.

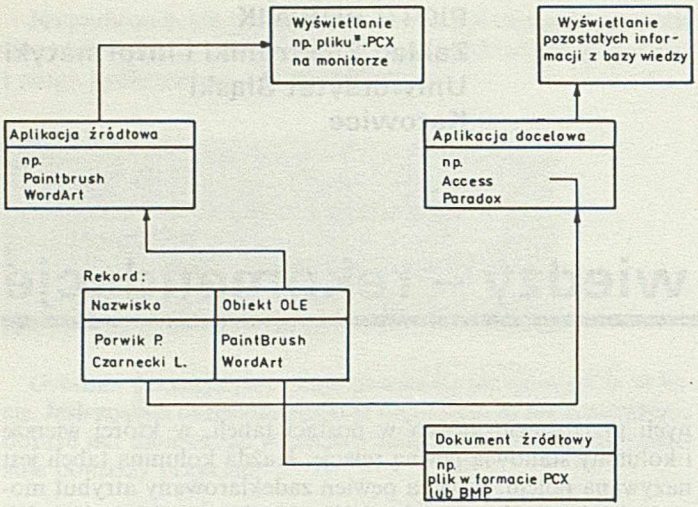
W modelu fizycznym relacja i ewentualne struktury pomocnicze są zapamiętywane w plikach. Ciąg bajtów zawierający krotkę nazywa się rekordem. Problem sposobu zapamiętywania bazy danych na zewnętrznych nośnikach informacji nie jest niestety problemem trywialnym i jest przedmiotem ciągłych badań i poszukiwań.

Mechanizmy OLE znane są ze środowiska Windows. Z programów dostarczonych wraz z systemem możliwości takie mają: *Paintbrush, Write, Sound Recorder, Cardfile* i *Object Packager*. Także inne aplikacje, nie będące częścią systemu, ale z nim współpracujące, mogą wykorzystywać te udogodnienia (np. *Ami Pro, Word for Windows* 2.0, *Word Perfect, Excel, Corel Draw*). Od niedawna mechanizmy OLE mogą być wykorzystywane także w bazach danych *Access* i *Paradox for Windows*.

Projektowanie bazy danych, w których umieszczać można obiekty OLE, w zasadzie nie różni się od metod projektowania baz bez takich obiektów. W bazach takich jest dostępny dodatkowy atrybut pola o nazwie *OLE object*. Atrybut ten ma cechy pozwalające na osadzaniu w tego typu polach obiektów o różnym źródle pochodzenia (np. dźwięków lub zdjęć).

Zawsze podczas projektowania struktury rekordu należało w standardowej bazie zdefiniować typ pola. Każde pole charakteryzowało się określoną szerokością, tzn. liczbą bajtów, jaką zajmie w rekordzie. W przypadku pola typu OLE informacja o jego szerokości jest zbędna, gdyż po prostu nie jest jeszcze znana na tym etapie przygotowywania bazy. W polu tym, w momencie umieszczania w nim obiektu, jest przechowywany identyfikator dokumentu źródłowego i jego treść wraz z nazwą aplikacji tworzącej format dokumentu źródłowego.

Pewnym odpowiednikiem logicznym (lecz nie funkcjonalnym) pola OLE jest pole notatnikowe (memo), występujące w klasycznych bazach danych. Pola typu OLE z oczywistych względów nie mogą być używane do tworzenia kluczy indeksowych i filtrów, nie mogą być też kryterium wiążącym rekordy z kilku baz.



Rys. 1. Zasada osadzania (lub korygowania) obiektu graficznego w aplikacji docelowej

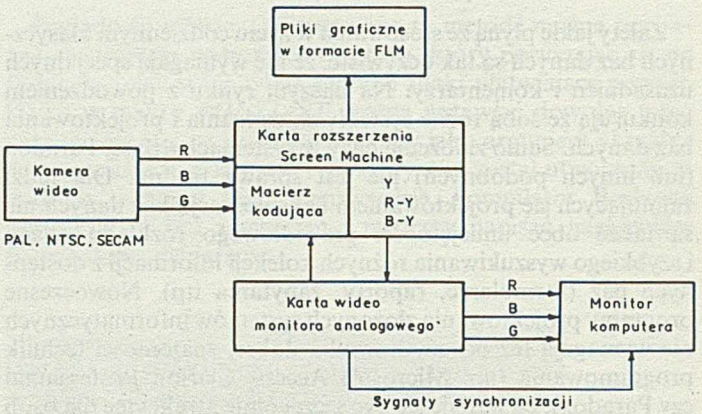
Osadzanie obiektu OLE w polu odbywa się na identycznych zasadach, jakie obowiązują w aplikacjach środowiska Windows, tzn. może odbywać się na zasadzie wbudowania lub dołączania. Dokładny opis tych metod znajduje się w dokumentacji każdego pakietu programów przeznaczonych dla tego środowiska. W bardzo przystępnym ujęciu opis ten przedstawiono także w pracy [6]. Zasadę osadzania (lub korygowania) obiektu graficznego w aplikacji docelowej przedstawia rysunek 1.

Zewnętrzna rejestracja danych dla obiektów OLE

W przypadku umieszczania zdjęć w aplikacjach Windows, trzeba dokonać konwersji rzeczywistego obiektu analogowego na odpowiadający mu obraz cyfrowy (zdyskredytowany), a następnie zapisać go w odpowiednim formacie w pliku dyskowym. Aby przeprowadzić taką operację, trzeba dysponować odpowiednim sprzętem do rejestracji i przetwarzania obrazu analogowego. Komputer powinien być więc wyposażony w odpowiedni układ sprzęgający. Rolę takiego układu może spełniać, na przykład, karta *Screen Machine* (SM) firmy *Fast Electronic GmbH* montowana w kmputerze PC. Urządzenie to, współpracując z kamerą wideo oraz z odpowiednim oprogramowaniem, umożliwia m.in. rejestrowanie obrazów typu zatrzymany kadr (ang. *still picture*), a także pozwala na ich dodatkową modyfikację (zmiany ostrości, nasycenia, kontrastu, jasności). Karta *Screen Machine* (i jej podobne) jest wielofunkcyjnym urządzeniem rejestrująco-wyświetlającym. Znajomość jej działania jest niezbędna, aby móc dysponować obiektem graficznym odpowiedniej jakości i prenosić go lub skalować w innych aplikacjach.

Doprowadzane z kamery wideo do karty *SM* sygnały elektryczne barw podstawowych: czerwonej (*R*), zielonej (*G*) i niebieskiej (*B*) są zamieniane w układzie macierzy kodującej na sygnał luminancji *Y* oraz dwa sygnały różnicowe chrominancji *R-Y* oraz *B-Y*. Sygnał luminancji zawiera informacje o jaskrawości, a sygnały chrominancji – o barwie i nasyceniu fotografowanej sceny. Wydzielenie sygnałów luminancji i chrominancji z pozoru wydaje się zbędne (przecież monitor także jest sterowany zespolonym sygnałem *RGB*). Należy jednak wziąć pod uwagę fakt, że zarejestrowany obraz może być później poddawany wielokrotnej obróbce bez ponownego instalowania kamery, a więc opisana operacja jest niezbędna.

Zarejestrowana za pomocą kamery scena jest przechowywana w pliku o specjalnym formacie *FLM*. Format ten jest obsługiwany wyłącznie przez kartę *Screen Machine* i nie jest rozpoznawany przez inne aplikacje. W pliku tego typu są przechowywane m.in. dyskretne informacje o luminancji i chrominancji zarejestrowanego obrazu. Nie przechowuje się więc, tak jak w innych popularnych formatach graficznych, palety kolorów i indeksowych odwołań do tej palety. Takie rozwiązanie umożliwia tworzenie na podstawie źródłowego pliku *FLM* innych formatów graficznych (*PCX, TIFF, GIF, DIP/BMP, Clipboard, TGA, RLE*), tworzonych na bieżąco w środowisku Windows w zależności od możliwości aplikacji, w której obiekt ma zostać umieszczony. Uproszczoną zasadę rejestrowania obrazu z wykorzystaniem kamery i karty *Screen Machine* przedstawia rysunek 2.



Rys. 2. Uproszczona zasada rejestrowania obrazu z wykorzystaniem kamery i karty Screen Machine

Rejestracja obrazu wideo i jego transformacja do formatu *FILM* są całkowicie niezależne od aktualnie ustalonych parametrów (trybu pracy) karty wideo komputera i przeprowadzane są w taki sposób, że później umożliwiają uzyskanie rozdzielczości 16 777 216 kolorów. Pliki *FLM* są niestety bardzo duże (kilkaset kilobajtów) i po wykorzystaniu powinny być usuwane z dysku.

Jakość rejestracji danych oraz wymagania sprzętowe

Samodzielne tworzenie bazy wiedzy, w której można umieszczać grafikę, obiekty OLE oraz dźwięk, wymaga niestety poniesienia znacznych nakładów finansowych. Umieszczanie w bazie ilustracji dźwiękowych lub zdjęć pochodzących z kamery wideo wymaga odpowiedniego wyposażenia stanowiska pracy. Trzeba przecież obraz analogowy przetworzyć w jego cyfrowy substytut i zapamiętać w odpowiednio formatowanych plikach. Komputer powinien być więc wyposażony w odpowiedni układ sprzęgający. Rolę takiego układu może spełniać montowana w komputerze PC wspomniana już karta *Screen Machine* firmy *Fast Electronic GmbH*. Ponieważ rejestrowane obrazy mogą być barwne, o wymaganym dużym stopniu rozdzielczości, należy raczej stosować odpowiednio szybkie sterowniki graficzne i dobre monitory kolorowe.

Rejestracja i odtwarzanie dźwięku na dobrym poziomie wymaga także odpowiednich, montowanych w komputerze, kart rozszerzających (np. *Soud Galaxy NX Card*). W zależności od potrzeb ilustracje dźwiękowe mogą podlegać syntezie mono – lub stereofonicznej. Rejestrowany dźwięk można poddawać modyfikacjom, np. zmianom rytmu, tonacji itp., zgodnie z ewentualnymi wymogami.

Stanowisko pracy wyposażone w wyżej wymienione urządzenia powinno mieć do dyspozycji sprzęt w następującej konfiguracji:

- komputer z procesorem minimum 80386DX/40 MHz,
- monitor kolorowy *VGA(SVGA)*,
- szybką kartę grafiki z możliwością wyświetlania co najmniej 256 kolorów i rozdzielczością 800 × 600 (np. karta *PARADISE* z akceleratorem firmy *WD*),
- dysk twardy o pojemności ok. 200 MB.

Jak wiadomo, kolor pojedynczego piksla na ekranie monitora jest wyznaczany za pomocą mieszania kolorów podstawowych metodą addytywną, znaną (choć już nie stosowaną) w fotografii barwnej. Mieszając ze sobą trzy barwy zasadnicze można otrzymać inne barwy, np. zielona z niebieską daje kolor niebieskozielony, niebieska z czerwoną daje purpurę i tak dalej. W praktyce nie jest to takie proste, ponieważ nie występują barwy zupełnie „czyste", a więc ich mieszanie daje zawsze różne odcienie. W naturze kolory powstają przez odejmowanie barw, co nazywa się procesem subtraktywnym. Przedmioty barwne pochłaniają część widma słonecznego odbijając resztę i ta odbita część decyduje o tym, co nazywamy barwą danego przedmiotu. Informacje te są niezbędne przy fotografowaniu sceny i jej późniejszej cyfrowej obróbce. Nieznajomość tych prawideł może spowodować, że przeniesione po obróbce do aplikacji docelowych obrazy nie będą mieć zadowalającej jakości (mogą być zamazane lub oddawać fałszywe kolory).

Format *FLM* umożliwia ustalenie nasycenia każdego z kolorów podstawowych za pomocą jednego bajtu (jeden bajt dla koloru *R*, jeden dla koloru *B* i jeden dla koloru *G*). Innymi słowy, format *FLM* umożliwia zdekodowanie informacji o 24-bitowym kolorze. Każdy z kolorów może przyjąć dowolną wartość z zakresu 0–255, co w rezultacie daje do 255∗255∗255 = 16 777 216 różnych kolorów. Format *FLM* zachowuje więc wierność kolorów największą z możliwych do uzyskania. Dysponując plikiem w formacie *FLM* można od tej chwili wielokrotnie dokonywać konwersji na inne, wymienione wcześniej formaty graficzne. Stosując takie zabiegi można jednak otrzymywać niepowtarzalne wyniki, gdyż końcowy efekt zależy już od aktualnie ustalonego trybu pracy karty wideo, a także od typu wybranej konwersji graficznej. Uzyskiwanie zadowalających efektów zależy więc w pierwszym rzędzie od standardu wmontowanej karty oraz jakości zastosowanego monitora. Należy więc stosować te same palety kolorów podczas konwersji z formatu *FLM* i w aplikacjach docelowych, wykorzystujących przygotowane obrazy.

