1 1988

informatyka

Dr Hans-Dieter Baumbach o technologii programowania

Języki obiektowe

Turbo Prolog

WYDAWNICTWO NOT SIGMA

Cena 200 zł

informatyka

1 1988

Nr 1
Miesięcznik Rok XXIII
Styczeń 1988

Organ Komitetu
Naukowo-Technicznego NOT
ds. Informatyki

Materiałów nie zamówionych redakcja nie zwraca

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

Zakł. Graf. „Tamka". Zam. 0918-1300/87. Obj. 4,0 ark. druk. Nakład 8650 egz. U-23.

ISSN 0542-9951. INDEKS 36124

Cena egzemplarza 200 zł
Prenumerata roczna 2400 zł

WYDAWNICTWO
NACZELNA ORGANIZACJA TECHNICZNA
CZASOPISM I KSIĄŻEK TECHNICZNYCH
SIGMA
00-950 Warszawa
skrytka pocztowa 1004
ul. Biała 4

W NUMERZE: Strona

W NAJBLIŻSZYCH NUMERACH:

- Katarzyna Łubińska prezentuje system dydaktyczny do nauki języka ADA/SM.
- Ryszard S. Michalski omawia aspekty procesu uczenia się oraz przedstawia swoje poglądy na temat związków pomiędzy paradygmatami, strategiami i orientacjami tego procesu.
- Michał Kirpluk i Piotr Sobolewski zajmują się implementacją dedukcyjnej bazy danych Holmes.
- Robert Jaworski i Krzysztof Płowiec opisują monitor ekranowy Mazovii 1016.
- Maciej Sysło dzieli się z Czytelnikami uwagami o powszechnym kształceniu informatycznym studentów wyższych uczelni.
- Jan Bielecki omawia skojarzenia parametrów z argumentami w języku C.
- Zbigniew Fryźlewicz opisuje mikrokomputerowy analizator—tekster protokołów, przyspieszających lokalizację błędów transmisji wynikłych z nieprawidłowej implementacji protokołów.

Nasze wspólne problemy

Wśród setek czasopism zachodnich prawdopodobnie nie ma takiego jak INFORMATYKA: w jednym numerze można znaleźć zarówno opis funkcji BIOS-a w IBM PC, jak i artykuł prof. Roberta Kowalskiego o programowaniu w języku logiki. Czym więc jest INFORMATYKA i dla kogo powinna być przeznaczona? Pytanie to dość często zadają sobie zarówno Czytelnicy, jak i zespół redakcyjny. Odpowiedź jest szczególnie trudna, jeśli zważymy, że z jednej strony nasze czasopismo jest nadal jedynym w kraju periodykiem profesjonalnym, a z drugiej strony — nastąpiły liczne głębokie podziały specjalizacyjne wśród informatyków, a w konsekwencji znaczne zróżnicowanie zainteresowań i wymagań określonych grup tego środowiska. Do niedawna musieliśmy zaspokajać nawet potrzeby fanów informatyki w poznawaniu ZX Spectrum. Obecnie, dzięki pojawieniu się periodyków mikrokomputerowych, możemy już pomijać nawet zagadnienia związane z budową i oprogramowaniem IBM PC.

W moim przekonaniu, INFORMATYKA powinna przekazywać — może nieco spłycony, ale przez to szerzej dostępny — przegląd postępów informatyki w każdej z jej dziedzin. Inaczej mówiąc, powinien to być przegląd nowości, przekazywanych niestety z kilkumiesięcznym opóźnieniem. Trzeba jednak pamiętać, że dla większości informatyków, zwłaszcza pracujących poza dużymi aglomeracjami, nasze czasopismo jest często jedynym dostępnym w lokalnej bibliotece z zakresu aktualnej literatury zawodowej. Różne wydawnictwa, w rodzaju biuletynów i raportów instytutowych, ukazują się rzadko i z jeszcze większym poślizgiem czasowym. A jaki jest obecnie dostęp do literatury obcojęzycznej — wystarczy choćby spojrzeć na pustawe półki Centralnej Biblioteki Technicznej w Politechnice Warszawskiej.

Dużą grupę czytelników INFORMATYKI stanowią również studenci, dla których ze względu na bardzo powolną aktualizację podręczników, jest to w wielu przedmiotach właściwie jedyna literatura pomocnicza w języku polskim. Można więc stwierdzić, że przynajmniej dla tych dwóch, liczebnie dużych grup środowiska INFORMATYKA jest nie tylko potrzebna, ale nawet niezbędna.

Trzeba też stwierdzić, że istnieje w kraju pewna grupa informatyków, którzy chlubią się tym, że INFORMATYKI nie wzięli do rąk od kilku czy kilkunastu lat. Dla nich podstawowym źródłem wiadomości są prywatnie zdobywane czasopisma zachodnie, wyżebrane sprawozdania z konferencji, czy przetrzebione brakiem środków dewizowych zasoby bibliotek. Żeby nie było wątpliwości — ja ich za to nie potępiam, bo sam korzystam z takich źródeł. Mam tylko jeden zarzut, że zdobytą z takim trudem wiedzą nie dzielą się z innymi w kraju, dążąc za wszelką cenę tylko do publikacji w czasopismach zagranicznych. W karierze naukowej bowiem bardziej liczą się u nas takie publikacje, i to nawet wtedy, gdy prezentują nie najwyższy poziom i są publikowane w czasopiśmie o małej randze. Twierdzi się, że opublikowanie porządnego przeglądowego artykułu w INFORMATYCE nie jest warte odnotowania w dorobku naukowym.

Wróćmy jednak do naszych problemów. Jakie są skutki takiego stanu? Mamy stałe kłopoty z namówieniem dobrych specjalistów do prezentowania swojej wiedzy i osiągnięć na naszych łamach. Ratują nas przedruki z materiałów PTI, tłumaczenia publikacji znanych autorów zagranicznych oraz niewielka grupa autorów stale z nami współpracujących. Jest to jednak ciągle za mało, aby systematycznie podnosić merytoryczny poziom czasopisma. Obieg publikowanych w INFORMATYCE informacji jest więc niejako wymuszany wyłącznie przez redakcję. Nikły jest bowiem odzew czytelników, brak jest polemik z materiałami o treści często nawet celowo przejaskrawionej, a wreszcie zupełny brak zainteresowania tym, co się dzieje w naszym kraju w tej dziedzinie — oczywiście z wyjątkiem boomu mikrokomputerowego — ale i ten już traci oddech. Stan taki nie stwarza, przynajmniej u mnie, podstaw do optymizmu.

Spróbuję jednak podsumować miniony rok oraz przedstawić najbliższe zamierzenia. Wydaje się, że plany zaprezentowane rok temu w „Naszym miejscu na mapie" (Informatyka nr 3, 1987) zostały w znacznym stopniu zrealizowane. Każdy wierny nam czytelnik mógł skompletować co najmniej kilka pożytecznych dla siebie artykułów. W połowie roku odszedł z naszej redakcji MIKROKLAN, który w nieco zmienionej formie zabrał nam część tematyki związanej z problematyką IBM PC. Nawet specjalnie tym się nie martwimy, gdyż i tak zamierzaliśmy zająć się bardziej specjalistycznymi tematami i zastosowaniami.

Osobiście sądzę, że należałoby zrezygnować z dalszego prowadzenia działu „Dydaktyka". Muszę przyznać, że jego reanimacja udała się tylko częściowo. Jeżeli bowiem nawet ostry materiał prof. Tadeusiewicza (Informatyka nr 2, 1987) nie wywołał żadnego odzewu, to znaczy, że problemy dydaktyki właściwie obchodzą tylko kilku...leńców. Oczywiście, pomimo formalnego zamknięcia działu, będziemy nadal publikować dobre materiały na ten tak ważny temat.

Zamierzamy kontynuować dotychczasową linię czasopisma zawodowego — bez ścisłego definiowania tego pojęcia — utrzymując i rozwijając istniejące stałe działy. Oprócz prezentacji kolejnych języków programowania, systemów operacyjnych, sieci lokalnych, baz danych i zagadnień sztucznej inteligencji, szerzej zajmiemy się problemami inżynierii oprogramowania — działu bardzo istotnego dla właściwego technicznego przygotowania produkcji oprogramowania. Więcej uwagi poświęcimy systemom ekspertowym oraz prezentacji profesjonalnych systemów komputerowych — stacji roboczych.

W nawiązaniu do tego, o czym wspomniałem wcześniej, proponuję otwarcie kolumny ze streszczeniami referatów i artykułów publikowanych przez polskich autorów w zagranicznych czasopismach i materiałach konferencyjnych. O aktualności, a zwłaszcza kompletności tych informacji zadecydują przede wszystkim ich autorzy. Myślę, że moja propozycja — przynajmniej częściowo — może zmienić dotychczasową sytuację, gdy o pracach sąsiadów zza ściany dowiadujemy się, najczęściej przypadkowo, wyłącznie ze źródeł zagranicznych.

Systematyczny wzrost nakładu INFORMATYKI, który przekroczył już 8650 egzemplarzy, oraz podniesienie o przeszło 90% jej ceny (do 200 zł), powinny doprowadzić do tego, że wreszcie przestaniemy być czasopismem deficytowym. Od Was Czytelnicy i potencjalni autorzy, w znacznym stopniu zależy jednak, czy jej ukazywanie się będzie miało sens.

Trzeba niestety stwierdzić, że w dalszym ciągu aktualny jest ostatni akapit z zeszłorocznego artykułu wstępnego. W drukarni ciągle brakuje lewej kreski ukośnej, a korektorzy pracowicie odwracają literę „a", udającą chwilowo handlowe at. W dalszym ciągu brakuje papieru i mocy produkcyjnych, co nie pozwala na powrót INFORMATYKI do jej pierwotnej objętości, a także powoduje znaczne opóźnienia w ukazywaniu się kolejnych numerów. Niestety, rozwiązanie tych problemów już od nas nie zależy.

Pozostaje mi jeszcze zapewnić Was, Czytelnicy, że zawsze będziemy wdzięczni nie tylko za życzliwe, ale i krytyczne uwagi, postulaty i opinie, tak aby nasza praca — w znacznym stopniu społeczna — miała sens i przynosiła korzyść naszemu informatycznemu środowisku.

WACŁAW ISZKOWSKI

ANTONI KRECZMAR
Instytut Informatyki
Uniwersytet Warszawski

POLSKIE TOWARZYSTWO INFORMATYCZNE

Pierwsza wersja artykułu została przedstawiona podczas Szkoły Jesiennej PTI „Współczesne kierunki rozwoju informatyki" Mrągowo, listopad 1986 r.

Języki obiektowe (I)

Historycznie pierwszym językiem obiektowym była Simula-67, jakkolwiek w tamtych odległych czasach nie zdawano sobie sprawy z tej właściwości języka. Dopiero pojawienie się języka Smalltalk, który oferowany wraz z bogatym oprogramowaniem wspomagającym i na wyspecjalizowanym sprzęcie podbija świat, uświadomiło społeczności informatycznej znaczenie „obiektowości". Ostatnie lata przyniosły nowe wyniki w tej dziedzinie. W Polsce powstał język Loglan, w którym wzorując się na Simuli-67 istotnie wykorzystano pojęcie obiektu. Ten sam kierunek rozwoju reprezentuje Paragon, którego autorem jest Mark Sherman — cc ciekawsze będący także jednoosobowym wykonawcą całego cyklu pracy, tj. od projektu do implementacji języka.

Co łączy te wszystkie języki programowania? Dlaczego mówi się, że są one obiektowe? Otóż ich wspólną cechą jest możliwość operowania obiektami. Bardzo dobrze — powie uważny Czytelnik — ale cóż to jest obiekt? Przecież ten termin nic nie mówi, tym bardziej, że informatyka nie wprowadziła jeszcze na stałe definicji tego pojęcia do swego bogatego słownika. Postaram się zatem rozpocząć od wyjaśnienia czym jest obiekt i jak można go używać w językach programowania.

OBIEKTY I KLASY

Obiekt jest egzemplarzem struktury utworzonej według pewnego wzorca. W informatyce przyjęto nazywać takie wzorce klasami. A zatem, klasa określa wzorzec, według którego można utworzyć dowolną (oczywiście skończoną) liczbę obiektów. Ich wspólną cechą jest to, że powstały według jednego wzorca, jednakże każdy taki obiekt jest niepowtarzalny, a więc inny niż pozostałe utworzone obiekty.

Związek między obiektami a klasami przypomina świat Platona. Klasa reprezentuje wszystkie cechy podobnych przedmiotów. Klasy istniejące w idealnym świecie są wzorcami, według których powstają obiekty (przedmioty). Przypomnijmy, co pisze sam Platon w dialogu Parmenides (PWN, Warszawa, 1961, 130, V-B, tłum. W. Witwickiego).

— A taką mi rzecz powiedz. Tobie się wydaje, jak mówisz, że istnieją postacie pewne, w których uczestniczą te tutaj rzeczy i stąd mają ich nazwy; na przykład te, które uczestniczą w podobieństwie, nazywają się podobne, w wielkości wielkie, a w piękności i sprawiedliwości są sprawiedliwe i piękne?

— Tak jest — mówi Sokrates.

— Nieprawdaż; albo w całej postaci, albo w jakiejś części uczestniczy to, co uczestniczy? Czy może istnieć jakieś inne uczestniczenie poza tym?

— No, jakże? — powiada.

— Więc czy wydaje ci się, że cała postać jest w każdym z wielu przedmiotów, zostając jedną, czy jak?

— No, cóż przeszkadza, Parmenidesie — powiedział Sokrates — co jej przeszkadza być w nich całej?

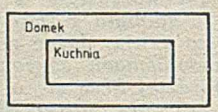
— Więc ona, będąc czymś Jednem i tym samym w licznych przedmiotach oddzielonych od niej, będzie w nich cała tkwiła i w ten sposób gotowa być oddzielona od siebie.

— No nie — powiada; — gdyby była taka, jak dzień, który będąc jednym i tym samym, w wielu miejscach jest równocześnie i zgoła nie jest dzięki temu oddzielony sam od siebie; może w ten sposób i każda postać może być jedną we wszystkich równocześnie i zostawać tą samą.

— Ty bardzo sympatycznie, Sokratesie — powiada — jedno i to samo równocześnie na wielu miejscach kładziesz; zupełnie jakbyś nad wieloma ludźmi jeden żagiel rozpinał i mówił, że oto jeden, a jest cały nad wieloma. Czy nie myślisz, że twierdzisz coś w tym rodzaju?

Ta rozmowa pomiędzy Parmenidesem i Sokratesem uzmysławia nam, jak trudno jest podać precyzyjną definicję obiektu i klasy — ponoć dialog „Parmenides" należy do najtrudniejszych dialogów Platona. Zamiast podawać zatem formalną definicję obiektu i klasy, posługując się pojęciami pochodzącymi z logiki i algebry, postaram się wprowadzić oba te pojęcia metodą przykładów i ich uogólnień. Jest to dobra klasyczna metoda, która ma tę zaletę, że nie wymaga podawania przykładów dla trudnych, formalnych definicji, których nie da się zrozumieć inaczej niż przez właściwie podane przykłady.

Zacznę od prostego przykładu, który nas podbuduje duchowo. Opiszę klasę obiektów umownie nazywanych „domek". Słowo „domek" może kojarzyć się z wieloma pojęciami, ale każdy domek ma pewne cechy wyróżniające go od innych przedmiotów, ma na przykład pewną liczbę izb, drzwi wejściowe, pewną liczbę okien, kuchnię, łazienkę itp. Jeżeli ktoś chce opisać formalnie klasę takich obiektów zwracając uwagę tylko na te cechy, które są mu potrzebne do opisu tej klasy, oraz te, których prawdopodobnie będzie w przyszłości używać, to wystarczy podać w jakiejś kolejności listę takich cech wraz z nazwami (nazwy są konieczne, albowiem nazwy te pozwalają odwoływać się do pojęć). Ale co na takiej liście może się znajdować? Otóż mogą być to znowu inne klasy. Na przykład, w każdym „domku" jest „kuchnia" (jest to znowu założenie umowne, wiemy że są domki bez kuchni, ale dla nich można przecież wprowadzić inną klasę). Zatem nasz „domek" będzie miał zawsze „kuchnię", a być może coś jeszcze, ale o tym powiem później Taką klasę można łatwo zilustrować (rys. 1).



Rys. 1.

W języku programowania ta definicja może przybrać postać następującą:

```
domek: class
  kuchnia: class
  end kuchnia;
end domek;
```

Między słowami kluczowymi **class** i **end** umieszcza się właśnie listę cech przynależących do danej klasy. Cechy te nazywa się zgodnie z terminologią informatyczną, atrybutami. Nazwa klasy poprzedza jej definicję, którą kończy słowo kluczowe **end**, po którym znowu może pojawić się nazwa klasy (tego rodzaju ortografia nie występuje we wszystkich wymienionych na wstępie językach obiekto-

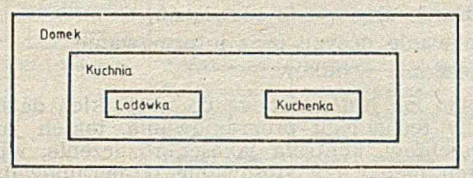
Dr hab. ANTONI KRECZMAR ukończył w 1967 r. studia na Wydziale Matematyczno-Fizycznym Uniwersytetu Warszawskiego. W 1973 r. obronił rozprawę doktorską, a w 1978 r. — habilitację. Pracuje na stanowisku docenta na Wydziale Matematyki, Informatyki i Mechaniki Uniwersytetu Warszawskiego. Jest jednym ze współtwórców języka programowania Logan, za opracowanie którego w 1986 r. zespół otrzymał nagrodę państwową pierwszego stopnia.

wych, jednakże znacznie zwiększa czytelność tekstu programu i zmniejsza liczbę nieporozumień). Zapisując kolejne wewnętrzne definicje, będę się starał stosować systematycznie wcięcia tekstu, co nie należy już do ortografii języka, ale również zwiększa znacznie czytelność programu.

W kuchni mogą znajdować się przedmioty, które warto wprowadzić jako artybuty tej klasy, np. lodówka i kuchenka. Odpowiednia definicja byłaby wówczas następująca:

```
domek: class
  kuchnia: class
    lodówka: class
    end lodówka;
    kuchenka: class
    end kuchenka;
  end kuchnia;
end domek;
```

Takiej klasie odpowiadałby rysunek 2.



Rys. 2.

Jak dotąd podałem przykłady klas, których atrybutami są inne klasy. Z drugiej strony taka najbardziej wewnętrzna klasa, jak na przykład „kuchenka", nie ma w tym przykładzie żadnych atrybutów. Oczywiście, dwie różne klasy bez atrybutów nie różnią się strukturą wewnętrzną, a jedynie samą nazwą. Z taką sytuacją mamy do czynienia bardzo rzadko. Najczęściej klasy, oprócz atrybutów, które także są klasami, mają pewne inne atrybuty, które nie posiadają już żadnej struktury wewnętrznej. Takie atrybuty bez struktury wewnętrznej są atrybutami ilości lub jakości. Na przykład „lodówka" może mieć jako atrybuty liczby określające wymiary, napięcie znamionowe, pobór mocy, pojemność, a także kolor, nazwę producenta, kierunek otwierania drzwi itp.

Wielkości liczbowe w informatyce są tzw. typami pierwotnymi „integer" i „real". Nie będę ich tu definiować. Typy jakościowe, jak na przykład — kolor, użytkownik może zdefiniować sam, tak jak w Pascalu, np.:

```
kolor = (biały, niebieski, zielony, siny, szkarłatny)
```

Dysponując dużym wachlarzem typów atrybutów, można podać nową definicję domku:

```
domek: class
  liczba_izb: integer;
  kubatura: real;
  izba: class
    powierzchnia: real
  end izba;
  kuchnia: class
    powierzchnia: real;
    lodówka: class
      szerokość, wysokość, głębokość: integer;
      napięcie, pojemność, pobór mocy: real;
      kolor_lodówki: kolor;
      kierunek_otwierania_drzwi: boolean;
      nazwa_producenta: text;
    end lodówka;
    kuchenka: class
    end kuchenka;
  end kuchnia;
end domek;
```

Można tę definicję rozbudowywać dalej, ale nie ma takiej potrzeby, gdyż takie rozbudowywanie łatwo jest wykonać bez konieczności przepisywania klas już zbudowanych (zwiększa to przyjemność dalszego wyposażania tego idealnego domku). Przypominam w tym miejscu, że mamy do czynienia z domkiem idealnym, takim bardziej platońskim. Jeżeli jednak komuś chodziłoby o domki konkretne, to takim domkom konkretnym odpowiadają w języku obiektowym właśnie obiekty.