Tabela 1

Liczba bitów na jeden piksel	Liczba kolorów
1	obraz mono
4	16
6	64
8	256
15	(32K) 32 768
16	(64K) 65 536
24	16 777 216 (*True Color*)

Liczba kolorów w palecie, czyli całkowita liczba kolorów możliwych do wykorzystania, zależy od liczby bitów przeznaczonych do zakodowania informacji o kolorze oraz od rozdzielczości przetworników cyfrowo-analogowych (*C/A*), używanych do konwersji wektora cyfrowego na zespolony sygnał *RBG*, sterujący działami elektronowymi monitora. Na przykład:

3 × 6 = 18 bitów – paleta 256 K kolorów
3 × 8 = 24 bity – paleta 16 384 K kolorów (*True Color*)

Z kolei liczba dostępnych jednocześnie na ekranie kolorów jest wyznaczona liczbą bitów użytych do opisu koloru pojedynczego piksla. Zależności te przedstawia tabela 1.

W standardzie karty *VGA* i *SVGA* kolor jest kodowany według modelu *RGB*. Oznacza to, że numer koloru nie wskazuje bezpośrednio barwy punktu, lecz jest tylko wskaźnikiem (indeksem) do tabeli opisującej kolory palety. W zależności od jej zawartości, kolor – np. o numerze 7 – może być reprodukowany na ekranie jako czerwony, żółty lub dowolnie inny.

Liczba kolorów dostępnych na ekranie monitora zmienia się wraz ze zmianą rozdzielczości (trybu pracy), sterującej monitorem karty wideo. Dla wielu różnych kart parametry te mogą różnić się znacznie i dlatego należy szczególnie ostrożnie dokonywać ich zakupów. Przykładowe rozdzielczości karty *Paradise* z akceleratorem firmy *Western Digital* prezentuje tabela 2.

Czynnikiem wpływającym na jakość tworzonego i przenoszonego do innych aplikacji obiektu są jego wymiary. Zaleca się,

Tabela 2

Liczba kolorów	640 × 480	800 × 600	1024 × 768	1280 × 960	1280 × 1024
16	X	X	X	X	X
256	X	X	X		
32 768	X	X			
65 536	X	X			

aby rejestracja obrazu z kamery była dokonywana w skali, w której będzie ona umieszczona w aplikacji docelowej. Wprawdzie przeskalowania można dokonywać później, lecz wiąże się to przeważnie z powstawaniem trudnych do korygowania zniekształceń geometrycznych obrazu w aplikacji docelowej.

W przypadku rejestracji ilustracji dźwiękowych, wymagania sprzętowe mogą być mniejsze. Wystarczy tu, aby stanowisko komputerowe było wyposażone w kartę mulimedialną do zapisu i odtwarzania dźwięków wraz z odpowiednim oprogramowaniem. Jakość karty wideo i rodzaj stosowanego monitora mają tutaj charakter drugoplanowy. Zasady osadzania obiektów dźwiękowych w bazach danych są podobne do powyżej opisanych. W takim przypadku w miejscu ich lokowania, umieszczane są ikony (np. w postaci mikrofonu), oznaczające występowanie obiektu dźwiękowego.

LITERATURA

[1] Accelerator card for Windows. Paradise Western Digital. User Manual, 1992
[2] Marciniak A.: Turbo Pascal 5.5. Wydawnictwo Nakom, 1991
[3] Pavlidis R.: Grafika i przetwarzanie obrazów. WNT, 1987
[4] Screen Machine Manual. Fast Electronic GmbH, 1992
[5] Sound Galaxy NX Card. User Manual, 1993
[6] Zieliński M., Górbiel A.: Windows 3.1. dla dociekliwych. Wydawnictwo PLJ, 1992.

Piąta Jubileuszowa Międzynarodowa Górska Szkoła PTI Szczyrk
21-25.06.1993 r.

MIROSŁAW WIECZOREK
Politechnika Śląska
Gliwice

Podstawowe zagadnienia programowania zorientowanego obiektowo – polimorfizm i dziedziczenie klas w języku C++

Język C++, opracowany na początku lat osiemdziesiątych przez Bjarne Stroustrupa jest nowoczesnym językiem programowania ogólnego przeznaczenia. Z paroma małymi wyjątkami, C++ jest nadzbiorem języka C, co oznacza, że prawie wszystko, co jest dozwolone w C, jest dozwolone również w C++. Oczywiście w C++ wprowadzono wiele rozszerzeń usprawniających pracę programisty oraz mechanizmy umożliwiające programowanie zorientowane obiektowo. I właśnie ta możliwość programowania obiektowego w połączeniu z efektywnością programowania odziedziczoną po języku C sprawiła, że C++ stał się nowym językiem programowania i coraz bardziej zyskuje na popularności.

W tradycyjnym, proceduralnym podejściu do programowania, program opisuje serię czynności, jakie mają być wykonane, aby osiągnąć pożądane wyniki, czyli po prostu algorytm. Natomiast w podejściu obiektowym program opisuje system czy też zbiór wzajemnie oddziaływujących na siebie obiektów, w pewien sposób hierarchicznie zorganizowanych i wymieniających między sobą informacje.

Programowanie obiektowe opiera się zasadniczo na kilku kluczowych koncepcjach. Są to:

- abstrakcja danych i procedur, czyli oderwanie się od szczegółów implementacji struktur danych i procedur a skoncentrowanie się na istocie rzeczy;
- koncepcja klasy, czyli definiowanego przez programistę nowego typu, określonego przez pewien zbiór wartości i operacji, które na tym typie mogą być wykonywane; z koncepcją klasy wiąże się ściśle pojęcie hierarchii klas;
- hermetyzacja, czyli ukrywanie szczegółów implementacyjnych i rozróżnienie między częścią implementacyjną a definicyjną;
- dziedziczenie kodu, czyli możliwość wykorzystania już istniejących definicji klas lub całej ich hierarchicznej struktury.

Oczywiście nic nie stoi na przeszkodzie pisania programów opartych na tych założeniach, przy wykorzystaniu dowolnego języka programowania, jednak tylko w przypadku języków zorientowanych obiektowo koncepcje te są elementem języka i są ściśle związane z jego opisem.

Język C++ stał się językiem obiektowym dzięki wprowadzeniu do jego definicji koncepcji klasy zapewniającej wszystkie wymienione powyżej istotne cechy wyróżniające programowanie obiektowe. W dalszej części zostanie dokładnie przedstawiony sposób definicji klasy w języku C++, przykłady konstruowania hierarchii klas wraz z przykładami dziedziczenia kodu.

Oprócz tego zostanie zaprezentowany sposób definicji i użycia funkcji wirtualnych dających możliwość tzw. dynamicznego łączenia (ang. *dynamic binding*) oraz stosowania funkcji polimorficznych.

Pojęcie klasy

W języku C++ klasa jest typem danych definiowanym przez użytkownika. Deklaracja klasy określa reprezentację i zbiór wartości obiektu danej klasy oraz zbiór operacji, które można na obiektach danej klasy przeprowadzić. Jak wspomniano, klasy mogą służyć do wprowadzania nowych typów, np. klasa **Drzewo_binarne** czy **Stos**, oraz do definiowania abstrakcji, które w naturalny sposób nie odpowiadają predefiniowanym typom z języka C++, np. klasa **Okno**, klasa **Student** czy też klasa **Dom**. Liczba rodzajów klas jest ograniczona tylko inwencją programisty.

W języku C++ z klasą są związane cztery atrybuty:

- zbiór pól składowych określających reprezentację klasy, przy czym liczba pól może wynosić zero;
- zbiór funkcji składowych określających zbiór operacji, które można wykonywać na obiekcie danej klasy; w klasie nie jest konieczne deklarowanie funkcji składowych;
- widoczność elementów klasy, pozwalająca na hermetyzację i ukrywanie reprezentacji klasy;
- nazwa klasy, służąca jako identyfikator typu i mogąca występować wszędzie tam, gdzie mogą wystąpić identyfikatory typów predefiniowanych.

Definicja klasy

Definicja klasy składa się z dwóch części: nagłówka klasy złożonego ze słowa kluczowego **class** oraz nazwy klasy i ciała klasy objętego nawiasami klamrowymi. Po zamykającym nawiasie klamrowym musi wystąpić średnik lub lista deklaracji. Na przykład:

```
class okno
{
   int xl, yl, rozmiar_x,
                    rozmiar_y ;
   char *bufor ;
   int ramka ;
} ;
```

Zdefiniowaliśmy klasę o nazwie **okno** zawierającą sześć pól: cztery opisujące położenie okna i jego rozmiar, jedno zawierające adres obszaru przechowującego zawartość okna oraz jedno określające rodzaj ramki. W ten sposób opisaliśmy reprezentację klasy.

Tak zdefiniowanej klasy i jej nazwy możemy używać wszędzie tam, gdzie używamy typów predefiniowanych. Na przykład:

```
okno o1, o2 ;
        // dwa obiekty klasy okno
okno tablica_okien[ 20 ] ;
        // tablica 20 elementów
        // klasy okno
int s = sizeof( okno ) ;
        // przypisanie zmiennej s
        // rozmiaru pamięci
        // zajmowanej przez obiekt
```

Zdefiniujemy teraz klasę zawierającą oprócz pól składowych również następujący zestaw operacji:

```
class lista
{
   int wartosc ;
   lista *nast ;
   void dodaj( int war ) ;
   void drukuj() ;
} ;
```

Klasa **lista**, oprócz pól składowych zawierających przechowywaną wartość oraz wskaźnik do następnego elementu w liście, zawiera również zbiór, w tym wypadku dwóch operacji, które można wykonywać na obiekcie klasy **lista**. Operacje te realizują dodawanie nowego elementu oraz drukowanie zawartości listy. W tej chwili nie będziemy się zajmować implementacją tych operacji.

Do tej pory mieliśmy do czynienia z trzema atrybutami klasy, nic nie mówiliśmy o widoczności i dostępności składowych klasy, co niestety jest bardzo istotne. Tak istotne, że poprzednio zdefiniowane klasy są w zasadzie bezużyteczne.

Widoczność każdej ze składowych pewnej klasy można określić na trzy sposoby: składowa klasy może być publiczna, prywatna lub zabezpieczona. Co to znaczy?

Otóż dostęp do składowych publicznych nie jest w żaden sposób ograniczany, tzn. mogą być one używane (są widoczne) w dowolnym miejscu w programie, w którym jest widoczna deklaracja klasy. Natomiast w przypadku składowych prywatnych, są one widoczne tylko w obrębie funkcji składowych danej klasy (dla ścisłości również w funkcjach zaprzyjaźnionych). Pole, czy też składowa zabezpieczona, ma podobne zasady widoczności, jak składowa prywatna z tym, że składowa zabezpieczona jest również widoczna w funkcjach składowych klas pochodnych. Co to dokładnie znaczy, zostanie omówione w dalszej części artykułu.

Dlaczego więc poprzednie definicje są w zasadzie bezużyteczne? Mając zdefiniowaną klasę **lista** zadeklarujemy obiekt o nazwie l1:

```
lista l1 ;
```

Dostęp do składowych klasy (pól i funkcji) odbywa się identycznie jak w przypadku struktur i unii znanych z języka C, czyli podaje się identyfikator zmiennej (w naszym przypadku będzie to reprezentant naszej klasy, czyli obiekt), a następnie po kropce – identyfikator składowej (pola albo funkcji). Na przykład

```
l1.wart = 12 ;
```

albo

```
l1.drukuj() ;
```

W naszym przypadku odwołania te będą jednak nieprawidłowe, ponieważ w klasie lista **wszystkie** składowe są **prywatne**, ponieważ przyjmuje się, że bez wyszczególnionego specyfikatora dostępu wszystkie składowe klasy są prywatne.

Zmieńmy teraz definicję naszej klasy **lista** w ten sposób, aby była bardziej przydatna:

```
class lista
{
  public:
     int wart ;    // pole publiczne
     void dodaj( int war ) ;
                   // funkcja publiczna
     void drukuj() ;
                   // funkcja publiczna
  private:
     lista *nast ; // pole prywatne
} ;
```

Specyfikator widoczności składa się z jednego z trzech słów kluczowych: **public:**, **private:** oraz **protected:**, i powoduje zmianę domyślnej widoczności składowych klasy. Zmiana ta rozciąga się od miejsca wystąpienia specyfikatora aż do nawiasu klamrowego kończącego definicję klasy, albo do wystąpienia kolejnego specyfikatora. W ramach jednej definicji klasy może wystąpić dowolna liczba specyfikatorów i w dowolnej kolejności.

Rozważmy następujące odwołania do pól składowych:

```
lista l1, l2 ;

l1.wart = 123 ;      // o'k
l1.dodaj( 1 ) ;      // o'k
l1.drukuj() ;        // o'k
l1.nast = &l2 ; // blad, pole nast
            //jest polem prywatnym
```

Właściwy dobór specyfikatorów widoczności umożliwia właśnie hermetyzację i ukrywanie szczegółów implementacyjnych. To, czego nie widać, czyli składowe prywatne, można w dowolny sposób zmieniać bez potrzeby zmian w później stworzonym kodzie. Oczywiście, pod warunkiem że zmiany te będą sensowne. Zalecane jest również, aby dostęp do pól składowych był realizowany przez zbiór odpowiednich funkcji składowych, a same pola były prywatne. Uniemożliwia to przypisanie polom składowym niepożądanych wartości.