Wyobraźmy sobie, że ktoś chce według wzorca klasy „domek" utworzyć kilka domków. W programie, w którym występuj definicja klasy „domek" można zadeklarować trzy różne nazwy, które będą odpowiadały trzem różnym domkom:

```
domek_Tomka, domek_Romka, domek_Atomka: domek;
```

Taka deklaracja mówi, że te trzy nazwy mogą wskazywać na obiekty klasy „domek" i tylko na takie. Nie znaczy to, że od razu muszą wskazywać na obiekty klasy domek, czasem mogą wskazywać, a czasem nie. Otóż w momencie deklarowania nie wskazują na nic. Dopiero wówczas, gdy programista podejmie decyzję, że dana nazwa ma wskazywać na dany obiekt, może taki obiekt utworzyć i związać go z tą nazwą. Do tego celu służy instrukcja generowania obiektu, która we wszystkich językach obiektowych ma podobną postać. Na przykład, poniższe trzy instrukcje generowania obiektu „domek":

```
domek_Tomka:=new domek;
domek_Romka:=new domek;
domek_Atomka:=new domek;
```

utworzą trzy egzemplarze domku, każdy związany z inną nazwą. Jest to bardzo ważny szczegół, który częstokroć nie doceniany przez programistów może prowadzić do wielu błędów. Otóż, gdyby spróbowano w jednej konstrukcji związać te trzy nazwy z domkiem, np.:

```
domek_Tomka, domek_Romka, domek_Atomka:=new domek;
```

to powstałby jeden egzemplarz domku związany z trzema różnymi nazwami. To nie musi być błąd, gdyż czasem programista właśnie tak chce postąpić, jednakże trzeba na tę istotną różnicę zwrócić baczną uwagę — dotyczy to w szczególności początkujących programistów.

Trzy obiekty klasy „domek", które wygenerowano, mają nieokreślone wartości atrybutów (mamy tu do czynienia dokładnie z tym samym zjawiskiem co w przypadku nazw obiektów jedynie zadeklarowanych, a nie wygenerowanych). Można teraz przystąpić do określenia atrybutów, np.:

```
domek_Tomka.liczba izb:=5;
domek_Tomka.kubatura:=3000.5;
domek_Romka.liczba izb:=3;
```

Atrybuty nieklasowe, tzn. typu nie będącego klasą, określa się za pomocą zwykłej instrukcji przypisania. Warto w tym miejscu wspomnieć, że dostęp do atrybutu obiektu uzyskuje się przez nazwę, po której następuje kropka. Nazwa wskazuje na obiekt, a kropka jest znakiem interpunkcyjnym oddzielającym tę nazwę od nazwy atrybutu. Takich kropek może być zresztą w jednym wyrażeniu wiele (gdy trzeba się dostać do bardzo zagnieżdżonego atrybutu).

Aby poprawnie wykonać dostęp do atrybutu klasowego, należy najpierw przygotować nazwę, która będzie wskazywać na taki nowo utworzony obiekt. W przeciwnym razie, utworzy się obiekt bez możliwości odwołania się do niego. Zatem w klasie „domek" można umieścić, na przykład, deklaracje nazw obiektowych:

```
domek:class
  liczba_izb:integer;
  kubatura:real;
  izby:array [ ] of izba;
  kuchnia_moja:kuchnia;
  kuchnia:class
  ...
  end kuchnia;
end domek;
```

a następnie generować odpowiednie obiekty w sposób zdalny:

```
domek_Tomka.kuchnia_moja:=domek_Tomka.new kuchnia;
domek_Tomka.izby[1]:=domek_Tomka.new izba;
domek_Tomka.izby[2]:=domek_Tomka.new izba;
```

Klasy mogą mieć parametry. Sposoby przekazywania parametrów w klasach są takie same jak w procedurach nie będę się zatem rozpisywać na ten temat. Dla uproszczenia założę, że będzie mowa tylko o parametrach przekazywanych przez wartość (parametry wejściowe).

dokończenie na s. 10

HANS-DIETER BAUMBACH
VEB Robotron-Projekt
Drezno
NRD

Stan i tendencje rozwoju technologii programowania w Kombinacie Robotron

W artykule przedstawiono najważniejsze zasady stosowane w wytwarzaniu oprogramowania w specjalistycznym przedsiębiorstwie Robotron-Projekt w Dreźnie. Treść artykułu oparto na głównych tezach referatu wygłoszonego na sympozjum zorganizowanym w maju 1986 roku przez ZETO we Wrocławiu.

TECHNOLOGIA PROGRAMOWANIA W ROBOTRONIE

Podstawą prac nad rozwojem technologii programowania w przedsiębiorstwie Robotron-Projekt w Dreźnie są zasady inżynierii oprogramowania. Prace te uwzględniają:
— zorganizowany i systematyczny przebieg wytwarzania,
— stosowanie właściwych zasad metod i sposobów postępowania,
— wykorzystywanie gotowych komponentów programowych,
— stosowanie odpowiednich narzędzi programowych.
Opierając się na tych zasadach, od wielu lat prowadzi się z powodzeniem prace podstawowe w dziedzinie technologii programowania przy ścisłym współdziałaniu z innymi krajami DWPG. Prace te są konsekwentnie kontynuowane, z uwzględnieniem następujących dwóch kierunków działania:

1) produkcji gotowych produktów programowych dla użytkowników w kraju i za granicą,

2) produkcji, stosowania i ciągłego doskonalenia narzędzi programowych dla potrzeb własnego przedsiębiorstwa przy tworzeniu wspomnianych gotowych produktów programowych.

Koncepcja faz

Podstawową zasadą inżynierskiego podejścia do wytwarzania oprogramowania jest podział procesu wytwarzania na fazy i etapy (model okresu istnienia oprogramowania). W tabeli pokazano standardowy model okresu istnienia oprogramowania. Jest on statystycznym odwzorowaniem dynamicznego procesu, w którym powtarzają się typowe czynności wytworzenia i zastosowania produktu programowego. W każdej fazie lub etapie realizowane są czynności, których wzajemne proporcje w poszczególnych fazach mogą zmieniać się Czynnościami tymi są:

- definiowanie wymagań,
- realizowanie rozwiązań,
- dokumentowanie,
- sprawdzanie oraz ocena rozwiązań,
- podejmowanie decyzji oraz informowanie,
- wykorzystanie wyników.

Koncepcja faz jest podstawą do wszystkich dalszych prac z dziedziny technologii programowania, takich jak: planowanie projektów, kontrola jakości, rozliczenie, wprowadzenie nowych metod czy stosowanie oprogramowania narzędziowego.

Koncepcja planowania projektów

W procesie wytwarzania oprogramowania (z uwzględnieniem podziału pracy) są wykonywane równolegle różne czynności. Ażeby złożoność tego procesu opanować, projekt należy rozłożyć na poszczególne rodzaje czynności, przestrzegając podstawowej zasady podziału projektu tak, aby pracochłonność poszczególnych zadań składowych nie przekraczała 10 do 25 osobodni. Na koncu każdego z tych zadań powstaje gotowy produkt lub część produktu, tzn. każde zadanie jednostkowe daje jako wynik gotowy produkt końcowy lub półfabrykat przeznaczony do dalszego opracowania. Poszczególne zadania jednostkowe muszą być realizowane w krótkich terminach i w sposób ciągły z zachowaniem wymaganej kolejności, aby w każdej chwili można było stwierdzić stan realizacji projektu. Plan projektowania jest podstawą efektywnego kierowania pracami.

Koncepcja stosowania metodologii

Stosowanie metod i zasad w dziedzinie wytwarzania oprogramowania oznacza przestrzeganie sprawdzonych sposobów postępowania, niezależnie od osobowości projektanta i jego podejścia do rozwiązania problemu, oraz konsekwentne i świadome ich uwzględnianie w pracy (np. programowanie strukturalne, projektowanie metodą zstępującą (ang. top-down), tablice decyzyjne, metoda hierarchiczna, standaryzacja). Wiele metod wydaje się banalnymi, a trudność ich użycia polega na ich wzajemnym oddziaływaniu i powiązaniu.

Koncepcja użycia narzędzi

Narzędzia wspomagające pracę wykonawcy oprogramowania są to środki programowe do realizacji metod i zasad postępowania zmierzających do automatyzacji faz, etapów i czynności. Wykorzystanie narzędzi powinno być oparte na właściwej metodologii, aby osiągnąć maksymalną skuteczność wspomagania.

W technologii programowania Robotronu koncepcje te są połączone i wspomagane przez odpowiednie gotowe produkty programowe.

PRODUKTY TECHNOLOGII PROGRAMOWANIA

Do najważniejszych produktów tej technologii należą: podręcznik Robotronu, normy państwowe i zakładowe oraz oprogramowanie narzędziowe.
Podręcznik technologii programowania Robotronu składa się z siedmiu części:

A. Ogólna charakterystyka technologii programowania;
B. Technologia ramowa;
C. Pojęcia;
D. Zasady, metody, sposoby postępowania;
E. Narzędzia;

Dr HANS-DIETER BAUMBACH studiował w latach 1961—1967 na Uniwersytecie Technicznym w Dreźnie, Wydział Gospodarka Przedsiębiorstwa. Od 1967 roku zatrudniony w Kombinacie ROBOTRON. Pracę doktorską na temat: „Problematyka tworzenia oprogramowania" obronił w 1975 roku. Prowadził zespoły robocze zajmujące się opracowaniem narzędzi programowych oraz technologii tworzenia oprogramowania. Współpracownik międzynarodowych zespołów roboczych zajmujących się technologią oprogramowania w RWPG.

F. Planowanie, kierowanie i organizowanie projektowania;
G. Specyficzne technologie użytkownika (przykłady).

Podstawowe normy są następujące:
- Norma NRD TGL 44546 (projekt). Model faz okresu istnienia oprogramowania,
- Normy Zakładowe Robotronu (KROS):
KROS 0324. Plan zapewnienia jakości (projekt);
KROS 0304/01-04. Dokumentacja systemu oprogramowania.