Mówiąc o definicji klasy, nie od rzeczy będzie wspomnieć o specjalnych funkcjach składowych, jakimi są **konstruktory** i **destruktory.**

Konstruktor jest funkcją składową o nazwie identycznej z nazwą klasy i jest niejawnie wywoływany w momencie tworzenia obiektu danej klasy. Konstruktor w najprostszym przypadku ma zadanie inicjalizacji pól składowych klasy określonymi wartościami. Dla bardziej skomplikowanych klas może to być jeszcze otwieranie plików (np. klasa opisująca plik zorganizowany w strukturę B-drzewa) oraz realizowanie przydziału pamięci (np. klasa opisująca tablicę dynamiczną). U nas będzie to po prostu nadanie polu **nast** wartości 0, co oznacza pusty wskaźnik. Rozszerzmy teraz naszą klasę **lista** o taki właśnie konstruktor:

```
class lista
{
  public:
      lista() { nast = NULL ; }
    int wart ;
      void dodaj( int war ) ;
      void drukuj() ;
  private:
      lista *nast ;
                  // pole prywatne
} ;
```

Konstruktor nie może zwracać żadnej wartości i **musi** być niestatyczną funkcją składową danej klasy. Warto zauważyć, że w ciele definicji klasy pojawiła się definicja funkcji. Jest to całkowicie poprawne. Definicja funkcji składowej może znajdować się albo w ciele definicji klasy (jak w ostatnim przykładzie), albo poza nią. Jednak w tym drugim przypadku w definicji klasy powinna pojawić się deklaracja prototypowa (tak jak we wszystkich poprzednich przykładach).

Destruktor jest funkcją, która jest niejawnie wywoływana w momencie niszczenia obiektu danej klasy. Dla obiektów statycznych będzie to następowało po zakończeniu działania programu. Dla obiektów automatycznych odbędzie się to w momencie opuszczenia bloku, w którym obiekt został utworzony. Nazwa destruktora odpowiada nazwie klasy poprzedzonej znakiem tyldy (~). Do naszej definicji klasy dopiszemy teraz destruktor:

```
class lista
{
            // poprzednie składowe
      ~lista() ;
   // ciało funkcji w innym miejscu
  private:
      lista *nast ;
};
```

To, co destruktor ma wykonywać, zależy tylko i wyłącznie od programisty, może to być np. zamknięcie pliku czy też zwolnienie przydzielonej obiektowi pamięci. W naszym przypadku destruktor powinien zwolnić pamięć przydzielaną kolejnym elementom w funkcji **dodaj ( ).**

Dziedziczenie klas

Dziedziczenie klas jest, jak już wspomniano, jedną z kluczowych koncepcji programowania zorientowanego obiektowo. Umożliwia wykorzystanie już istniejącego kodu oraz, co wydaje się ważniejsze, wykorzystanie funkcji polimorficznych. Rozpatrzmy następującą definicję klasy:

```
class osoba
{
  public:
    char nazwisko[ 30 ] ;
} ;
```

Dla uproszczenia pominiemy funkcje realizujące dostęp do pól klasy, a same pola uczynimy publicznymi. Mamy klasę o nazwie **osoba**, która to klasa opisuje po prostu określoną osobę. Powiedzmy, że chcemy teraz uczynić z tej osoby pracownika określonego zakładu. Zamiast implementować drugą klasę, która będzie zawierać informacje o nazwisku i, powiedzmy, o wynagrodzeniu, możemy wykorzystać już istniejącą klasę **osoba** i utworzyć nową klasę przez dziedziczenie.

```
class pracownik : public osoba
{
  public:
    long pensja ;
} ;
```

Przy takiej definicji klasy **pracownik** mamy do czynienia z klasą pochodną **(pracownik)**, która dziedziczy po klasie **osoba**, będącą klasą bazową. W czasie dziedziczenia, wszystkie składowe klasy bazowej wchodzą do klasy pochodnej. Można nawet powiedzieć, że klasa pochodna jest klasą bazową, ewentualnie rozszerzoną o nowe możliwości. W naszym przykładzie **pracownik** jest jednocześnie osobą. Natomiast **osoba** nie jest **pracownikiem**. I nie należy tego rozumieć, że klasa **pracownik** zawiera pole typu **osoba**. Określenie tego, czy jakaś klasa jest pochodną pewnej klasy bazowej, jest zapisane w nagłówku definicji klasy.

Po słowie kluczowym **class** występuje nazwa klasy, a po niej dwukropek. Po dwukropku natomiast występuje **lista** klas bazowych (w naszym przypadku jest to jedna klasa). Nazwy klas bazowych są rozdzielane przecinkami. Wynika z tego, że klasa może dziedziczyć właściwości niekoniecznie po jednej klasie bazowej. Mamy wtedy do czynienia z dziedziczeniem wielobazowym, którym jednak nie będziemy się na razie zajmować. Przed nazwą klasy, tak jak w naszym przykładzie, może wystąpić specyfikator dostępu. Specyfikator **public** oznacza, że klasa bazowa jest upubliczniona. Brak takiego specyfikatora powoduje, że domyślnie klasa bazowa jest uprywatniona. Co to wszystko oznacza? W przypadku uprywatnienia klasy bazowej, wszystkie pola klasy bazowej stają się, z punktu widzenia klasy pochodnej, polami prywatnymi, natomiast w przypadku upublicznienia pola klasy bazowej zachowują własny tryb widoczności.

Wspomniana już zależność między klasą bazową a klasą pochodną powoduje to, że obiektowi klasy bazowej można bez żadnych konwersji przypisać obiekt klasy pochodnej. Nigdy jednak odwrotnie i tylko wtedy, gdy klasa bazowa jest upubliczniona:

```
osoba os1, os2 ;
pracownik p1, p2 ;

os1 = os2 ;          // o'k
p1  = p2  ;          // o'k
os1 = p1  ;          // o'k
p2  = os2 ;          // błąd !
```

Drugą interesującą sprawą jest możliwość przypisywania wskaźnikowi na obiekt klasy bazowej adresu (lub wskaźnika) obiektu klasy pochodnej. Tak jak w poprzednim przypadku, klasa pochodna musi mieć upublicznioną klasę bazową, a przypisanie odbywa się bez specjalnych operatorów konwersji. Na przykład:

```
osoba os, *pos ;
pracownik pr, *ppr ;

pos = &os ;        // o'k
ppr = &pr ;        // o'k
pos = &pr ;        // o'k
pos = ppr ;        // o'k
ppr = &os ;        // błąd
```

Wprowadzając następną w hierarchii klasę możemy napisać:

```
class kierownik : public pracownik
{
  public:
    char wydział ;
    long dodatek ;
} ;
kierownik k ;
osoba os ;
os = k ;                    // o'k
```

Powyższy zapis jest całkowicie poprawny, ponieważ kierownik **jest** pracownikiem, a co za tym idzie, jest również osobą.

```
#include <iostream.h>
#include <string.h>
class osoba
{
  public:
     char nazwisko[ 30 ] ;
     virtual zarobek()
     {
        cout << nazwisko << " nie jest pracownikiem\n" ;
     }
} ;
class pracownik : public osoba
{
  public:
     long pensja ;
     zarobek()
     {
        cout << nazwisko << " zarabia " << pensja << endl ;
     }
} ;
class kierownik : public pracownik
{
  public:
     char wydział ;
     long dodatek ;
     zarobek()
     {
        cout << nazwisko<< " jest szefem i zarabia "
             << pensja+dodatek << endl;
     }
} ;
main()
{
  osoba *pracownicy[ 20 ] ;
  int liczba_osób  = 0 ;

  osoba os ;
  strcpy( os.nazwisko, "Kowalski" ) ;

  pracownik pr ;
  strcpy( pr.nazwisko, "Nowak" ) ; pr.pensja = 10000 ;

  kierownik kier ;
  strcpy( kier.nazwisko, "Berda" ) ;
  kier.pensja = 100000 ; kier.dodatek = 20000 ;

  pracownicy[ liczba_osób++ ] = &pr ;
  pracownicy[ liczba_osób++ ] = &os ;
  pracownicy[ liczba_osób++ ] = &kier ;

  for( int i = 0 ; i < liczba_osób ; i++ )
     pracownicy[ i ]->zarobek() ;
}
```

Pewne wątpliwości może budzić mechanizm takiego przypisania, który w rzeczywistości jest bardzo prosty. Polom obiektu **os** klasy osoba zostaną nadane wartości odpowiednich pól z obiektu **k** typu **kierownik**.

Funkcje polimorficzne

Na początek rozpatrzmy prosty przykład:

```
#include <iostream.h>
class A
{
  public:
    virtual void pisz()
    {
      cout << "Klasa A\n" ;
    }
} ;
class B : public A
{
  public:
    void pisz()
    {
      cout << "Klasa B\n" ;
    }
} ;
A a, *pa ;
B b, *pb ;
main()
{
  pb.pisz()
        // drukuje napis "Klasa B"
  pa = &b
  pa->pisz() ;   // również drukuje
                 // napis "Klasa B" !!
}
```

Dlaczego tak się dzieje? Przecież wskaźnik **pa** jest typu **A*pa** więc zapis **pa → pisz( )** powinien sugerować wywołanie funkcji pisz( ) należącej do klasy A. Dlaczego więc nie wywołuje? W tym przypadku mamy bowiem do czynienia z bardzo istotnym udogodnieniem C++ – funkcjami wirtualnymi czy też polimorficznymi. Polimorficznymi dlatego, że funkcja o nazwie **pisz( )** zachowuje się różnie dla obiektów o różnych typach, nawet jeśli typ wskaźnika mówi zupełnie coś innego.

W C++ deklaracja funkcji wirtualnej zaczyna się od słowa kluczowego **virtual**, po którym następuje normalna deklaracja funkcji, to znaczy typ wyniku, identyfikator funkcji oraz lista parametrów. Funkcja taka powinna być niestatyczną funkcją składową klasy. Taka deklaracja tworzy w zasadzie rodzinę funkcji polimorficznych, ponieważ każda funkcja zadeklarowana w klasie pochodnej i mająca taką samą nazwę, ten sam typ zwracanego rezultatu oraz taką samą listę parametrów, jest funkcją wirtualną. Niezależnie od tego czy została ona zadeklarowana ze słowem kluczowym **virtual**, czy też bez niego. Można również powiedzieć, że w ciągu klas funkcja staje się wirtualna od momentu zadeklarowania jej jako wirtualnej, gdyż wcześniej jest to zwykła funkcja składowa. Gdyby w poprzednim przykładzie jako wirtualną zadeklarować funkcję w klasie **B**, a nie **A**, na ekranie pojawiłyby się napisy:

```
Klasa A
Klasa B
```

Rozpatrzmy teraz bardzo prosty przykład użycia funkcji wirtualnych. W przykładzie tym tworzymy tablicę zawierającą informacje o trzech osobach, a następnie drukujemy informacje o ich wynagrodzeniu. Wynikiem działania naszego programu jest następujący wydruk:

```
Nowak zarabia 10000
Kowalski nie jest pracownikiem
Berda jest szefem i zarabia 120000
```

Najbardziej interesujące w tym programie jest to, że w pętli drukującej zarobek w żaden sposób nie sprawdzamy, jaką funkcję drukującą należy wywołać. To wszystko robi za nas kompilator. Pisząc ten program w C lub w innym języku, umożliwiającym stosowania funkcji wirtualnych, musielibyśmy wprowadzić w definicjach klas dodatkowe pola identyfikujące typ obiektu, a następnie badać wartość tego pola w pętli drukującej informacje, np. za pomocą rozbudowanej instrukcji **switch**. Byłoby to szczególnie kłopotliwe, jeśli operowalibyśmy na dużej liczbie typów obiektów.

LITERATURA

[1] Lippman S.B.: C++ Primer, 2-nd Edition. Addison-Wesley 1991

[2] Stroustrup B.: The C++ Programming Language. Addison-Wesley 1987

[3] Stroustrup B., Ellis M.: The Annotated C++ Reference Manual. Addison-Wesley 1990.

Nowe książki

WYDAWNICTWA NAUKOWO-TECHNICZNE

Fritz H. Grupe: Lotus 1–2–3 dla początkujących, wersja 3.0. WNT, Warszawa 1993 r., wyd. 1, s. 136. ISBN 83-204-1570-5

Wstępnie napisany podręcznik pakietu kalkulacyjnego Lotus 1–2–3, wersja 3.0 na poziomie elementarnym uczy posługiwania się pakietem metodą rozwiązywania typowych problemów w połączeniu z praktycznymi ćwiczeniami przy komputerze. Oprócz wiadomości podstawowych omawia zagadnienia: projektowania arkusza kalkulacyjnego i operowania nim, posługiwania się wyrażeniami i funkcjami, tworzenia wykresów oraz działania na plikach wieloarkuszowych.

Książka jest przeznaczona dla szerokiego kręgu czytelników mających w swojej praktyce zawodowej do czynienia z opracowywaniem danych liczbowych (księgowość, kosztorysowanie, sprawozdawczość itp.). Może być przydatna na kursach zastosowań komputerów lub do samodzielnej nauki indywidualnej.

Henryk Małysiak, Bolesław Pochopień, Eugeniusz Wróbel: Procesory arytmetyczne. WNT, Warszawa 1993 r., wyd. 1, s. 112. ISBN 83-204-1499-7

W książce omówiono koprocesory arytmetyczne 8087, 80287 i 80387, będące istotnymi elementami mikrokomputerów serii IBM PC. Podano w niej listę rozkazów i szczegółowo przedstawiono sposób ich działania. Zamieszczono również przykładowy program.