Okres istnienia oprogramowania

Proces	Faza	Etap	Wynik
Otrzymanie zlecenia	—	—	
Opracowanie analizy problemu	Analizowanie	—	Analiza w postaci opisowej Zdefiniowanie wymagań
Opracowanie założeń	Specyfikowanie	—	Specyfikacja wymagań
Wytwarzanie	Projektowanie	Projektowanie funkcjonalne (wstępne) systemu Projekt techniczny systemu Projektowanie programów Projektowanie modułów	Projekt funkcjonalny (wstępny) systemu Projekt techniczny systemu Projekt programów Projekt modułów
	Implementowanie (uruchamianie)	Implementowanie modułów Implementowanie programów Implementowanie systemu	Jednostki programowe
	Testowanie	Testowanie modułów Testowanie programów Testowanie systemu	Przetestowany moduł Przetestowany program Przetestowany system
	Dokumentowanie	—	Dokumentacja produktu programowego
Zastosowanie	Wdrożenie	—	Plan wdrożenia Uzasadnienie Świadectwo jakości Projekt pilotowy
	Eksploatowanie	—	Dane zakładowe Dokumentacja ośrodka obliczeniowego i działu informatyki Instrukcje organizacyjne Meldunki o zmianach
	Utrzymanie (pielęgnowanie)	—	Zmodyfikowany produkt programowy Zmodyfikowana dokumentacja Zmodyfikowane instrukcje organizacyjne
Wycofanie produktu	—	—	—

Poniżej omówiono najważniejsze narzędzia programowe:

1. TESYS — Technologiczny system wytwarzania oprogramowania

TESYS zawiera narzędzia do:
— sterowania programem (generowanie struktury programu za pomocą techniki szkieletowej i modułowej dla poszczególnych języków docelowych, programowanie strukturalne z językiem projektowania, technika tablic decyzyjnych, programowanie decyzyjne, programowanie normatywne, metoda Jacksona, inwersja programowa, wspomaganie testowania),
— definiowania danych i manipulowania danymi (zapis rekordu, definiowanie plików i manipulowanie plikami, opisywanie list z możliwościami dołączenia do DBS/R — systemu operacyjnego bazy danych Robotronu),
— dokumentowania (tworzenie aktualnej dokumentacji, zestawienia dokumentacyjne, kontrola kolejnych wersji, formatowanie tekstu, graficzne wyprowadzanie struktur gałęziowych),
— stosowania techniki makroinstrukcji,
— operowania bankiem metod i systemem bibliotecznym TESYS.

TESYS jest dostosowany do następującego sprzętu:
- Komputerów Jednolitego Systemu (np. EC 1055) i dołączania terminali (np. Robotron EC 7920) w odmianach:
TESYS-1 (Cobol) OS/JS,
TESYS-2 (Cobol, PL/I) OS/JS,
TESYS-2 (Cobol, PL/I), DOS-3/JS,
TESYS-2 (Cobol, PL/I) OS/JS z dodatkowym komponentem dla sprzężenia z DBS/R,
TESYS-3 (Cobol, PL/I, C) w projektowaniu dla PSU (Unix).
- Komputerów SKR (np. Robotron K 1630), jako TESYS-3 (Cobol, PL/I, C) w projektowaniu dla systemów operacyjnych MOOS 1600 oraz MVTOS 1600 (Unix).
- Mikrokomputerów 16-bitowych (np. Robotron A 7100), jako TESYS-3 (Cobol, C) w projektowaniu dla systemu operacyjnego SCP 1700 (CP/M-86).

Systemy TESYS-1 lub 2 były instalowane ponad 25 razy, m.in. dla takich odbiorców jak: Gosbank (ZSRR), Energoprojekt Sofia (Bułgaria), VEB Carl-Zeiss Jena (NRD) oraz we wszystkich ośrodkach przetwarzania danych (OVZ) w miastach powiatowych NRD. Wydajność programowania wzrasta przez zastosowanie systemu TESYS o 30 do 60 procent.

2. SEP 1600 — stanowisko wytwarzania oprogramowania komputerów SKR

SEP 1600 wspomaga wytwarzanie oprogramowania w językach Assembler, Cobol i Fortran oraz zawiera:
— narzędzia do projektowania (karty identyfikacji zadań, niezależne od języków wynikowych pseudokody, syntaktyczna i semantyczna kontrola projektowanego tekstu, pomoce do dokumentowania, wykresy strukturalne, diagramy cząstkowe),
— narzędzia do implementowania (automatyczne generowanie dostosowanego do kompilacji kodu języka docelowego z pseudokodu, wspomaganie kodowania we—wy oraz konwersji, tworzenie masek ekranu, wspomaganie przenoszenia i dokumentowania),
— narzędzia do testowania (symulacja środowiska testowanego za pomocą manipulatora danych testowych, tworzenie ram testów, tworzenie pseudomodułów, wspomaganie testowania modułów i programów metodą śladu i odtwarzania itp.),
— narzędzia do zarządzania (organizowanie, pielęgnowanie, wprowadzanie i wyprowadzanie ze zbiorów bibliotecznych).

System SEP 1600 został instalowany w systemie operacyjnym MOOS 1600 ponad 20 razy w kraju i za granicą. Uzyskany dzięki temu wzrost wydajności, zwłaszcza w fazie implementowania, wynosił od 30 do 50 procent.

3. LIST/M16 — procesor list dla komputerów 16-bitowych

LIST jest programem narzędziowym przeznaczonym do wspomagania programowania list, mającym następujące właściwości:
— niezależny od języków docelowych opis listy,
— sterowanie strukturą listy ponad instrukcjami języka,
— tworzenie pseudolist,
— funkcje tabelaryczne,
— specyfikowanie nagłówków tabulogramu i strony oraz stopek tabulogramu i strony.

Pracuje w systemach operacyjnych: SCP 1700 (CP/M-86) i MUTOS 1600 (Unix).

4. DIALOG/M16 — generator maskowania

DIALOG jest programem narzędziowym do wspomagania programowania dialogu użytkownika, mającym niezależne od języków wynikowych instrukcje do opisu maski. Zapewnia on:
— definiowanie rozmieszczenia maski (ang. layout) w formie obrazu,
— precyzowanie zmiennych i stałych przez opis formatu,
— sprawdzenie wartości zmiennych,
— technika okien, sprawdzanie prawidłowości syntaktycznej i semantycznej, protokołowanie,
— pseudoobraz dla kontroli,
— realizowanie kompletnych masek i poszczególnych okien.

Ponadto ułatwia on operowanie klawiszami funkcyjnymi, a także ustawianie atrybutów monitora ekranowego. Może pracować w systemie operacyjnym SCP 1700 (CP/M-86) i MUTOS 1600 (Unix).

dokończenie na s. 20

ANDRZEJ PAPROCKI
Instytut Maszyn Matematycznych
Warszawa

ADA/SM — kompilator podzbioru Ady dla komputerów SM-4

ADA/SM jest rezydentnym kompilatorem podzbioru języka Ada [1]) dla komputerów typu SM-4. Podzbiór ten, nazwany językiem ADA/SM, zawiera podstawowe konstrukcje Ady, pozwalając na pisanie zarówno sekwencyjne jak i współbieżnie wykonywanych programów. Kompilator jest przystosowany do współpracy z systemem operacyjnym czasu rzeczywistego DOS RW.

Kompilator ADA/SM opracowano w Instytucie Maszyn Matematycznych na zlecenie FMiK „Era". Jest on w tej chwili jedynym znanym autorowi rezydentnym kompilatorem podzbioru Ady dla komputerów SM-4.

CHARAKTERYSTYKA PODZBIORU

Dla opisu podzbioru zastosowano metodę przedstawiania przykładowych konstrukcji, wzorowaną na skróconym opisie Ady firmy Intermetrics [2] (p. tabela).

Podstawowym kryterium przy wyborze podzbioru była łatwość jego realizacji na kompilatorze o małej przestrzeni adresowej. Przy definiowaniu języka ADA/SM uwidoczniła się w pełni niezwykła spójność koncepcji Ady. Usunięcie poszczególnych konstrukcji powodowało często skomplikowanie pozostałych, które zdawały się być niepowiązane z usuwanymi. Ograniczenie się do podzbioru Ady spowodowało jednak konieczność wprowadzenia pewnych dodatkowych konstrukcji. Przykładem mogą być specjalne procedury obsługi wejścia--wyjścia wprowadzone ze względu na usunięcie z podzbioru jednostek rodzajowych.

Podstawowymi ograniczeniami języka ADA/SM w stosunku do normy Ady są: brak obsługi jednostek bibliotecznych i rozłącznej kompilacji (rozłącznie można kompilować jedynie procedury), brak typów niezawężonych i typów z wyróżnikiem, brak typów stałoprzecinkowych i zawężenia dokładności typów zmiennoprzecinkowych, brak przeciążania, jednostek rodzajowych i specyfikacji reprezentacji, brak instrukcji kodu i wejścia—wyjścia niskiego poziomu. Inaczej niż w Adzie zdefiniowano procedury wejścia—wyjścia, które w ADA/SM są wzorowane na Pascalu.

Pewne ograniczenia odnoszą się do możliwości korzystania z zadań. Zadanie jest jednostką kompilacji; wszystkie zadania, z których składa się program, są na jednym poziomie, tzn. zadanie nie może zawierać zadań wewnętrznych. Jednym sposobem komunikacji między zadaniami jest możliwość wywołania punktu wejścia i przekazanie wszystkich danych przez parametry. W ADA/SM zadania nie współdzielą ani procedur, ani danych.

Mgr ANDRZEJ PAPROCKI ukończył w 1978 roku studia na Wydziale Matematyki, Informatyki i Mechaniki Uniwersytetu Warszawskiego. Od tego roku pracuje w Instytucie Maszyn Matematycznych w Zakładzie Oprogramowania. Kierował zespołem projektującym i implementującym kompilator ADA/SM.

BUDOWA KOMPILATORA

Kompilator ADA/SM jest przystosowany do pracy w mapowanym systemie DOS RW. Obiektem wejściowym dla kompilatora jest plik źródłowy, który może być tworzony przez jeden ze standardowych programów redagujących, dostarczanych wraz z systemem DOS RW. Wynikiem pracy kompilatora jest wydruk programu z naniesionymi błędami i plik pośredni w formacie wymaganym przez Budowniczego Zadań systemu DOS RW. Proces kompilacji może być sterowany za pomocą kluczy. Razem z kompilatorem dostarczana jest biblioteka standardowa ADALIB, zawierająca procedury systemu wykonawczego i obsługę wejścia—wyjścia. Wykonywaniem programu wielozadaniowego steruje specjalne zadanie o nazwie ADAS, dostarczane również wraz z kompilatorem.

Kompilator jest pisany metodą tradycyjną. Ze względu na ograniczoną pamięć operacyjną zrezygnowano z najczęściej stosowanej dla kompilatorów Ady drzewiastej formy pośredniej typu Diana [6]. Kompilator ma strukturę wieloprzebiegową i składa się z: procedury sterującej; analizatorów: leksykalnego, składniowego, semantycznego; generatorów wydruku i kodu.

Procedura sterująca przetwarza polecenie wywołania kompilatora, ustawia wartości kluczy i identyfikuje pliki wywołania. Ponadto steruje wywołaniami poszczególnych przebiegów.

Analizator leksykalny, oprócz tradycyjnej funkcji dzielenia tekstu wejściowego na symbole, dodatkowo oblicza wartości literałów numerycznych, obsługuje pragmy związane z wydrukiem i alternatywnymi plikami wejściowymi oraz zamienia identyfikatory na jednoznacznie im przyporządkowane numery, co pozwala na skrócenie form pośrednich i utrzymanie założenia, że długość identyfikatora jest ograniczona do długości linii. Do identyfikacji słów zastrzeżonych zastosowana została jednoznaczna funkcja wyszukująca [7].

Analizator syntaktyczny jest typowy dla kompilatorów pisanych ręcznie — zastosowano metodę rekurencyjnych wywołań procedur. Daje się ona łatwo stosować dla gramatyk LL(1). Gramatyka prezentowana w normie Ady nie ma właściwości nie tylko LL(1), ale nawet LL(n), jednak daje się ona łatwo zamieniać na gramatykę LL(2), przy czym liczba miejsc, gdzie potrzebna jest znajomość dwóch symboli wejściowych, jest niewielka [5].

Analizator semantyczny stanowi największą część składową kompilatora. Ma budowę jednoprzebiegową w tym sensie, że cała forma pośrednia jest przeglądana tylko raz, jednak wiele informacji jest odkładanych na specjalnych listach do powtórnego wykorzystania. Tablica symboli jest tworzona i trzymana dla najdłuższej ścieżki programu. Dostęp do tablicy symboli jest realizowany przez funkcję wyszukującą z kluczem będącym numerem identyfikatora. Funkcja ta daje takie same wyniki jedynie dla zasobów o tym samym identyfikatorze. Analizator generuje kod dla maszyny stosowej. Adres maszyny stosowej składa się z nazwy jednostki kompilacyjnej, poziomu w ramach jednostki i przesunięcia w ramach poziomu.

Generator kodu wytwarza kod pośredni akceptowany przez Budowniczego Zadań. W kodzie wynikowym pamięć

Przykładowe konstrukcje ADY

Deklaracje obiektów i liczb:

```
A, B ,C : FLOAT;
I : INTEGER := 0; -- zmienna inicjowana
PRAWDA : constant BOOLEAN := TRUE;  -- stała
PI : constant := 3.14159;     -- liczba
```

Typy i podtypy:

```
type    MALY_INTEGER  is range -127..128; -- typ całkowity
subtype MALY_DODATNI  is MALY_INTEGER range
                              0..MALY_INTEGER'LAST;
subtype MALY_REAL     is FLOAT range -1.0..1.0;
type    DZIEN         is (Pn, Wt, Sr, Cz, Pt, So, Nd );
subtype DZIEN_ROBOCZY is DZIEN range Pn..Pt;
subtype LITERY        is range 'A'..'Z';
type    POCHODNY      is new BOOLEAN;
type    TABLICA       is array (CHARACTER,
                            DZIEN range So..Nd) of DZIEN;
type    LISTA;        -- niekompletna definicja
type    ELT_LISTY     is record
                           WARTOSC  : TABLICA;
                           NASTEPNY : LISTA;
                        end record;
type    LISTA         is access ELT_LISTY;
type    PRYWATNY      is private;
type    LIMITOWANY    is limited private;
```

Nazwy:

```
A         -- obiekt, literał, funkcja bezparametrowa
A(B)      -- element tablicy, wywołanie podprogramu
          -- wywołanie punktu wejścia
A(B)(C)   -- element tablicy dwuwymiarowej
          -- element tablicy będącej wynikiem wywołania
             funkcji
A.P       -- pole rekordu
          -- zasób P w podprogramie, pakiecie lub zadaniu A
A.all     -- obiekt wskazywany przez wartość wskaźnika A
A'(B)     -- obiekt B kwalifikowany typem A
```

Operatory: -- według malejącego priorytetu

```
**, abs, not,
*, /, mod, rem,
+, -,                 -- jednoargumentowe
+, -, &,
=, /=, <=, >=, <, >,
and, or, xor,

and then, or else     -- zwarte formy warunkowe
```

Instrukcje:

```
I := 1;           -- przypisania

if ZAJETA then  -- instrukcja warunkowa
   A := 1;
elsif WOLNE then
   P(A);
else
   raise WYJATEK;
end;

case DZIEN is     -- instrukcja wyboru
   when Pn..Sr => OTWARCIE := 9.30;
   when CzlPt  => OTWARCIE := 14.00;
   when others => OTWARCIE := -1.00;
end case;

for CH in TABLICA'RANGE(1)  -- pętla
loop
   T(CH,So) := Pn;
end loop;

PETLA: while I>0    -- Pętla nazywana
loop
   P(I);
   I := I - 1;
   exit PETLA when KONIEC;  -- Wyjście z pętli
end loop PETLA;

return -1;              -- powrót z funkcji
return;                 -- powrót z procedury
```

Podprogramy:

```
-- Deklaracje podprogramu:

procedure GET( CO   : out ELEMENT;
               STOS : in out T_STOS );
function KLUCZ( HASLA : in INTEGER) return T_KLUCZ;


-- Ciało podprogramu:

procedure GET( CO   : out ELEMENT;
               STOS : in out T_STOS) is
   K : INTEGER;     -- deklaracje lokalne
begin
   -- instrukcje procedury
end GET;

-- Wywołanie podprogramu

WEJSCIE := KLUCZ( 10 );
GET( ELT, STOS_1);
```

Pakiety:

```
-- Specyfikacja pakietu; definiuje zasoby dostępne

package URZADZENIE is
   type DOSTEP is ( ZNAKOWY, BLOKOWY );
   type ZASOB is private;
   SY: constant ZASOB;
   procedure ZAJMIJ( CO : in out ZASOB );
private
   type ZASOB is (DK_1, DK_2, DK_3, DK_4);
   SY: constant ZASOB := DK_1;
end URZADZENIE;


-- Ciało pakietu; zawiera ciała procedur ze specyfikacji

package body URZADZENIE is
   U : ZASOB;    -- wewnętrzna zmienna dostępna
                 -- tylko w ciele pakietu
   procedure ZAJMIJ( CO : in out ZASOB ) is
   begin
      -- instrukcje procedury
   end;
end URZADZENIE;
```

Zadania:

```
-- Specyfikacja zadania definiuje punkty wejścia
-- Ciało zadania zawiera odpowiednie instrukcje accept

task ZASOB is
   entry CZYTAJ( INDEKS: in CHARACTER; V : out INTEGER);
   entry ZAPISZ( INDEKS: in CHARACTER; E : in  INTEGER);
end;

task body ZASOB is
   type TABLICA is array( CHARACTER ) of INTEGER;
   T : TABLICA;
begin
   loop
      select
         accept CZYTAJ( INDEKS: in CHARACTER;
                        V : out INTEGER)
         do
            V := T( INDEKS );
         end CZYTAJ;
      or
         accept ZAPISZ( INDEKS: in CHARACTER;
                        E     : out INTEGER)
         do
            T( INDEKS ) := E;
         end ZAPISZ;
      end select;
   end loop;
end;
```

```
delay 50;  -- instrukcja opóźnienia zadania

select     -- instrukcja czekania selektywnego
   when WOLNY =>
      accept ...
      end;
or
   terminate;
end select;

select   -- warunkowe wywołanie wejścia
   ZASOB.CZYTAJ( 'A', J );
   return;
else
   null;
end select;


select   -- wywołanie wejścia uwarunkowane czasem
   ZASOB.CZYTAJ( 'A', J );
   return;
else
   delay 3;
end select;

abort ZASOB;  -- awaryjne usunięcie zadania
```

Wyjątki:

```
BLAD : exception;   -- deklaracja wyjątku

raise BLAD; -- instrukcja powodowania wyjątku

begin
   ....
exception           -- część obsługi wyjątków
   when BLAD =>
      WRITELN( TI, "Wystąpił błąd");
   when NUMERIC_ERROR =>
      raise;        -- spowodowanie tego samego wyjątku
   when others =>
      return(0);
end;
```

Przykładowe konstrukcje ADY

Wejście/wyjście:

```
Ze względu na brak w podzbiorze jednostek rodzajowych
wyjście/wyjście zostało zdefiniowane podobnie jak w Pascalu

-- typy dla systemu wejścia/wyjścia
type TEXT_FILE     is limited private;    -- pliki tekstowe
type FIX_REC_FILE is limited private;    -- ciągi wartości
type KIND_FILE_ACCESS is ( READ_ACCESS, WRITE_ACCESS,
                           APPEND_ACCESS);
type KIND_FILE_ATR is ( ORDINARY, SHARED, SPOOL, TEMPORARY);

Procedury otwarcia:
procedure OPEN_TEXT_FILE( F   : out TEXT_FILE;
                          FN  : STRING;  -- nazwa zewnętrzna
                          FACC : KIND_FILE_ACCESS;
                          FATR : KIND_FILE_ATR );
procedure OPEN_FIX_REC_FILE( F   : out TEXT_FILE;
                          FN  : STRING;  -- nazwa zewnętrzna
                          FACC : KIND_FILE_ACCESS;
                          FATR : KIND_FILE_ATR );
procedure OPEN_TERMINAL( F : out TEXT_FILE);

Procedury obsługi plików tekstowych
READ( F:TEXT_FILE, ....);  -- dowolna liczba parametrów typu
                           -- skalarnego lub tablic znaków.
READLN( F:TEXT_FILE, ...); -- tak jak READ po wczytaniu
                           -- przejście do nowej linii
WRITE( F:TEXT_FILE, ...);  -- dowolna liczba parametrów typu
                           -- skalarnego lub tablica znaków
WRITELN( F:TEXT_FILE,...); -- tak jak WRITE, po wypisaniu
                           -- danych nowa linia
SKIP_PAGE( F : TEXT_FILE)  -- opuszczenia znaków ze strony
PAGE( F : TEXT_FILE);      -- wypisanie znaku nowej strony
SKIP_LINE(F:TEXT_FILE; ILE:INTEGER); -- pominięcie ILE
                                     -- linii z wejścia
NEW_LINE(F:TEXT_FILE; ILE:INTEGER);  -- wypisanie ILE
                                     -- linii pustych
SCREEN(F:TEXT_FILE);         -- czyszczenie ekranu terminala

Funkcje:
EOLN( F : TEXT_FILE ) return BOOLEAN; -- badanie końca linii

Procedury obsługi plików stałorekordowych
pragma FILE_ELEMENT( plik, typ_powiązany ); --    powiązanie
          -- zmiennej typu FIX_REC_FILE z typem elementu
GET( F : FIX_REC_FILE; ELT: typ_powiązany );
PUT( F : FIX_REC_FILE; ELT: typ_powiązany );

Badanie końca pliku
function EOF( F : TEXT_FILE) return BOOLEAN;
function EOF( F : FIX_REC_FILE) return BOOLEAN;

Procedury zamknięcia
CLOSE( F : FIX_REC_FILE);
CLOSE( F : TEXT_FILE);
```