Książka jest przeznaczona dla elektroników i informatyków zainteresowanych budową mikrokomputerów.

Jacek Kudrewicz: Fraktale i chaos. WNT, Warszawa 1993 r., wyd. 1, s. 120. ISBN 83-204-1607-8

Książka jest skróconym wykładem o fraktalach i chaosie. Przedstawiono w niej podstawowe pojęcia matematyczne związane z chaotyczną dynamiką, zbiorami niezmienniczymi, dziwnymi atraktorami i repilerami, zbiorami Julii i Mandelbrota oraz układem iterowanych odwzorowań. Przytoczono wiele przykładów i ilustracji graficznych.

Claude Delannoy: Ćwiczenia z języka C. WNT, Warszawa 1993 r., wyd. 1, s. 126. ISBN 83-204-1612-4

Książka składa się z dwóch części. W pierwszej proponuje się ćwiczenia, które należy rozwiązywać w fazie opanowywania podstaw języka C. Część ta składa się z siedmiu rozdziałów, opartych na układzie typowego podręcznika języka C: operatory i wyrażenia, wejście-wyjście, instrukcje sterujące, funkcje, tablice i wskaźniki, ciągi znaków, struktury. Każdy rozdział zawiera ćwiczenia mające ułatwić bezpośrednie przyswojenie materiału oraz niewielkie ćwiczenia, nie zawierające trudniejszych problemów algorytmicznych, korzystające z pojęć wyjaśnionych w poprzednich rozdziałach.

Druga część zawiera problemy bardziej skomplikowane i różnorodne, zarówno jeśli chodzi o zakres stosowanych technik programowania, jak i tematykę. Należy je rozwiązywać dopiero po opanowaniu podstaw języka, to jest zagadnień omawianych w części pierwszej. Wszystkie programy są napisane w języku C zgodnym z normą ANSI, dzięki czemu mogą być uruchamiane na większości współczesnych komputerów.

Claude Delannoy: Ćwiczenia z języka C++ – programowanie obiektowe. WNT, Warszawa 1993 r., wyd. 1, s. 178. ISBN 83-204-1611-6

Książka umożliwia opanowanie programowania obiektowego w języku C++. Przedstawiono w niej ćwiczenia o wzrastającym stopniu trudności oraz ich elementarne rozwiązania. Układ książki odpowiada typowemu podręcznikowi C++. Na początku każdego rozdziału przedstawiono skrót wiadomości, niezbędnych do rozwiązania podanych następnie ćwiczeń. Ćwiczenia mają różny stopień trudności – od bezpośredniego zastosowania podanych wiadomości, aż do realizacji złożonych klas, takich jak: zbiory, dynamiczne wektory, listy, tablice bitów, ciągi znaków, stosy, liczby zespolone. Przy okazji pokazano podstawowe techniki programowania, np.: realizację iteratora i zarządzanie licznikiem odwołań. Prócz składni języka C++ z książki można nauczyć się także projektowania i tworzenia własnych klas.

Marc j.Rochkind: Programowanie w systemie Unix dla zaawansowanych., WNT, Warszawa 1993 r., wyd. 1, s. 312. ISBN 83-204-1596-9

Autor podręcznika wyjaśnia w sposób przejrzysty i systematyczny, jak stosować w programach funkcje systemowe Unixa. Ułatwia doświadczonym programistom zrozumienie funkcji systemowych, tak aby używali ich mądrzej i bez straty na przenośności oprogramowania. Korzystając z własnego doświadczenia pokazuje kiedy i dlaczego stosuje się funkcje systemowe.

W książce omówiono pięć najważniejszych wersji systemu Unix:

- System V oferowany przez firmę AT & T;
- System III, będący podstawą wielu bieżących implementacji;
- Wersję 7, czyli Unix Siódmego Wydania, pochodzący z firmy Bell Laboratory;
- Berkeley 4.2BSD powstały w Uniwersytecie Kalifornijskim;
- Xenix, popularny system mikrokomputerowy firmy Microsoft.

Prezentowane wersje systemowe dotyczą operacji wejścia-wyjścia na plikach i terminalach, procesów (wielozadaniowości), sygnałów oraz administrowania systemem.

Jest to „zaawansowana" książka o Unixie. Czytelnik powinien znać język programowania C oraz podstawy Unixa.

Piąta Jubileuszowa Międzynarodowa Górska Szkoła PTI Szczyrk
21-25.06.1993 r.

ARTUR KASPRZYK
Uniwersytet Wrocławski

Obiektowo zorientowane analiza i projektowanie (2)

W pierwszej części artykułu omówiono etapowo zorientowaną analizę. Część druga zostanie poświęcona projektowaniu systemu.

Projektowanie systemu

Projektowanie systemu jest pierwszym krokiem w stronę *jak to zrobić*. W trakcie jego trwania podejmuje się decyzje dotyczące organizacji systemu (podział na podsystemy) oraz przydziału podsystemów do oddzielnych jednostek sprzętowych i programowych. Organizacja całego systemu za pomocą podsystemów (ang. *subsystem*) jest nazywana architekturą systemu (ang. *system architecture*). Podział problemu na mniejsze składowe pozwala na dalszą, niezależną pracę nad każdym z nich. W trakcie projektowania systemu powinno się wziąć pod uwagę następujące czynniki:

- podział systemu na podsystemy,
- identyfikację współbieżności wewnątrz systemu,
- przydział podsystemom osobnych procesorów i procesów,
- ustalenie sposobu dostępu do wspólnych zasobów,
- ustalenie metody implementacji sterowania w systemie,
- obsługę warunków brzegowych.

W zależności od specyfiki projektowanego systemu, niektóre czynniki będą wymagały bardzo uważnej analizy, podczas gdy część z nich będzie można w rozważaniach pominąć.

Podział systemu na podsystemy

Pierwszym krokiem przy projektowaniu systemu powinien być jego podział na podsystemy dokonany w taki sposób, by każda nowo utworzona składowa tworzyła spójną pod jakimś względem całość. Za kryterium podziału przyjęło się brać usługi, jakie podsystem ma wykonywać. Jeżeli projektowany jest system obsługi klienta w banku, podsystemy mogłyby zawierać interfejs graficzny, obsługę kont, obsługę bankomatów itd. Każdy podsystem powinien mieć precyzyjnie zdefiniowany interfejs z resztą systemu. Dzięki takiemu rozwiązaniu możliwe będzie dokonywanie ewentualnych zmian wewnątrz podsystemu, bez ingerencji w pozostałą część systemu. Wobec tego, każdy podsystem będzie mógł być tworzony niezależnie od innych.

Podział systemu może być dokonywany na dwa sposoby. Pierwszy polega na podziale na „poziome" warstwy. Każda warstwa jest spojrzeniem na problem „z innej wysokości". Warstwę górną buduje się albo na podstawie tylko warstwy leżącej bezpośrednio pod nią – wówczas mamy do czynienia z architekturą zamkniętą (ang. *closed architecture*), albo na podstawie wszystkich warstw leżących pod nią – taką architekturę nazywa się otwartą (ang. *open architecture*). W przypadku architektury zamkniętej, odwołania w warstwie górnej odnoszą się tylko do jednej warstwy. Oznacza to, że ewentualne zmiany w którejś z warstw spowodują konieczność uaktualnienia jedynie warstwy leżącej bezpośrednio pod nią. Dzięki temu łatwo można przenosić systemy na różne platformy sprzętowe, zmieniając tylko najniższą warstwę. Rozwiązanie takie ma jednak tę wadę, iż występuje duża redundancja informacji. Na każdym poziomie należy przedefiniowywać wszystkie obiekty poprzedniej warstwy. Architektura otwarta zmniejsza redundancje informacji. Z drugiej strony, zmiany w którejś z warstw mogą mieć wpływ na każdą inną warstwę leżącą powyżej, co może sprawić sporo kłopotów w przypadku awarii lub ewentualnej przyszłej rozbudowy systemu.

Alternatywą dla podziału na poziome warstwy jest podział na pionowe partycje. Każda partycja świadczy określone usługi. Jednocześnie partycje są bardzo słabo ze sobą powiązane, zmniejszając w ten sposób liczbę wysyłanych do siebie komunikatów. Jak zwykle bywa w życiu, praktyka pokazała, iż najlepszym rozwiązaniem jest połączenie obydwu metod. Tworzy się więc systemy „poziomo-pionowe".

Związki między podsystemami mogą być dwojakiego rodzaju: klient-dostawca (ang. *client-supplier*) lub równoprawny (ang. *peer-to-peer*). W pierwszym przypadku klient żąda od dostawcy wykonania określonej usługi. Dostawca wykonuje ją dając określoną odpowiedź. Dostawca musi wcześniej znać interfejs dostawcy, podczas gdy znajomość interfejsu klienta przez użytkownika jest zbędna. Dzięki temu, takie połączenie jest proste do wykonania i pielęgnacji. W przypadku związku równoprawnego, obydwa podsystemy mogą żądać od siebie wykonania usług. Każdy z nich powinien znać sposób komunikacji ze sobą.

Na zakończenie etapu należy wybrać odpowiednią topologię systemu. Sposób komunikacji podsystemów może być różny. Zawsze należy wybierać topologię najprostszą tzn. taką, aby jak najbardziej zmniejszyć liczbę wzajemnych odwołań między podsystemami.

Rozpoznawanie współbieżności

Do tej pory wszystkie obiekty występujące w systemie były traktowane jako niezależne. Działo się tak, gdyż modelowane obiekty miały odpowiadać obiektom rzeczywistym. Niestety wciśnięcie Rzeczywistości w sztywne ramy zer i jedynek nie może obejść się bez skutków ubocznych. W praktyce obiekty muszą dzielić się ograniczonymi zasobami wspólnych dóbr. Dlatego też należy wydobyć te obiekty, które **muszą** pracować równolegle oraz te, których działanie wzajemnie się wyklucza, czyli **nie mogą** równocześnie funkcjonować.

Współbieżność obiektów łatwo jest odczytać z diagramu stanów. Dwa obiekty są współbieżne, jeżeli mogą niezależnie od siebie odbierać zdarzenia i stosować się do nich. Niezależne systemy są o tyle wygodne, że można im przydzielić osobne jednostki sprzętowe, nie obciążając się dodatkowymi kosztami

wzajemnej komunikacji. Nie oznacza to, że niezależne systemy nie mogą pracować z tym samym procesorem. Dobrą platformą są dla nich systemy umożliwiające pracę wielozadaniową (ang. *multitasking*).

W praktyce niektóre obiekty, pomimo tego, iż konceptualnie są niezależne, w określonym stopniu zależą od siebie. Obiekty takie można umieścić na jednym wątku (ang. *thread of control*). Wątkiem nazywa się ścieżkę prowadzącą przez zbiór diagramów stanów, na której tylko jeden obiekt jest aktywny. Jeżeli dotychczas aktywny obiekt przekazuje kontrolę innemu obiektowi, pozostając jednocześnie w stanie oczekiwania na inne zdarzenie, to wątek zostaje przekazany temu drugiemu obiektowi i pozostaje w jego diagramie tak długo, aż kontrola nie zostanie z powrotem przekazana obiektowi poprzedniemu. Jeżeli obiekt wysyła zdarzenie do drugiego obiektu, a sam nie przestaje działać, wątek zostaje rozdzielony. Wątki są implementowane w systemie jako procesy (ang. *task*).

Przydział procesora i procesu podsystemom

Każdy współbieżny podsystem powinien być przydzielony do jednostki sprzętowej lub programowej, umożliwiającej realizację współbieżności. Platformą sprzętową może być jeden z procesorów, dedykowany wyłącznie do obsługi podsystemu. Może nią również być wyspecjalizowany układ scalony, realizujący określone zadanie. Wybór zależy od wymagań stawianych systemowi. Jeżeli system ma do czynienia z dużą liczbą danych, które muszą być przetworzone w rozsądnym czasie, nie wystarczy symulacja współbieżności na jednym procesorze. W takich wypadkach należy uruchomić system na maszynie wieloprocesorowej. Pożądane efekty może dać także zastosowanie wyspecjalizowanych układów scalonych, świadczących określone usługi. Rozwiązanie takie ma tę wadę w stosunku do metod programistycznych, że nie pozwala na łatwą zmianę stosowanych algorytmów. W takich przypadkach, jak zwykle, warto pomyśleć o kompromisie.

Zarządzanie danymi

Zarządzanie danymi jest kluczowym problemem przy konstrukcji praktycznie każdego systemu. Systemy meteorologiczne przechowują dane o pogodzie z ostatnich kilku miesięcy, systemy militarne zawierają informacje, które nie powinny wyjść poza mury jednostki. Najprostszym rozwiązaniem jest umieszczanie danych w plikach. Jest to prosty i skuteczny sposób. Ma on jednak kilka znaczących wad. Po pierwsze, implementacja plików różni się praktycznie w każdym rodzaju sprzętu, co w dużym stopniu ogranicza przenośność systemu. Poza tym, dostęp do plików jest dokonywany na dość „niskim" poziomie, co powoduje, iż system powinien być dodatkowo wyposażony w wygodny mechanizm obsługi.