Atrybuty:

```
P'BASE        -- typ bazowy
P'CALLABLE    -- czy zadanie jest zakończone, przerwane lub
              -- jest wykonane abort
P'COUNT       -- liczba wywołań punktu wejścia w kolejce
P'FIRST       -- dolna granica typu
P'FIRST(N)    -- dolna granica N-tego indeksu tablicy
P'LAST        -- górna granica typu
P'LAST(N)     -- górna granica N-tego indeksu tablicy
P'LENGTH(N)   -- liczna wartości N-tego indeksu
P'POS(X)      -- liczba pozycyjna dla X
P'PRED(X)     -- poprzednik X
P'RANGE(N)    -- typ będący zakresem N-tego indeksu
P'SIZE        -- liczba bajtów dla reprezentacji typu P
P'SUCC(X)     -- następnik X
P'TERMINATED  -- czy zadanie zostało przerwane
P'VAL(X)      -- wartość typu P o numerze pozycji X
```

Pragmy:

```
FILE_ELEMENT( plik, typ)      -- powiązanie zmiennej plikowej
                              -- z typem elementu
FORM(arg)                     -- czy wydruk stronicowany
                              -- (arg = OFF lub ON)
INCLUDE( plik )               -- alternatywny plik źródłowy
INTERFACE( język, podprogram) -- podprogram w innym języku
LIST(arg)                     -- czy wydruk ma być generowany
                              -- (ON lub OFF)
PAGE                          -- przejścia do nowej strony
                              --  wydruku
PAPER_SIZE(wierszy, kolumn)   -- rozmiar strony wydruku
RUN_ERRORS(arg)               -- czy kontrola dynamiczna
                              -- (ON lub OFF)
FLOAT_FORMAT(F)               -- liczby rzeczywiste są wypisywane
                              -- na plik w formacie wykładniczym
FIXED_FORMAT(F, n, m)         -- liczby rzeczywiste są wypisywane
                              -- na plikF w formacie n cyfr
                              -- przed kropką i m cyfr po kropce
```

Standard:

Zasoby zdefiniowane na poziomie pakietu STANDARD:

```
type INTEGER is "universal_integer" range -32_768..32_767;
type FLOAT is "universal_real"
              range -1.7e38..1.7e38 digits 8;
type BOOLEAN is (FALSE, TRUE);
typ CHARACTER is -- znaki według standardu ASCII
CONSTRAINT_ERROR : exception;
NUMERIC_ERROER   : exception;
PROGRAM_ERROR    : exception;
STORAGE_ERROR    : exception;
TASKING_ERROR    : exception;
IO_NAME_ERROR    : exception;
IO_ACCESS_ERROR  : exception;
IO_DATA_ERROER   : exception;
IO_STATUS_ERROR  : exception;
IO_EOF_ERROR     : exception;
```

dla danych jest rezerwowana dynamicznie na stosie. Powiązania między poziomami wywołań procedur są realizowane na zasadzie tablicy DISPLAY. Pamięć przydzielana w programach źródłowych przez wykonanie alokatora jest rezerwowana w tym samym obszarze co pamięć dla zmiennych deklarowanych. Obsługa wyjątków jest realizowana metodą obsługi przerwań systemowych. Dla programów wielozadaniowych rezerwowany jest obszar wspólny na przekazywanie parametrów punktów wejścia, a każde zadanie programu jest tłumaczone na zadanie w systemie operacyjnym. Istnieje także zadanie systemowe ADAS, sterujące wykonywaniem zadań. W zadaniu ADAS znajdują się opisy stanu każdego zadania programu.

REALIZACJA I TESTOWANIE

Kompilator zrealizowano na komputerze SM-4 w językach Pascal i Macro-11. Projekt kompilatora był napisany w Adzie i nie wymagał istotnych zmian na etapie realizacji. Podstawowym problemem okazał się rozmiar kompilatora i konieczność zmieszczenia go w 32 K słów pamięci operacyjnej. Program kompilatora jest silnie nakładkowany. Średnia szybkość kompilacji wynosi 50—60 linii tekstu na minutę. Z powodu efektywności trzeba było zrezygnować z większości sprawdzeń zakresów udostępnianych przez Pascal, co spowodowało zwiększenie trudności w wyszukiwaniu błędów. Do celów uruchamiania stworzono specjalne programy wydruku i analizy form pośrednich między przebiegami. Przy uruchamianiu okazało się, że najczęściej stosowana w praktyce metoda wydruków kontrolnych nie zdaje egzaminu ze względu na dodatkową zajętość pamięci. W analizatorze semantycznym zastosowano zbiór procedur pozwalających na konwersacyjne wstrzymanie działania i oglądanie struktur w dowolnym węźle przetworzonego drzewa składni.

W trakcie uruchamiania powstał duży zbiór testów. Po oddaniu kompilatora do eksploatacji nadal pojawiały się błędy w jego działaniu. Okazało się, że poprawienie błędów bardzo często powoduje wprowadzenie nowych. Dlatego wydzielono zbiór około 120 specjalnie dobranych testów, które po każdej zmianie kompilatora są automatycznie kompilowane. Wyniki testów dla nowej wersji kompilatora są automatycznie porównywane ze zbiorami wzorcowymi dla tych testów i tworzony jest raport testowania. Pozwala to na szybkie stwierdzenie, czy w nowej wersji nie nastąpiło wprowadzenie nowych błędów.

* * *

W chwili obecnej oddana jest do eksploatacji wersja 1.1 kompilatora ADA/SM. Równolegle powstała wersja 1.0 kompilatora ADA2/SM. Oba kompilatory akceptują dokładnie ten sam język. Podstawową różnicą jest możliwość kompilowania przez ADA2/SM istotnie większych programów, gdyż tablica symboli kompilatora ADA2/SM została zrealizowana jako osobne zadanie systemu DOS RW, a dostęp do tej tablicy odbywa się na zasadzie komunikacji między zadaniami. Zwiększenie możliwości zostało opłacone wydłużeniem czasu kompilacji.

Kompilator ADA/SM uzyskał nagrodę III stopnia na Ogólnopolskich Targach Oprogramowania SOFTARG '86 oraz III Nagrodę Specjalną przyznawaną przez Ministra-Kierownika Urzędu Postępu Naukowo-Technicznego i Wdrożeń.

LITERATURA

[1] Ada Programming Language. ANSI/MIL-STD-1815A

[2] Ada Language Reference Card. Intermetrics, 1983

[3] ADA/SM — Opis języka. Dokumentacja firmowa. FMiK „Era"

[4] ADA/SM — Podręcznik użytkownika. Dokumentacja firmowa. FMiK „Era"

[5] Bonet R. et al.: Top-down syntax diagrams for Ada. Real-Time Data News, No. 2, 1981

[6] Goos G., Wulf W.: Diana Reference Manual. Intermetrics, March 1981

[7] Łubińska K., Paprocki A.: Jednoznaczna funkcja wyszukująca na przykładzie słów kluczowych języka Ada. Biuletyn Informacyjny Nauki i Techniki Komputerowe, nr 6, 1983.

[1]) Ada jest nazwą zastrzeżoną Departamentu Obrony USA.

JANUSZ POPKO
Instytut Maszyn Matematycznych
Warszawa

Jednostka centralna Mazovii 1016

Jednostka centralna jest podstawowym modułem funkcjonalnym 16-bitowego profesjonalnego mikrokomputera Mazovia 1016, do której są dołączone pozostałe moduły systemu (klawiatura, monitor ekranowy, drukarka). Mazovia 1016 wykonana w postaci wolnostojącego, zamkniętego modułu konstrukcyjnego o wymiarach 500×405×187 mm, zawiera:

- pakiet procesora z ośmioma złączami magistrali systemowej,
- czteronapięciowy zasilacz z układem wentylacji,
- trzy pakiety sterowników podstawowych urządzeń peryferyjnych (monitora ekranowego, drukarki, pamięci na dyskach elastycznych) umieszczone w złączach magistrali,
- dwie wbudowane pamięci na dyskach elastycznych 5,25 cala, o pojemności 360 KB (wysokość — 41 mm).

Przyjęte rozwiązania układowe i konstrukcyjne jednostki centralnej pozwalają na dalszą rozbudowę mikrokomputera przez dołączenie dodatkowych pakietów funkcjonalnych (np. sterowników urządzeń peryferyjnych, koprocesorów), jak również wbudowanie urządzeń peryferyjnych, np. pamięci Winchester.

W artykule omówiono pakiet procesora oraz krótko scharakteryzowano pozostałe bloki funkcjonalne. W zakończeniu przedstawiono ocenę kompatybilności jednostki centralnej Mazovia 1016 z wzorcem IBM PC.

PAKIET PROCESORA

Pakiet procesora JC-M86 jest wykonany w postaci obwodu drukowanego umieszczonego poziomo w obudowie jednostki centralnej. Na pakiecie rozmieszczono następujące bloki funkcjonalne:

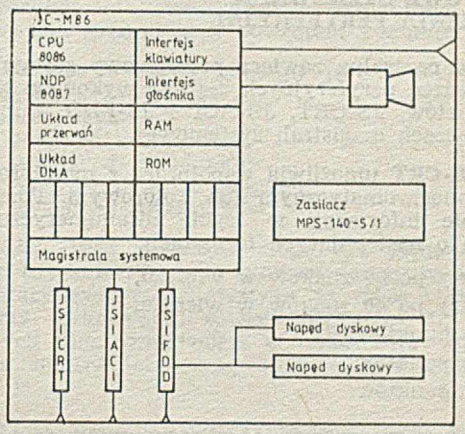
- blok procesora,
- układ DMA,
- pamięć ROM,
- pamięć RAM,
- lokalne układy we—wy,
- interfejs systemowy.

Procesor składa się z 16-bitowego mikroprocesora K1810 WM86 (analog 8086) oraz układów bezpośrednio z nim współpracujących. Mikroprocesor ma 16-bitową szynę danych i 20-bitową szynę adresową, co pozwala na zaadresowanie pamięci o pojemności 1 MB. Wykonuje operacje na słowach 8- i 16-bitowych, a jego lista rozkazów obejmuje rozkazy arytmetyczne z mnożeniem i dzieleniem, logiczne i sterujące oraz rozkazy działające na ciągach danych. Istnieje również możliwość dołączenia koprocesora numerycznego typu 8087.