Innym rozwiązaniem problemu jest zastosowanie istniejących systemów baz danych. Systemy baz danych są wyposażone w wygodne metody dostępu do danych (szczególną zaletą jest ogólnie przyjęty, standardowy sposób manipulowania danymi w postaci języka SQL) oraz efektywne algorytmy uaktualniania i przeszukiwania swoich zasobów. Wadą ich stosowania jest nienajlepsza współpraca z proceduralnymi językami programowania oraz trudność w zapamiętywaniu złożonych struktur danych.

Zarządzanie wspólnymi zasobami

W systemie, w którym niezależne procesy muszą korzystać ze wspólnych zasobów, istnieje problem synchronizacji dostępu do nich. Zasoby systemu można podzielić na kategorie:
- fizyczne – procesor, drukarka, dyski, pamięć itd.,
- logiczne – identyfikatory obiektów (ponieważ każdy obiekt jest jednostką niezależną i rozróżnialną, musi w związku z tym mieć niepowtarzalny identyfikator), nazwy klas i plików.

Dostęp do każdego globalnego zasobu powinien być kontrolowany. Typowym rozwiązaniem tego problemu jest stosowanie monitorów (ang. *monitor, guardian object*). Monitory są pośrednikiem między procesem wysyłającym żądanie dostępu do zasobu, a samym zasobem. Monitor przechwytuje żądanie dostępu i sprawdza, czy dostęp jest możliwy. Jeżeli jest – wówczas zasób jest przydzielany procesowi. Za pomocą monitorów możemy zarządzać pamięcią lub czasem procesora (jeżeli współbieżność jest realizowana w jednostce z jednym procesorem). Czasem konieczne jest, by dostęp do zasobu odbywał się bez udziału monitora. Wówczas w zasobie umieszcza się zamek (ang. *lock*), przez który procesy komunikują się bezpośrednio z zasobem. Takie rozwiązanie jest niebezpieczne, gdyż niekontrolowany dostęp do zasobu może spowodować wiele zamieszania.

Innym problemem systemów ze współdzielonymi zasobami są blokady (ang. *deadlock*). Blokada występuje w momencie gdy dwa lub więcej, procesów stara się o dostęp do zasobu i żaden z nich nie ma szans otrzymania go. Przykładem może być dostęp do stacji taśm magnetycznych. Jeżeli dwa procesy starają się o dostęp do stacji i każdy z nich zajął już jedną z nich (a każdemu potrzebne są dwa), wówczas następuje blokada. Rozwiązaniem problemu jest zastosowanie algorytmów wywłaszczających (ang. *preemptive algorithm*).

Istnieją także metody rozwiązywania problemu dostępu do wspólnych zasobów logicznych. Zabezpieczenie przed nadawaniem przez różne procesy tych samych numerów identyfikacyjnych różnym obiektom uzyskuje się przydzielając każdemu procesowi przedział do jakiego mogą należeć identyfikatory tworzonych w nim obiektów.

Ustalenie metody implementacji sterowania w systemie

Można wyróżnić dwa rodzaje przepływu sterowania w systemie: przepływ zewnętrzny oraz przepływ wewnętrzny. Przepływ zewnętrzny odpowiada przepływowi widzialnych na zewnątrz aplikacji zdarzeń między obiektami, podczas gdy przepływ wewnętrzny odpowiada przepływowi sterowania wewnątrz procesu. Wyróżniamy trzy kategorie zewnętrznych przepływów sterowania:
- sterowanie proceduralne (ang. *procedure-driven*), w którym sterowanie znajduje się wewnątrz kodu programu,
- sterowanie zdarzeniami (ang. *event-driven*), w którym sterowanie jest w rękach dyspozytora (ang. *dispatcher*),
- sterowanie współbieżne, w którym sterowanie znajduje się jednocześnie w niezależnych obiektach.

Zarządzanie warunkami brzegowymi

Część pracy przy projektowaniu systemu należy poświęcić warunkom brzegowym systemu, na które składają się: inicjalizacja, zakończenie oraz awarie. W momencie inicjalizacji należy pamiętać o nadaniu wartości początkowych odpowiednim składowym systemu. Należą do nich, między innymi: stałe, zmienne globalne, zadania, monitory itp. Zakończenie pracy systemu powinno wiązać się przede wszystkim z usunięciem z pamięci wszystkich niepotrzebnych obiektów oraz ze zwolnieniem zajętych zasobów. Błędy powinny być tak obsługiwane, by drobna usterka nie powodowała zawieszenia całego systemu.

Etap projektowania systemu jest pierwszym z etapów rozpatrywania systemu w dziedzinie implementacji. Następnym krokiem jest projektowanie obiektów.

Projektowanie obiektów

Jest to ostatni etap poprzedzający implementację. Jego zadaniem jest takie zoptymalizowanie systemu, aby w jak największym stopniu spełniał stawiane mu wymagania. Optymalizacja może dotyczyć pamięci operacyjnej, szybkości działania, przyszłej rozbudowy czy pielęgnacji systemu. Na tym etapie należy wybrać algorytmy implementujące operacje, struktury danych reprezentujące powiązania i atrybuty. Projektowanie obiektów jest więc kolejnym etapem „ulepszania" systemu przez dodanie do niego nowych szczegółów. Aby ten proces ułatwić i nieco zautomatyzować, warto kierować się poniższymi wskazówkami:

- połącz trzy modele analizy w jeden,
- zaprojektuj algorytmy implementujące operacje,
- zoptymalizuj ścieżki dostępu do danych,
- zaprojektuj powiązania,
- wyznacz reprezentację obiektów,
- utwórz z klas logiczne moduły.

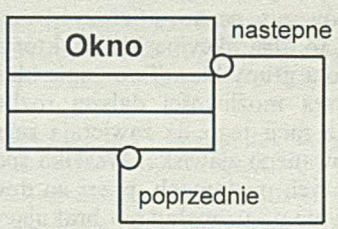
Wynikiem analizy systemu są trzy modele, odpowiadające trzem ortogonalnym sposobom widzenia systemu. Model obiektów jest statycznym „szkieletem" (klasy, atrybuty, powiązania), z którym kojarzy się odpowiednie zachowanie (operacje). Zadaniem projektanta jest przekształcenie akcji, zawartych w modelu dynamicznym, oraz procesów z modelu funkcjonalnego, na operacje na obiektach.

Każdy diagram stanu pokazuje cykl życia określonego obiektu (ang. object life cycle). Przejścia między stanami odpowiadają wykonaniu określonej operacji na obiekcie. Wynika stąd, iż każdemu zdarzeniu odpowiada konkretna akcja, dokonana na obiekcie. Zdarzenie wysyłane przez obiekt może reprezentować operacje wykonywane na innym obiekcie. Ponieważ zdarzenia pojawiają się zwykle w parach, odpowiadających wykonaniu operacji i zwróceniu wartości (a tym samym – powrocie sterowania do pierwszego obiektu), to oznacza, że należą one do jednego wątku, przekazywanego między obiektami, i mogą być reprezentowane jako jedno zadanie. Akcjom w diagramie stanów mogą odpowiadać całe diagramy przepływu danych w modelu funkcjonalnym. Zadaniem projektanta jest przekształcenie tego grafu w liniową sekwencję kroków algorytmu. W zależności od poziomu, na którym rozpatrywane są procesy, algorytm może być – na przykład – prostym przejściem przez strukturę modelu obiektów w celu odnalezienia wartości, albo skomplikowanym procesem, składającym się z wywołań wielu operacji. Od wielu innych kryteriów może zależeć wybór algorytmu. W przypadku problemów numerycznych lub inżynierskich, istotnym problemem jest złożoność obliczeniowa. Wybrane zostaną z pewnością te metody, których zapotrzebowanie na czas lub pamięć jest najmniejsze. Jeżeli tworzony system stanowić ma bazę dla innych aplikacji, pod uwagę należy wziąć łatwość dokonywania zmian oraz rozszerzeń. Algorytmy silnie związane ze strukturami danych, bądź korzystające ze specyficznej własności obiektu, mogą sprawić wiele kłopotu, nawet w przypadku małych zmian. Jeżeli jest to możliwe, warto pisać algorytmy „ogólniejsze". Podobna uwaga dotyczy typów argumentów obiektów. Argumenty „ściśle dopasowane" do funkcji obiektu w systemie mogą okazać się niewystarczające w przypadku modyfikacji lub rozbudowy systemu.

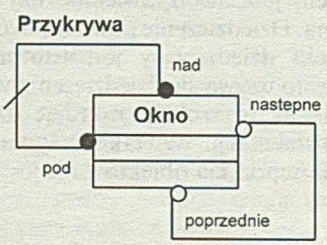
Na etapie projektowania obiektów zachodzi często potrzeba uzupełnienia listy operacji obiektów o te, których istnienie jest związane z implementacją. Z tej samej przyczyny do systemu mogą być dodane również całe obiekty. Przykładem mogą być omówione wcześnie monitory, synchronizujące dostęp do wspólnych zasobów, lub niskopoziomowe klasy, odpowiedzialne za graficzny sprzęg z użytkownikiem. Bardzo często w takich przypadkach pojawiają się problemy z umieszczeniem niektórych operacji w klasach (praktycznie nie występuje to w przypadku obiektów z dziedziny problemu). Szczególnie jest to widoczne wtedy, gdy z operacją jest związanych kilka obiektów. Ogólną regułą w takich przypadkach jest umieszczanie operacji w tych obiektach, na których jest ona wykonywana, a nie w tych, które operacje wywołują. Czasami spojrzenie na sieć powiązań ułatwia decyzję – jeżeli obiekty i powiązania między nimi tworzą gwiazdę wokół jednego z obiektów, wówczas jest on obiektem docelowym.

Jak wspomniano wcześniej, istnieje odwrotna zależność między systemem optymalnym (ogólnie pojętym) a rozszerzalnym. Najczęściej otrzymana w procesie analizy struktura połączeń obiektów jest mało optymalna. Poprawiając ją warto zwrócić uwagę na trzy aspekty:

- Czy nie należy dodać powiązań, które wprawdzie nie stanowią dodatkowej informacji, ale mogą przyspieszyć działanie lub ułatwić dostęp? Szczególnie przydatne w takich przypadkach są powiązania kwalifikowane, implementowane jako indeksy. Jeżeli stosunek elementów spełniających zadany warunek do liczności całej populacji jest mały, a wyszukiwanie jest wykonywane często, wówczas indeks znacznie je przyspieszy. Możliwe są także usprawnienia wewnątrz obiektu posiadającego listę cech. Kosztem większego wysiłku można ją zaimplementować jako tablicę haszowaną lub przynajmniej zastosować określony porządek;
- Czy można dodać atrybuty lub połączenia, które wprawdzie można wyprowadzić, ale w gotowej postaci mogłyby przyspieszyć działanie? Jeżeli obliczenia tych wartości są częste, wówczas umieszczenie ich explicite wewnątrz obiektu dałoby z pewnością pożądany efekt. „Sztandarowym" przykładem są nakładające się na ekranie okna. Pierwszym nasuwającym się rozwiązaniem jest diagram pokazany na rysunku 1. W momencie, gdy należy usunąć okno znajdujące się w środku „stosu", należy sprawdzić, które okna poniżej były przez nie przykryte, i je „odsłonić". Jest to proste, ale czasochłonne. Jeżeli operacja będzie przeprowadzana często, wówczas strata czasu będzie duża. Przyspieszenie można uzyskać dodając nowe połączenie, symbolizujące okna przykrywane przez obiekt. Pokazuje to rysunek 2.



Rys. 1. Okno „znające" jedynie swego następcę i poprzednika



Rys. 2. Wyprowadzone powiązania „Przykrywa"

Przekreślony symbol powiązania oznacza, iż jest ono wyprowadzone (ang. *derived*) z istniejących już powiązań. Wartość wyprowadzona musi być uaktualniana w momencie, gdy zmie-

nia się jeden z jej atrybutów bazowych (takich, od których jest ona zależna). Aktualizację można wykonywać na wiele sposobów. Najbardziej naturalne jest dokonywanie aktualizacji w momencie, gdy zmienia się dowolny atrybut bazowy. Jeżeli zmiany te będą częste, wtedy niestety proces będzie zajmował dużo czasu. Można więc wyznaczyć okres, po upływie którego atrybuty wyprowadzane będą aktualizowane. Jest to „hurtowe" podejście do rzeczy. Wadą jest to, iż w przypadku dużej liczby wyprowadzonych atrybutów aktualizowane są wszystkie, bez względu na to, ile bazowych atrybutów zmieniło swoją wartość.

Uważnego potraktowania przez projektanta wymaga sterowanie przebiegiem programu. Przebieg sterowania został już określony w modelu dynamicznym. Teraz zadanie polega na jego realizacji. Pierwszą metodą jest umieszczenie sterowania systemu w samym programie (ang. *procedureдriven system*). Jest to najbardziej rozpowszechniona do tej pory metoda. Stanem programu jest miejsce, w którym w danej chwili znajduje się sterowanie. Zdarzeniami są dane wejściowe. W zależności od otrzymanej danej, program rozgałęzia się (decydują o tym umieszczone w programie instrukcje warunkowe). Sposób przechodzenia od diagramu stanu do programu może wyglądać następująco:

- najpierw należy ustalić główną ścieżkę diagramu stanu, będzie ona odpowiadała „normalnemu" wykonaniu programu;
- przejściom w diagramie, odgałęziającym się od głównej ścieżki, należy przyporządkować odpowiednie wyrażenia warunkowe;
- cykle występujące w diagramie należy zaimplementować jako pętle; pozostałe ścieżki są po prostu wyjątkami, np., mogą to być błędy.

Ponieważ generalizacja zwiększa powtórną używalność klas, warto jej również poświęcić trochę uwagi. Pierwszych ulepszeń można dokonać wyszukując wspólne metody w hierarchii klas. Zdarza się czasem, że operacje zdefiniowane w różny sposób, mające różne argumenty i różne nazwy, w gruncie rzeczy świadczą te same, lub bardzo zbliżone usługi. W takim razie można je „doprowadzić do jednej postaci" i umieścić we wspólnym przodku.

Na tej samej zasadzie można zmienić strukturę obiektów, wyszukując klasy o podobnym zachowaniu. Jeżeli znajdą się takie klasy, wówczas warto utworzyć wspólnego przodka (najczęściej jest to abstrakcyjna klasa), który będzie zawierał cechy wspólne dla grupy klas. Tworzenie klas abstrakcyjnych znacznie zwiększa możliwości dalszej rozbudowy systemu. Dzieje się tak z racji tego, iż zawierają one jedynie ogólne aspekty modelowanego zjawiska. Wszelka specjalizacja dokonuje się na niższych poziomach, przez co dodawanie nowych klas polega jedynie na uzupełnianiu brakujących cech.

Częstym błędem jest nieodpowiednie używanie mechanizmów dziedziczenia. Dziedziczenie powinno być stosowane tylko wtedy, gdy obiekt dziedziczący jest istotnie rozszerzeniem poprzednika. Często używa się dziedziczenia w sytuacji, w której dokładnie można to zrobić, traktując dziedziczenie jako mechanizm implementacji. W takich sytuacjach warto się zastanowić, czy kompozycja obiektów nie jest lepszym rozwiązaniem.

* * *

Technologie obiektowe są nadal swego rodzaju nowością. Nie wytworzyły się jeszcze żadne standardy, zarówno w przypadku narzędzi, jak i metodologii. Najdalej posunęły się chyba obiektowo-zorientowane języki programowania. Do tej pory raczkują obiektowe bazy danych (tworzy się różne ich „mutacje" polegające – na przykład – na przeróbkach systemów relacyjnych). Obiektowe narzędzia CASE czy I-CASE też nie są jeszcze powszechne. Wszystko jednak wskazuje na to, iż najbliższe lata będą zdominowane przez technologię obiektową.

Przedstawiona w artykule technika tworzenia systemów obiektowych jest jedną z wielu, jakie powstały w ostatnim czasie. Jak wszystkie techniki obiektowe, ma ona tę zaletę, że na każdym etapie cyklu życia obiektu posługuje się tą samą terminologią i notacją. Jej autorzy twierdzą, iż nie jest to tylko nowa metodologia, ale również nowy sposób myślenia.

Celem artykułu nie było nauczenie stosowania techniki OMT, lecz tylko zasygnalizowanie problemu. Skrótowe potraktowanie, a nawet opuszczenie niektórych partii materiału wynika w głównej mierze z obszerności materiału. Zainteresowanych szczegółami tej techniki odsyłam do literatury.

LITERATURA

[1] Davis A.M.: A comparison of techniques fot the specification of external system behavior. Communications of the ACM, 31(9), pp. 1098–1115

[2] Harel D.: On visual formalism. Communication of the ACM, 31(5), pp. 514–529

[3] Martin J., Odell J.: Object-Oriented Analysis and Design. Prentice-Hall, Engelwood Cliffsa, N.J. 1992

[4] Odell M.: What is object state? Journal of Object-Oriented Programming, 5(2), pp. 19–22

[5] Rumbaugh J.: An object or not an object? Journal of Object-Oriented Programming, 5(3), pp. 20–25

[6] Rumbaugh J.: Object-Oriented Modelling and Design. Prentice-Hall Int., Engelwood Cliffs, N.J. 1991

[7] Winblad A.L.: Object-Oriented Software, Addison-Wesley 1990.

AKADEMICKA OFICYNA WYDAWNICZA RM

W grupie tematycznej INFORMATYKA serii „Problemy Współczesnej Nauki i Techniki" ukazały się dotąd następujące książki:

Andrzej Szałas: Zarys dedukcyjnych metod automatycznego wnioskowania. Akademicka Oficyna Wydawnicza, Warszawa 1992, wyd. 1, s. 121. ISBN 83-900451-7-6

W prezentowanej książce przedstawiono opis współczesnych dedukcyjnych metod automatycznego wnioskowania, bazujących na logice klasycznej i logikach modalnych. Na ogół pozycje poświęcone podobnej tematyce ograniczają rozważania do logiki klasycznej. Z drugiej strony, logikom modalnym poświęca się coraz więcej uwagi w różnych działach sztucznej inteligencji, a zwłaszcza w tych, które są związane z analizą i rozumieniem języka naturalnego. Dlatego też logikom modalnym poświęcono w książce wiele uwagi.

W przypadku logiki klasycznej zostały omówione najpowszechniej stosowane techniki, oparte na zasadzie rezolucji, metodzie tablic analitycznych, metodzie koneksji i wnioskowaniu naturalnym. W przypadku logik modalnych przedstawiono metody sprowadzania wnioskowania do dedukcji w logice klasycznej, przy czym główny nacisk został położony na te techniki, które można stosować w sposób automatyczny.

Książka jest przeznaczona zarówno dla Czytelnika nie wprowadzonego w problematykę automatycznego wnioskowania, jak i Czytelnika bliżej znającego tę tematykę. Jest adresowana do szerokiego grona praktyków zajmujących się oprogramowaniem, a także do teoretyków zainteresowanych czysto informatycznymi aspektami logik.

Ryszard Tadeusiewicz, Sieci neuronowe. Akademicka Oficyna Wydawnicza, Warszawa 1993, wyd. 1, s. 299. ISBN 83-85769-03-X

Książka poświęcona jest nowoczesnemu działowi informatyki, jakim jest teoria sieci neuronowych. Sieci te mają w stosunku do typowych systemów obliczeniowych dwie zasadnicze zalety. Po pierwsze obliczenia w sieciach neuronowych wykonywane są równolegle; tysiące sztucznych neuronów składających się na sieć, wykonują przypadające im zadania obliczeniowe równocześnie, w związku z czym szybkość pracy sieci neuronowych może docelowo miliony razy przewyższać szybkość aktualnie eksploatowanych komputerów. Po drugie, sieci nie trzeba programować. Opisane w książce metody uczenia i samouczenia sieci pozwalają uzyskać ich celowe i skuteczne działanie nawet w sytuacji, kiedy twórca sieci nie zna algorytmu, według którego można rozwiązać postawione zadanie. W książce omówiono wszystkie najważniejsze osiągnięcia w dziedzinie sieci neuronowych, uzyskane w okresie kilkudziesięciu lat prac nad tą dziedziną. Uwzględniono zarówno teoretyczne podstawy (zwłaszcza fascynującego procesu uczenia sieci), ale podano także konkretne dane (z cenami i adresami producentów) na temat dostępnego sprzętu i oprogramowania dotyczącego sieci, a także adresy (EMAIL) ośrodków zagranicznych, w których można pogłębiać zdobyte wiadomości.

Książka stanowi przystępne i łatwe wprowadzenie dla początkujących, ale zawiera także informacje bardzo zaawansowane, przeznaczone dla profesjonalistów. I wreszcie obok relacji z badań obcych (głównie amerykańskich) zawiera wiadomości o osiągnięciach autora i innych polskich badaczy, niektóre po raz pierwszy właśnie tu publikowane, jest więc w pełnym tego słowa monografią tematu.

Wojciech Mokrzycki, Encyklopedia przetwarzania obrazów. Akademicka Oficyna Wydawnicza, Warszawa 1992, wyd. 1, s. 86. ISBN 83-9451-4-1

Encyklopedia jest systematycznym zbiorem ok. 400 terminów, pojęć i ich wzajemnych związków z zakresu szeroko pojętego komputerowego przetwarzania obrazów (generowanie, przetwarzanie i analizowanie obrazów). Jest równocześnie propozycją polskiej terminologii w danej dziedzinie wiedzy.

W czasie, gdy informatyka w coraz większym stopniu dotyczy przetwarzania informacji obrazowej, Encyklopedia będzie pożyteczna nie tylko dla zawodowych informatyków, zajmujących się problematyką grafiki komputerowej, przetwarzaniem i analizą obrazów, ale również dla specjalistów z innych dziedzin nauki i techniki, zwłaszcza wykorzystujących komputery do wspomagania projektowania itp. Korzyści z Encyklopedii mogą mieć również studenci, zwłaszcza kierunków informatycznych i pokrewnych.

Przygotowanie Encyklopedii jest próbą wciągnięcia innych autorów do dalszych systematyzacyjnych prac w zakresie szeroko rozumianego komputerowego przetwarzania obrazów.

Leonard Bolc, Jacek Zaremba: Wprowadzenie do uczenia się maszyn. Akademicka Oficyna Wydawnicza, Warszawa 1992, wyd. 1, s. 161, ISBN 83-900451-5-X

Prezentowana książka jest poświęcona uczeniu się maszyn (ang. *machine learning*). Jest to dziedzina, która stała się w ostatnich latach kierunkiem wiodącym w sztucznej inteligencji i skupia na sobie uwagę wielu badaczy. Z metodami uczenia się maszyn związane są nadzieje na możliwość tworzenia oprogramowania samodostosowującego się do zmian w jego otoczeniu.

W książce przedstawiono podstawowe problemy związane z maszynowym przyswajaniem wiedzy. We wprowadzeniu scharakteryzowano pojęcie uczenia się, główne kierunki badań i typologię opracowanych metod. W kolejnych rozdziałach bardziej szczegółowo zostały omówione wybrane strategie uczenia się maszyn. Szczególnie dużo uwagi poświęcono metodom intuicyjnego przyswajania wiedzy.

Książka jest przeznaczona dla wszystkich zainteresowanych badaniami w dziedzinie sztucznej inteligencji.

Mamy nadzieję, że prezentowana książka będąca jedynie wprowadzeniem do tematyki uczenia się maszyn zaowocuje z czasem, również w naszym kraju, dalszymi publikacjami w tej tak burzliwie rozwijającej się obecnie na świecie dziedzinie wiedzy.

W przygotowaniu znajdują się następujące pozycje:

★ L. Bolc. S. Wierzchoń: *Propagacje niepewności w systemach inteligentnych. Metody numeryczne i logiczne*
★ P. Borowik: *Metody automatycznego wnioskowania w logikach wielowartościowych*
★ J. Cytowski: *Algorytmy genetyczne*
★ J. Doroszewski: *Struktura wiedzy i rozwiązywania problemów medycznych*
★ J. Kulikowski: *Komputerowe przetwarzanie obrazów biomedycznych*
★ M. Muraszkiewicz, H. Rybiński: *Reprezentacja wiedzy w bazach danych*
★ P. Rychlik, M. Wójcik: *Od logiki do reprezentacji wiedzy*
★ A. Skowron: *Podstawy sztucznej inteligencji*
★ J. Urbański: *Systemy utrzymywania wiarygodności*

Książki są do nabycia m.in. w:

- Oficyna Księgarsko-Wydawnicza ELEKTRONIKA, R. Wójcik i S-ka, ul. Mokotowska 51/53, 00-534 **Warszawa**, tel., faks: (02) 628-16-14
- Oficyna Księgarsko-Wydawnicza ELEKTRONIKA, R. Wójcik i S-ka, ul. Piotrkowska 39, 90-410 **Łódź**, tel., faks: (0-42) 32-51-64
- Oficyna Księgarsko-Wydawnicza ELEKTRONIKA, ul. Św. Mikołaja 51/53, 50-127 **Wrocław**, tel., faks: (0-71) 44-84-34
- Ośrodek Rozpowszechniania Wydawnictw Naukowych Polskiej Akademii Nauk ORPAN, Pałac Kultury i Nauki, 00-901 **Warszawa,** tel.20-02-11 wew. 27-43, 40-05
- Główna Księgarnia Naukowa im. B. Prusa, Krakowskie Przedmieście 7, 00-068 **Warszawa,** tel. 26-18-35, 26-64-49
- Główna Księgarnia Techniczna, ul. Świętokrzyska 14, 00-050 **Warszawa,** tel. 26-63-38
- Hurtownia Księgarni ELEKTRONIKA, ul. Mińska 25, **Warszawa,** tel., faks: (0-22) 13-38-45

Wskazówki dla Autorów

Nadsyłane artykuły nie mogą być publikowane lub przeznaczone do opublikowania w innych czasopismach.