Podstawowy cykl pracy mikroprocesora trwa cztery takty zegarowe po 210 ns każdy. Cykl ten jest automatycznie wydłużany do 5 taktów przy operacjach we—wy. Zegar mikroprocesora pracuje z częstotliwością 4,77 MHz. Lokalna szyna danych o szerokości 16 bitów zapewnia komunikację z pamięciami RAM, ROM i układami we—wy umieszczonymi na pakiecie procesora. Pamięć ROM ma pojemność 48 KB i zawiera podstawowe procedury obsługi urządzeń we—wy (tzw. BIOS), testy diagnostyczne sprawdzające poprawność pracy jednostki centralnej po każdorazowym włączeniu zasilania oraz interpreter języka Basic. Pamięć ROM zajmuje obszar od F4000 do FFFFF w przestrzeni adresowej mikroprocesora i jest zbudowana z sześciu układów EPROM typu 2764.

Lokalna pamięć RAM na pakiecie procesora może mieć pojemność 256 KB lub 640 KB (maksymalną, dopuszczalną przez system operacyjny). Zajmuje ona początkowy obszar adresowy mikroprocesora i jest wykonana z układów pamięciowych DRAM typu 4164 (64 Kb) lub 41256 (256 Kb). Odświeżanie zawartości pamięci odbywa się za pomocą odpowiednio zaprogramowanego kanału DMA.

Informacje do pamięci RAM mogą być przesyłane przez mikroprocesor jak i urządzenia peryferyjne pracujące w trybie bezpośredniego dostępu do pamięci. Umożliwia to układ typu 8237 zawierający cztery niezależne kanały DMA. Transmisja w dowolnym z kanałów DMA odbywa się na zasadzie zatrzymania pracy mikroprocesora. Jeden spośród tych kanałów jest wykorzystywany do odświeżania pamięci RAM.



Schemat blokowy jednostki centralnej Mazovia 1016

Na pakiecie procesora znajduje się też 8-poziomowy układ przerwań obsługujący przerwania zgłaszane przez magistralę systemową, jak również przerwania zegarowe generowane przez zegar programowy. Dodatkowe niemaskowalne przerwanie NMI mikroprocesora jest generowane w wypadku wystąpienia błędów w pracy jednostki centralnej, jak np. błąd parzystości w pamięci RAM.

Lokalne układy we—wy pozwalają na dołączenie klawiatury, sterują pracą głośnika i generatora przerwań zegarowych oraz umożliwiają ustawienie i odczyt różnorodnych wewnętrznych sygnałów sterujących.

MAGISTRALA SYSTEMOWA I ZASILANIE

Rozbudowa jednostki centralnej odbywa się przez instalowanie dodatkowych pakietów w złączach magistrali systemowej, umieszczonych na pakiecie procesora. Magistrala systemowa jest przedłużeniem szyny we—wy mikroprocesora 8086, przy czym szyna danych ma szerokość 8 bitów. Jej dopasowanie do 16-bitowej szyny mikroprocesora realizuje specjalny układ „konwertera szyny". Magistrala systemowa jest doprowadzona na ośmiu złączach pośrednich 64-kontaktowych (2×32) i obejmuje:

- 8-bitową dwukierunkową szynę danych,
- 20-bitową szynę adresową,
- cztery linie sterujące operacjami zapisu i odczytu do (z) pamięci i urządzeń we—wy,
- sześć linii przerwań,
- dwa sygnały zegarowe,
- siedem sygnałów sterujących przesłaniami w kanałach DMA,
- sygnał inicjujący operację odświeżania pamięci RAM,
- trzy pomocnicze sygnały sterujące,
- cztery napięcia zasilające +5 V, —5 V, +12 V, —12 V.

Magistrala systemowa działa synchronicznie z mikroprocesorem. W wypadku współpracy z wolniejszymi modułami funkcjonalnymi mogą one wydłużyć cykl pracy mikroprocesora za pośrednictwem sygnału gotowości. Magistrala pozwala praktycznie na dołączenie dowolnego modułu funkcjonalnego pod warunkiem, że będzie to moduł podporządkowany. Nie jest możliwe przekazanie sterowania innemu modułowi poza procesorem. Pakiety są umieszczane w złączach magistrali prostopadle do pakietu procesora. Tylna część pakietu jest wyposażona w tzw. szyldzik, będący częścią obudowy jednostki centralnej.

Jednostka centralna jest wyposażona w zasilacz impulsowy typu M PS-140-5/1 spełniający również funkcje wentylacyjne. Zasilacz wytwarza cztery napięcia zasilające:

+5 V/15 A
—5 V/0,5 A
+12/0,5 A (i 4,0 A)
—12/0,5 A

Napięcie +12 V jest wytwarzane w dwóch źródłach, z których jedno, o większej wydajności, służy wyłącznie do zasilania urządzeń peryferyjnych wbudowanych w jednostkę centralną. Jedno z pięciu złącz wyjściowych doprowadza zasilanie do pakietu procesora, a pozostałe służą do zasilania urządzeń peryferyjnych.

PODSTAWOWE STEROWNIKI I URZĄDZENIA PERYFERYJNE

Jednostka centralna zawiera podstawowy zestaw sterowników urządzeń peryferyjnych. Są one wykonane w postaci trzech pakietów, JS-CRT, JS-ACI i JS-FDD, zainstalowanych w złączach magistrali systemowej.

Pakiet JS-CRT umożliwia współpracę z monitorem ekranowym monochromatycznym lub kolorowym. Pozwala na wyświetlanie informacji w trybie alfanumerycznym lub graficznym o następujących formatach:

- 25 wierszy po 80 znaków w wierszu,
- 25 wierszy po 40 znaków w wierszu,
- 720×350 punktów,
- 640×200 punktów,
- 320×200 punktów.

Sterownik ma własną pamięć obrazu umieszczoną w przestrzeni adresowej mikroprocesora i może współpracować z monitorami o częstotliwości odświeżania ekranu 50 Hz i 60 Hz.

Sterownik JS-FDD pozwala na dołączenie czterech napędów pamięci na dyskach elastycznych 5,25 cala oraz drukarki mozaikowej lub innego urządzenia, pracującego w standardzie Centronics. Przesyłanie informacji do (z) pamięci dyskowych odbywa się w trybie DMA.

Sterownik JS-ACI umożliwia dołączenie do jednostki centralnej urządzeń wyposażonych w styk (V24), jak np. ploterów. Pozwala on na przesłanie informacji szeregowo, z szybkością od 50 do 9600 bodów.

Jednostka centralna jest standardowo wyposażona w dwie pamięci na dyskach elastycznych 5,25 cala, dwustronnych, o pojemności 360 KB każdy. Pamięci są wmontowane w specjalną „kieszeń" w obudowie jednostki centralnej i połączone kablem do sterownika JS-FDD. Zasilanie pamięci dyskowej jest doprowadzone z zasilacza MPS-140-5/1.

* * *

Przy projektowaniu jednostki centralnej Mazovia 1016 przyjęto założenie osiągnięcia pełnej zgodności funkcjonalnej ze wzorcem IBM PC oraz — maksymalnie bliskiej zgodności konstrukcyjnej.

Pełną zgodność funkcjonalną osiągnięto dzięki wiernej adaptacji całej architektury wzorca, tj. zarówno procesora jak i sterowników urządzeń we—wy. W tym celu zastosowano analogiczne układy LSI produkcji krajów socjalistycznych i zachodnich. Jedynym odstępstwem było zastąpienie mikroprocesora 8088, z 8-bitową szyną danych, mikroprocesorem K18110WM86 produkcji ZSRR, z 16-bitową szyną danych, zgodnego funkcjonalnie z układem 8088. Ponadto dokonano maksymalnie zgodnej adaptacji BIOS-a.

Przyjęte rozwiązania konstrukcyjne zachowały podstawowe cechy wzorca w takim stopniu, że umożliwiają instalowanie pakietów opracowanych w standardzie IBM PC w jednostce centralnej Mazovia 1016 (za pośrednictwem specjalnego przedłużacza). Ograniczenia techniczne w produkcji Mazovii 1016 spowodowały zwiększenie o ok. 40 mm wysokości pakietów sterowników oraz zastąpienie złącz krawędziowych magistrali systemowej złączami pośrednimi. Przeprowadzone badania wykazały blisko stuprocentową kompatybilność jednostki centralnej Mazovia 1016 ze wzorcem. Występujące niezgodności są związane z większą szybkością działania użytego mikroprocesora w stosunku do 8088.

Języki obiektowe

dokończenie ze s. 3

Jeżeli klasa ma parametr formalny wywoływany przez wartość, to odpowiedni parametr aktualny określa jego wartość w momencie generowania, tak jak w przypadku wywołania procedury. Przykładowo, dla klasy „domek" atrybuty „liczba izb" oraz „kubatura" można umieścić na liście parametrów:

```
domek:class (liczba_izb:integer, kubatura:real);
  izby:array [ ] of izba;
  ...
  end domek;
```

Wówczas generowanie obiektu takiej klasy pozwala jednocześnie określić wartości tych artybutów:

```
domek_Tomka:=new domek (5,300.5);
domek_Romka:=new domek (3, domek_Tomka.kubatura*2);
```

Klasy umożliwiają tworzenie obiektów i wykonywanie na tych obiektach pewnych czynności z zewnątrz za pomocą dostępu kropkowanego (zwanego dostępem zdalnym). Języki obiektowe pozwalają jednak na coś więcej. Otóż klasa może mieć także zdefiniowany pewien ciąg akcji, które są wykonywane w momencie generowania obiektu.

Wracając do przykładu modnego ostatnio budownictwa jednorodzinnego, klasę „domek" można zapisać tak, aby generowanie „kuchni" i „izb" wykonywało się samoczynnie dla każdego obiektu, bez konieczności wykonywania tego z zewnątrz. Taka deklaracja może mieć następującą postać:

```
domek:class (liczba_izb:integer, kubatura:real);

  izby:array [1:liczba_izb] of izba;
  izba:class
  end izba;
  kuchnia_moja:kuchnia;
  kuchnia:class
  end kuchnia;
begin
  for i:1 to liczba_izb do izby i :=new izba;
  kuchnia_moja:new kuchnia;
end domek;
```

Wykonanie instrukcji generowania obiektu klasy „domek", np.:

```
domek_Tomka:=new domek (5,3000.5);
```

spowoduje teraz utworzenie obiektu, przesłanie parametrów i wykonanie ciągu instrukcji wyznaczonych przez deklarację klasy. A zatem zostaną utworzone „izby" oraz „kuchnia moja". Takie same czynności zostaną wykonane przy generowaniu innych obiektów klasy „domek".