Materiał oprócz tekstu zasadniczego powinien zawierać:

★ *krótki życiorys zawodowy Autora i jego zdjęcie,*
★ *wykaz literatury,*
★ *tabele,*
★ *materiał ilustracyjny (rysunki, zdjęcia czarno-białe, wydruki) dołączony do artykułu (nie wklejać materiału w tekst),*
★ *podpisy pod ilustracje.*

Na osobnej stronie prosimy podać tytuł naukowy, imię i nazwisko, nazwę zakładu pracy, adres domowy (koniecznie!), numer telefonu oraz informację, jaką drogą przekazać honorarium – kasa Wydawnictwa, poczta, bank.

Tekst artykułu prosimy dostarczyć w formie maszynopisu lub wydruku komputerowego (pisany jednostronnie, z podwójnym odstępem – bardzo ważne! – czyli 30 wierszy na stronie i 60 znaków w wierszu).

Wykaz literatury w porządku alfabetycznym.

Tabele – każda na osobnej stronie – powinny być ponumerowane, opatrzone tytułem oraz ściśle związane z tekstem (zaznaczone miejsce tabeli w tekście).

Rysunki – winny być czytelne (zaznaczyć ich miejsce w tekście), mogą być wykonane ołówkiem.

Zdjęcia – czarno-białe, kontrastowe, na błyszczącym papierze.

Wydruki – czytelne, kontrastowe, wykonane na białym papierze. Format – maksymalnie 18 cm w podstawie.

Każde pierwsze słowo opisu rysunków powinno być pisane dużą literą.

Podpisy pod ilustracje powinny być napisane na oddzielnej stronie, oprócz kolejnego numeru powinny zawierać tytuł (rysunku, zdjęcia, wydruku) i ew. legendę dotyczącą poszczególnych elementów.

Autor opublikowanego w INFORMATYCE artykułu otrzymuje honorarium i bezpłatny egzemplarz okazowy.

Materiałów nie zamówionych redakcja nie zwraca.

Imprezy zagraniczne

SYSTEMS'93

W dniach od 18 do 22 października br. odbędą się w Monachium Międzynarodowe Targi oraz Kongres SYSTEMS'93. Ta cykliczna międzynarodowa impreza informatyczna o 25-letniej już tradycji odbywa się ostatnio w cyklu dwuletnim, na przemian z Targami i Kongresem SYSTEC, ukierunkowanymi na problematykę komputerowego sterowania. Tegoroczna SYSTEMS będzie już trzynastą imprezą o tej nazwie, organizowaną przez Monachijskie Towarzystwo Targów i Wystaw na terenach wystawowych, bardzo dogodnie zlokalizowanych w centrum miasta w pobliżu Dworca Głównego (godziny otwarcia od 9. do 18.). W ostatnich latach Targi SYSTEMS wysunęły się na drugą pozycję w Europie zarówno pod względem powierzchni wystawowej (w 1991 r. 125 000 m^2), liczby wystawców (w 1991 r. prawie 1900 firm z 26 krajów), jak i liczby zwiedzających (w 1991 r. ponad 160 000 osób z 62 krajów), ustępując jedynie bezkonkurencyjnym hanowerskim Targom CeBIT.

Tegorocznym hasłem przewodnim imprezy jest „Technologia informacyjna dla Europy", odzwierciedlające zarówno aktualne trendy integracyjne naszego kontynentu, jak i ostrzegające przez nieustannie rosnącą w branży przewagę technologiczną i rynkową USA oraz Dalekiego Wschodu. Wyrazem tego rodzaju trendów jest przejęcie w tym roku organizacji towarzyszącego Targom Kongresu przez Radę Europejskich Towarzystw Informatycznych CEPIS (*Council of European Professional Informatics Societes*), zrzeszającą towarzystwa informatyczne 16 krajów europejskich (wśród nich Węgier, ale jeszcze bez Polski).

W materiałach informacyjnych podano następujące główne akcenty tegorocznej ekspozycji:

- mobilne przetwarzanie,
- multimedia na PC,
- nowe trendy rozwoju oprogramowania,
- pamięci optyczne,
- sieci komputerowe,
- technologia telekomunikacyjna.

Część kongresowa SYSTEMS'93 obejmie dwie następujące dwudniowe konferencje naukowe:

- w dniach 18 i 19. października – I. Europejski Kongres Informatyczny, organizowany przez wspomnianą organizację CEPIS, oraz
- w dniach 19. i 20. października – I. Europejski Kongres Telekomunikacyjny.

Oprócz powyższych imprez o profilu naukowym, w dniu 20. października odbędzie się II. Międzynarodowa Konferencja Sprzedawców Produktów Informatycznych, której tematem będą problemy wynikające z zaostrzającej się w branży walki konkurencyjnej oraz poszukiwanie nowych, bardziej skutecznych metod i form organizacyjnych stymulacji sprzedaży.

Ze względu na późniejszy o ponad pół roku termin SYSTEMS'93 w stosunku do tegorocznych Targów CeBIT oraz coraz szybszy postęp w technologii informatycznej, z dużą pewnością w zbliżającej się ekspozycji można spodziewać się prezentacji wielu rzeczywistych nowości.

Stosunkowa bliskość Monachium i coraz lepsze połączenia komunikacyjne tego miasta z naszym krajem, a także duża międzynarodowa ranga SYSTEMS powinny niewątpliwie skłonić wielu zainteresowanych w Polsce do odwiedzenia tej imprezy.

Dalsze szczegóły o SYSTEMS'93 można uzyskać w warszawskim przedstawicielstwie Międzynarodowych Targów i Wystaw w Monachium tel. (22) 17-75-83.

Oprogramowanie Komputerowych Systemów Czasu Rzeczywistego

Zagadnienia związane z wytwarzaniem oprogramowania do zastosowań przemysłowych

Ogólnopolska Konferencja Naukowo-Techniczna poświęcona tej tematyce organizowana przez Koło Stowarzyszenia Elektryków Polskich przy Fabryce Wagonów PAFAWAG, Sekcję Trakcji Elektrycznej Oddziału Wrocławskiego SEP oraz Oddział Dolnośląski Polskiego Towarzystwa Informatycznego, odbędzie się

we Wrocławiu w dniach 22–23.09.1994 r.

W początkowym okresie stosowania systemów komputerowych czasu rzeczywistego, oprogramowanie dla tych systemów było najczęściej wynikiem prac naukowo-badawczych.

Gwałtowny wzrost zastosowania systemów komputerowych, w tym także systemów sterowania i diagnostyki wbudowanych w urządzenia lub obiekty technicznie, radykalnie zmienił rolę oprogramowania. Oprogramowanie stało się produktem przemysłowym, opracowywanym zgodnie z istniejącymi normami, według zatwierdzonej technologii oraz podlegającym odpowiednim badaniom kwalifikacyjnym, podobnie jak wszystkie inne wyroby przemysłowe. Projektowanie oprogramowania do zastosowań przemysłowych przestało być twórczością naukową, a nabrało cech normalnej działalności inżynierskiej, jak w innych dziedzinach.

Konferencja poświęcona będzie takiemu właśnie, inżynierskiemu podejściu do wytwarzania oprogramowania dla systemów czasu rzeczywistego, stosowanych w takich dziedzinach, jak: kolejnictwo, elektroenergetyka, sterowanie ruchem lotniczym, przemysł lotniczy, automatyka przemysłowa, systemy przeznaczone dla wojska.

Główny zakres tematyczny konferencji jest następujący:
- Niezawodność i bezpieczeństwo systemów komputerowych.
- Specyfikacja wymagań dla systemu komputerowego i dla oprogramowania (ang. *system requirements specification, software requirements specification*).
- Stosowane metody projektowania oprogramowania.
- Języki, systemy operacyjne, narzędzia wspomagające.
- Organizacja i kierowanie procesem projektowania oprogramowania, zapewnienie wymaganej jakości oprogramowania, dokumentacja projektu, tryb sporządzania i zatwierdzania dokumentów.
- Testowanie, atestacja i certyfikowanie oprogramowania oraz współpraca Polski w tym zakresie z EWG.
- Tworzenie oprogramowania z istniejących standardowych części składowych (ang. *software reuse technology*).
- Pielęgnacja oprogramowania w eksploatacji, wprowadzanie zmian i atestacja oprogramowania po wprowadzeniu zmian.
- Standaryzacja w zakresie zagadnień objętych tematyką konferencji.
- Informacje na temat oprogramowania, dla objętych tematyką konferencji systemów, projektowanego aktualnie lub zaprojektowanego w ostatnich latach w kraju (zawierające w miarę możliwości informacje dotyczące ww. zagadnień).

Planowane jest zaproszenie przedstawicieli odpowiedniego komitetu technicznego UIC (Międzynarodowego Związku Kolei), którzy zaprezentują następujące normy międzynarodowe związane z tematyką konferencji:
- UIC 738R (1990): *Processing and transmission of safety informations* – przetwarzanie i transmisja informacji mających wpływ na bezpieczeństwo.
- IEC 9 (Secretariat) 305 I (1992): *Train Communication Network* – pociągowa sieć informatyczna – projekt normy IEC opracowany wspólnie z UIC.

Stworzy to możliwość bliższego poznania aktualnej praktyki inżynierskiej w dziedzinie projektowania oprogramowania dla komputerowych systemów czasu rzeczywistego stosowanych w kolejnictwie, które zaliczane są do najbardziej krytycznych ze względu na niebezpieczeństwo.

Zgłoszenia referatów oraz udziału w konferencji można nadsyłać do dnia **5.12.1993 r.** pod adresem:
Biuro konstrukcyjne F.W. PAFAWAG
ul. Fabryczna 12
53-609 Wrocław
z dopiskiem na kopercie „KONFERENCJA"

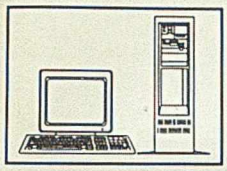
Bliższych informacji udziela mgr inż. Zdzisław Żurakowski, adres jw., tel. (071) 56-20-88, telefaks: 56-28-48.

INFORMACJI O PRENUMERACIE udziela Zakład Kolportażu Wydawnictwa Sigma NOT (ul. Bartycka 20, 00-716 Warszawa), skr. poczt. 1004, 00-950 Warszawa, tel. 40-00-21 w. 248, 249, 293, 297, 299 lub 40-30-86 i 40-35-89.

EGZEMPLARZE ARCHIWALNE CZASOPISM można nabyć za gotówkę w Klubie Prasy Technicznej w Warszawie, ul. Mazowiecka 12 (tel. 26-80-16) lub zamówić pisemnie. Zamówienia na egzemplarze archiwalne czasopism przyjmuje Zakład Kolportażu, Dział Handlowy, 00-950 Warszawa, skr. poczt. 1004 (tel. 40-37-31) na rachunek dla instytucji lub za zaliczeniem pocztowym dla osób fizycznych.

Adamski W.: Sieci komputerowe w polskim przemyśle lotniczym INFORMATYKA 1993, nr 10, s. 1 Charakterystyka rozwoju w Zakładach PZL Mielec systemu informatycznego opartego na lokalnej sieci komputerowej, a także obszarów i efektów wykorzystania tego systemu.	**Adamski W.: Computer networks in polish aircraft industry** INFORMATYKA 1993, No. 10, p. 1 Characteristics of development in the PZL Mielec Works of information system based on local area network, as well as of areas and effects of the system utilisation.	**Adamski W.: Computernetze in der polnischen Flugzeugindustrie** INFORMATYKA 1993, Nr. 10, S. 1 Eine Charakteristik von Entwicklung in den PZL Mielec-Werken des auf Lokalnetzwerk basierten EDV-Systems, sowie von Bereichen und Effekten der Ausnutzung dieses Systems.
Nowacki Z.A.: Szablony klas i listy elastyczne w pakiecie narzędziowym POLY do języka C INFORMATYKA 1993, nr 10, s. 6 Charakterystyka pakietu narzędziowego POLY do języka C oraz omówienie zalet i zastosowań najnowszych rozszerzeń C, jakimi są szablony klas oraz listy elastyczne.	**Nowacki Z.A.: Class patterns and flexible lists in the POLY tool package for the C language** INFORMATYKA 1993, No. 10, p. 6 Characteristics of the POLY tool package for the C language and discussion of advantages and application of the newest C extensions, which are class patterns and flexible lists.	**Nowacki Z.A.: Klassenmuster und elastische Listen im POLY-Hilfsmittelpaket zur C-Sprache** INFORMATYKA 1993, Nr. 10, S. 6 Eine Charakteristik vom POLY-Hilfsmittelpaket zur C-Sprache und eine Besprechung von Vorteilen und Anwendungen solcher neuesten C-Erweiterungen, wie Klassenmuster und elastische Listen.
Porwik P.: Multimedialne bazy wiedzy – rekomendacje INFORMATYKA 1993, nr 10, s. 13 Charakterystyka rozwiązań oraz wymagań programowych i sprzętowych związanych z przygotowaniem i umieszczaniem w bazach wiedzy różnych form obrazów graficznych oraz dźwięku.	**Porwik P.: Multimedial science bases – recommendations** INFORMATYKA 1993, No. 10, p. 13 Characteristics of software and hardware solutions and requirements connected with preparation and insertion of different graphic images and sound for science bases.	**Porwik P.: Multimediale Wissenschaftsdatenbanken – die Empfehlungen** INFORMATYKA 1993, Nr. 10, S. 13 Eine Charakteristik von Hard- und Softwarelösungen und -anforderungen, die mit Vorbereitung und Unterbringung in den Wissenschaftsdatenbanken von verschiedenen Formen der Graphikbilder und des Klanges verbunden sind.
Wieczorek M.: Podstawowe zagadnienia programowania zorientowanego obiektowo – polimorfizm i dziedziczenie klas w języku C++ INFORMATYKA 1993, nr 10, s. 16 Charakterystyka rozszerzeń i mechanizmów języka C++, umożliwiających programowanie zorientowane obiektowo.	**Wieczorek M.: Basic problems of object-oriented programming – polymorphism and class succession in the C++ language** INFORMATYKA 1993, No. 10, p. 16 Characteristics of the C++ language extensions and mechanisms, which make possible for object-oriented programming.	**Wieczorek M.: Grundprobleme der objektorientierten Programmierung – Polymorphismus und Klassenvererbung in der C++-Sprache** INFORMATYKA 1993, Nr. 10, S. 16 Eine Charakteristik von Erweiterungen und Mechanismen der C++-Sprache, die eine objektorientierte Programmierung ermöglichen.
Kasprzyk A.: Obiektowo-zorientowane analiza i projektowanie (2) INFORMATYKA 1993, nr 10, s. 21 Druga część charakterystyki amerykańskiej, obiektowo-zorientowanej metodologii tworzenia systemów informatycznych OMT (Object Modelling Technique), obejmująca omówienie jej drugiego etapu – projektowania systemu.	**Kasprzyk A.: Object-oriented analysis and design (2)** INFORMATYKA 1993, No. 10, p. 21 Second part of characteristics of the OMT (*Object Modelling Technique*), an american, object-oriented methodology for data processing system building, which includes its second phase – the system design.	**Kasprzyk A.: Objektorientierte Analyse und Projektierung (2)** INFORMATYKA 1993, Nr. 10, S. 21 Zweiter Teil einer Charakteristik von der amerikanischen OMT (Object Modelling Technique)-Methodologie, einer der objektorientierten Methodologien für EDV-Systemebau, der eine Besprechung ihrer zweiten Etappe – der Systemprojektierung, umfasst.

JUNISOFTEX®



Przedsiębiorstwo Produkcyjno-Usługowe Innowacji Technicznych Biuro Techniczno-Handlowe 44-100 Gliwice ul. Konstytucj 11.
tel. 31-57-00, 31-51-31, 31-75-10, fax 31-75-10, centr. 31-90-81 do 88, telex 036233
Zespół wdrożeń i analiz systemów komputerowych wewn. 15 ,25, 282, Zespół inżynierii oprogramowania wewn. 14, 20, 245,
Zespół d/s serwisu systemów Novell i sprzętu komputerowego wewn. 16, 245, Dział Handlowy wewn. 18, 28, 201,
Szkoła Komputerowa wewn. 282, Sekretariat wewn. 250, Prezes wewn. 13, 272, Dyrektor wewn. 12, 272, Główny Księgowy wewn. 256,
** Konto: PKO I/O Gliwice 27661-225298-136 **

OFERUJEMY

Kompleksową komputeryzację przedsiębiorstw obejmująca dostawę i wdrożenie zintegrowanych systemów komputerowych takich jak system:

finansowo-kosztowy (automatyczne rozliczanie kosztów) + gospodarka materiałowa + środki trwałe + wyposażenie + obrót towarowy fakturowanie sprzedaży + kadry-płace + techniczne przygotowanie produkcji

Sprzęt komputerowy w tym:

Komputery Monitory Dyski twarde Drukarki (mozaikowe i laserowe) osprzęt sieciowy (karty sieciowe, kable zwykłe i światłowodowe)

Systemy i oprogramowanie licencjonowane

DOS Novell Btrieve SQL Base Windows

Szkolenia komputerowe

podstawy obsługi komputerów arkusze kalkulacyjne (Qpro, Lotus, EXCEL) edytory tekstów (WordPerfect, TAG, ChiWriter) bazy danych (dbase IV, Clipper, Paradox) systemy operacyjne (DOS, Novell)

SOFTWARE SUPPORT FOR DOS, NOVELL 2.2, 3.11, 4.0, SQL BASE firmy GUPTA

*

OPROGRAMOWANIE DLA PRZEDSIĘBIORSTW

oferujemy własnej produkcji zakładowy system informatyczny PHU-Perfect(N) obejmujący swoim zakresem:
system finansowo-kosztowy umożliwiający automatyczne rozliczanie kosztów, atomatyczną dekretację, analizę rozrachunków
system obrotu materiałami, wyrobami, towarami (kody paskowe) w tym: stany magazynowe, ewidencje zamówień, fakturowanie sprzedaży (VAT) z automatyczną dekretacją rozdzielników kosztów i sprzedaży do systemu finansowo-kosztowego,
system ewidencji majątku trwałego i wyposażenia
system kadrowo-płacowy (SQL) z automatyczną dekretacją rozdzielników płacowych do systemu finansowo-kosztowego
system technicznego przygotowania produkcji (uk. 1993 r)
(oprogramowanie w systemie SQL BASE w przygotowaniu)

*

OPROGRAMOWANIE DLA URZĘDÓW MIAST I GMIN

oferujemy własnej produkcji system finansowo-podatkowy UMG-Perfect(N) obejmujący swoim zakresem:
system finansowo-księgowy,
systemy podatkowe obejmujące: ustalanie wymiarów, naliczanie i kontrolę płatności należności podatkowych dla dowolnych podatników i/lub płatników podatków na ich kontach i kontach dochodów ze swobodną korektą dowolnych informacji w dowolnym okresie a w tym:
ewidencja zapłat z wykorzystaniem kodów paskowych, emisja wezwań do zapłaty, naliczanie odsetek.
system ewidencji majątku

*

Zapraszamy do współpracy w zakresie wdrożen naszych systemów partnerów, - firmy informatyczne z całego kraju !!!

0/7/92

POLSKIE LITERY + NOWE MOŻLIWOŚCI

AKREM (90-950 Łódź 1, box 308, tel. 36-48-74) oferuje pakiety programowe:

POLY do CLIPPERA 87/5

- indeksowanie wg polskiego alfabetu w dowolnym standardzie,
- wprowadzanie, kontrola i manipulacje na danych o polskojęzycznym obrazie (GET-y, funkcje UPPER, ISALPHA itd.),
- praktycznie wszystkie znane metody programowego definiowania i symulacji polskich liter:
  - symulacja na Herkulesie w trybie znakowym i graficznym,
  - nie znikające polskie litery na EGA, VGA, SVGA we wszystkich trybach (w tym tryby tekstowe 25/28/43/50 wierszy, tryby graficzne, 512 fontów w trybie tekstowym),
  - polskie klawiatury (Alt i/lub dowolnie wybrany "klawisz polskiej litery", możliwość zmiany sposobu wprowadzania "w locie", liczne dodatkowe udogodnienia, jak np. automatyczna zmiana wielkości już wprowadzonych liter, "inteligentny "CapsLock, itd.),
  - na drukarce trzy metody , w tym nowość: nie znikający "download" (automatycznie samodefiniujący się po każdym włączeniu drukarki, działa bez żadnych przeróbek na każdej drukarce emulującej standard Epson lub IBM) oraz druk w trybie graficznym i najprostsza metoda "nakładania",
- dynamiczne dostosowanie programu do dowolnego (nawet własnego) standardu polskich liter,
- komunikacja programów w CLIPPERZE z rezydentami pakietu,
- bezbłędna współpraca z dowolnymi wersjami DOS-u,
- a ponadto system tablica/dysk, tablice wielowymiarowe o zwiększonej pojemności, szybkie operacje na tablicach, system pakowania danych (40-50% oszczędności pamięci dyskowej bez straty szybkości i miejsca w pamięci operacyjnej), opcja szyfrowania danych (przydatna dla użytkowników pracujących w sieci), menu wielopoziomowe, dwuwymiarowe i logiczne (wybór wielu opcji jednocześnie), manipulowanie atrybutami i kolorami ekranu, wyświetlanie dużych tekstów w okienkach, uniwersalne podejście okienkowe, polski kalendarz i liczby słownie, możliwość indeksowania i wyszukiwania danych wg polskiej wymowy, opcja definiowania własnych szablonów formatu, "trójwymiarowe" rozszerzenie funkcji DBEDIT, możliwość pracy z ekranami o zwiększonej liczbie wierszy i kolumn, klawisze F11 i F12 itd.

W skład pakietu wchodzą: biblioteka funkcji, cztery programy rezydentne (nakładki na DOS), edytor własnych fontów dla Herkulesa, VGA i drukarki (Draft i NLQ), programy pomocnicze, pliki rozmaitych fontów, programy przykładowe oraz doskonała dokumentacja (podręcznik oczywiście po polsku), a w przypadku CLIPPERA 5.x dodatkowo plik QUICKPOL umożliwiający polonizację bez zmian w programie źródłowym oraz plik NEWORDER definiujący strukturalno-obiektowy język NewOrder (włączający do CLIPPERA m.in. obiekty klasy MENU).

POLY-C

Zawiera wymienione wyżej programy rezydentne i narzędziowe, podręcznik napisany specjalnie dla C oraz bibliotekę blisko 300 funkcji dostarczanych w postaci źródłowej, przystosowanych do użycia wraz z dowolnym kompilatorem języka C na IBM PC (standard ANSI lub C++, szablony dla Borlanda 3.1), a także możliwych do wykorzystania przez znających C programistów w CLIPPERZE, FOX PRO czy też w PARADOX-ie.

Biblioteka funkcji POLY-C zawiera większość funkcji z biblioteki POLY dla CLIPPERA, a ponadto szereg funkcji napisanych specjalnie dla C. W bibliotece została uwzględniona kompleksowa (ekran, klawiatura, drukarka, manipulacje znakowe, konwersja) wzorowo zorganizowana polonizacja programu, a ponadto takie zagadnienia, jak: dowolne operacje na datach i łańcuchach znaków, pakowanie i szyfrowanie danych, pełnoekranowe wprowadzanie danych (obiekty GET), wyświetlanie dużych tekstów w okienkach, uniwersalne podejście okienkowe, funkcja INTPOLY symulująca operacje BIOS-u w pamięci ekranu, manipulacje ekranowe, drukowanie danych, organizacja menu wielopoziomowych, dwuwymiarowych i logicznych (obiekty MENU i PROMPT, menu strumieniowe), przetwarzanie list, help kontekstowy, język NewOrder, komunikacja programu w C z rezydentami pakietu, itd.

NEW INDEX

Nowy system obsługi baz danych w C. Zawiera obsługę obiektów FIELD, zbiorów DBF i DBT z możliwością pakowania pól bez straty szybkości pracy oraz zbiorów NDX, NTX i NZX (pliki indeksowe najnowszej generacji zaprojektowane specjalnie dla C, dowolnie wiele indeksów różnych rodzajów w jednym zbiorze, możliwość programowego ustawienia wielkości strony indeksowej). Udogodnienia do pracy w sieci. Generator Pól. Przezroczysta baza danych. Podręcznik.

POLY-DOS

Pakiet POLY-DOS umożliwia wszechstronną polonizację (tryb tekstowy i graficzny) oraz m.in. przedefiniowanie klawiatury (makrodefinicje dowolnej długości), definiowanie własnych fontów na ekranie i drukarce, komunikacja rezydentów z programami napisanymi w dowolnym języku, itd. Podręcznik.

EURO-SOFTWARE

Do POLY5 i POLY87 można dołączyć również moduły EXPORT umożliwiające przystosowanie programu do dowolnego alfabetu europejskiego (indeksowanie, wprowadzanie danych, duże i małe litery). Zaleca się zakup EXPORT-u łącznie z pełnym POLY-DOS-em, ponieważ można wtedy całkowicie przedefiniować klawiaturę, ekran i drukarkę.

BALL STAR - MECZ

Piłkarska baza danych na IBM PC zawierająca dowolne tabele i prognozy. Podręcznik.

* * *

A oto ceny pakietów (z podatkiem, ale bez opłaty pocztowej).

POLY 87 (do Clippera 87)	- 1.200 tys. zł
POLY 5 (do Clippera 5.0, 5.01 i 5.2)	- 1.400 tys. zł
POLY 5 + 87	- 1.800 tys. zł
EXPORT 5 lub 87	- 200 tys. zł
EXPORT 5 + 87	- 300 tys. zł
POLY-C (źródła)	- 2.200 tys. zł
NEW INDEX (źródła)	- 2.000 tys. zł
NEW INDEX + POLY C (źródła)	- 3.200 tys. zł
POLY-DOS	- 700 tys. zł
MECZ	- 550 tys. zł

Przy jednoczesnym zakupie POLY-C i do CLIPPERA dodatkowe 500 tys. zł zniżki. Zamówienia przyjmowane są listownie lub telefonicznie. Wysyłka za pobraniem.

Oferujemy również CA-CLIPPER 5.2, CA-dBFast 2.0 (baza danych z kompilatorem pod Windows) i inne oprogramowanie firmy Computer Associates. W tym przypadku obowiązuje zamówienie pisemne wraz z kserokopią dowodu wpłaty na konto

AKREM, PBK SA V/O ŁÓDŹ, nr 374606-11729-136.

0/1/93