Atrybutami mogą być także procedury i funkcje. Przykłady takich atrybutów podano w dalszej części artykułu. Będą one w miarę naturalne, natomiast sztuczne rozbudowywanie klasy „domek" w celu zilustrowania tego pojęcia nie byłoby już tak przekonujące.

WŁADYSŁAW GĄSIOREK
Instytut Organizacji i Techniki
Transportu Kolejowego
Politechnika Krakowska

Pewne problemy konstrukcji i oprogramowania systemowego dla Mery 300

W latach 1978—1985 autor artykułu współpracował przy opracowaniu kilku systemów programowania na minikomputery Mera 300, zawierających łącznie ponad 10 tys. rozkazów maszynowych. Duża efektywność tych systemów stanowiła spore zaskoczenie dla specjalistów o wieloletnim doświadczeniu w opracowywaniu systemów informatycznych na ten rodzaj sprzętu komputerowego.

W artykule tym przedstawiono rozwiązania, które zadecydowały o takiej efektywności, a które z powodzeniem mogą być stosowane na innym sprzęcie komputerowym.

Są to:

- tworzenie programów wynikowych w kodzie pośrednim,
- równoległa realizacja wszystkich funkcji zarządcy systemu operacyjnego,
- cykliczna organizacja buforów urządzeń wejściowych.

CHARAKTERYSTYKA SYSTEMÓW PROGRAMOWANIA

W zależności od rodzaju zastosowania i konfiguracji sprzętowej, opracowane systemy umożliwiają dogodne wprowadzanie, korektę, uruchamianie, tworzenie kopii i dokumentacji zarówno programów systemowych, jak i użytkowych. Zawierają one m.in. translatory języków programowania, wzorowanych na makroasemblerze Mery 400.

Najbardziej rozbudowany jest SOMEK2. Umożliwia on m.in. równoległe realizowanie następujących funkcji:

- pisanie komentarzy operatorskich na klawiaturze DZM,
- przyjmowanie i realizowanie komend operatorskich,
- wykonywanie programów użytkowych w trybie pracy krokowej lub ciągłej,
- sterowanie wymianą danych między jednostką centralną a urządzeniami zewnętrznymi (pamięć dyskowa, taśmowa i kasetowa, drukarka znakowo-mozaikowa, czytnik i perforator taśmy papierowej),
- przyjmowanie meldunków od ośmiu abonentów połączonych z systemem za pośrednictwem central komutowanej sieci dalekopisowej (meldunek może zawierać ok. 6000 znaków),
- wysyłanie zwrotnych komunikatów redagowanych przez programy użytkowe po przetworzeniu meldunku (komunikat może zawierać ok. 400 znaków),
- wysyłanie danych przez kanały znakowe do komputera Odra 1300,
- przyjmowanie od komputera Odra lub od programów użytkowych systemu SOMEK telegramów adresowanych do dowolnego abonenta komutowanej sieci dalekopisowej; system może przechowywać równocześnie do 22 telegramów, z których każdy może zawierać ok. 4000 znaków,
- realizację funkcji autowzywaka umożliwiającego komputerowi dokonywanie połączeń z abonamentami komutowanej sieci dalekopisowej według adresów zawartych w telegramach, sprawdzanie ich znamienników oraz wysyłanie tekstów telegramów; system próbuje wysłać telegram siedmiokrotnie co 2 min, a następnie 24 razy co 4,5 min; system odnotowuje liczbę nieudanych połączeń oraz pozycję najdalszej cyfry, którą w dotychczasowych próbach zdołano „wybrać"; informacje te wraz z nie wysłanym telegramem mogą być w każdej chwili pobrane przez program użytkowy, w celu ich przechowania w zbiorze dyskowym, wydrukowania, ponownego wysłania inną drogą dalekopisową lub przesłania z powrotem komputerowi Odra.

Praca każdego z ośmiu kanałów jest odnotowywana w pięciu rejestrach zawierających:

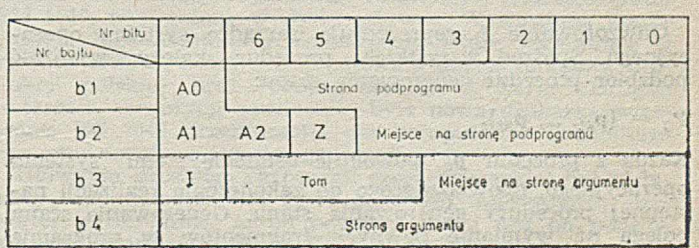
- czas pracy kanału,
- liczbę połączeń z systemem,
- liczbę przyjętych meldunków,
- liczbę nieprawidłowych połączeń,
- liczbę wysłanych telegramów.

Zawartości tych rejestrów mogą być pobierane przez programy użytkowe, w celu okresowego sporządzania statystyki pracy systemu. Statystyka ta może mieć zastosowanie przy wykrywaniu usterek technicznych, ustalaniu zbyt dużych nieprawidłowości w nadawaniu meldunków, określaniu możliwości rozszerzania oprogramowania użytkowego itp.

REALIZACJA PROGRAMÓW UŻYTKOWYCH W KODZIE POŚREDNIM

Realizacja programów w kodzie pośrednim może okazać się najlepsza nie tylko w wypadku języków algorytmicznych, o złożonej strukturze, lecz również dla języków asemblerowych, jeśli słowo maszynowe jest krótkie (np. jednobajtowe), a rozkazy wykonują operacje za pośrednictwem jednego lub kilku krótkich rejestrów podstawowych (np. jednego 8-bitowego akumulatora). W takim wypadku postać wynikowa programów użytkowych na ogół zawiera ciąg odwołań do podprogramów działających na dwu- lub wielobajtowych rejestrach zorganizowanych programowo. Podprogramy te mogą stanowić integralną część oprogramowania systemowego lub mogą być opracowywane przez użytkownika stosownie do jego potrzeb. Jeśli postać wynikowa jest ciągiem rozkazów, to odwołania do podprogramów są poprzedzane pamiętaniem śladu lub składowaniem adresów powrotu do programu na stosie. Sekwencję końcową podprogramów stanowią rozkazy powrotu według „śladu" lub adresu zapisanego na „stosie".

Jeśli natomiast postać wynikowa jest ciągiem kodów pośrednich, to odwołania do podprogramów realizuje zarządca systemu na podstawie adresów podprogramów zawartych w kodach pośrednich. Podprogramy te kończą rozkazy skoku do określonego miejsca zarządcy.



Rys. 1. Kod pośredni jednej z instrukcji

Realizację programów w kodzie pośrednim pokażę na jednej z kilkuset instrukcji języka ROMEK [2]: „DWI, DZK" — dzielenie całkowite liczby dwójkowej zawartej w 32-bitowym rejestrze programowym R1R2 przez liczbę dwójkową identyfikowaną w programie ze zmienną DZK. Wynik dzielenia jest zapisywany w R2, a reszta w R1.

Postać źródłowa tej instrukcji jest zmieniana przez translator na kod pośredni, przedstawiony na rys. 1. Kod ten zajmuje 4 bajty w pamięci procesora MOMIK 1000b. Pamięć ta zawiera 32 KB adresowanych przez określenie tomu T (od 0 do 7), strony S (od 128 do 255, a dla tomu zerowego od 0 do 127) i miejsca na stronie D (od 0 do 31).

Interpreter na podstawie wartości bitów A0, A1 i A2 ustala, że jest to instrukcja jednoargumentowa o adresie bezpośrednim lub indeksowym. Następnie, na podstawie zawartości bajtów b2, b3 i ewentualnie systemowego rejestru indeksowego pierwszego argumentu, ustala adres argumentu DZK. Wartość tego argumentu interpreter zapisuje do systemowego rejestru R i przekazuje sterowanie podprogramowi, który po podzieleniu zawartości R1R2 przez R umieszcza wynik w R2, resztę w R1 i przekazuje sterowanie zarządcy systemu operacyjnego.

Ta sama instrukcja może być realizowana jako następujący ciąg rozkazów tego procesora:
— pamiętaj ślad (1 bajt);
— przekaż sterowanie podprogramowi ustalającemu rodzaj argumentu instrukcji i pobierającego jego wartość do rejestru R (2 bajty);
— rodzaj argumentu (1 bajt);
— adres argumentu (2 bajty);
— pamiętaj ślad (1 bajt);
— przekaż sterowanie podprogramowi dzielenia R1R2 przez R (2 bajty).

Ciąg taki zajmuje 9 bajtów, chociaż postać tego zapisu wydaje się bardzo upakowana.

Analizując czasy realizacji tej instrukcji obydwoma sposobami można stwierdzić, że jest on nieco krótszy w wypadku zastosowania kodu pośredniego. O efektywności tego rodzaju programowania można się przekonać na przykładzie pakietu programów opracowanych dla potrzeb PKP [3]. Główny program tego pakietu może realizować dwukrotnie więcej funkcji, przechowując jednocześnie w pamięci operacyjnej trzykrotnie więcej danych niż analogiczny program zrealizowany uprzednio w języku wewnętrznym przez doświadczonego programistę.

Niebagatelną zaletą kodu pośredniego jest możliwość opracowania wydajnych translatorów. Transiacja programu zawierającego 1000 instrukcji trwa w systemie SOMEK ok. 30 s, a konsolidacja parę sekund. Nadzorowanie realizacji programów w kodzie pośrednim jest wygodne, zwłaszcza jeśli konstrukcja zarządcy jest analogiczna do opisanej w następnym punkcie.

RÓWNOLEGŁA REALIZACJA FUNKCJI ZARZĄDCY SYSTEMU OPERACYJNEGO

Każdy z wymienionych systemów operacyjnych funkcjonuje według następującego modelu M_{SO}:

$$M_{SO} = [\{(P_{n_i} \dots . p_k . \dots . p_1)_l\}, F_{GP}, W]$$

gdzie: $p_k \in P_k$ dla $k = 1, \dots, n_1$ przy czym
P_k jest zbiorem procedur realizowanych w fazie k, definiowanych jako określone ciągi rozkazów maszynowych,
l jest liczbą różnych funkcji zarządcy,

$F_{GP} = \{f_1, \dots, f_s\}$, a odwzorowanie f_i (dla $i = 1, \dots, s$)

ustala na podstawie wartości wskaźników W i bazy k, która procedura ma być realizowana w fazie $k + 1$.

Odwzorowanie f_i reprezentuje zarządcę systemu operacyjnego. Spośród wszystkich procedur należy wyróżnić podzbiór procedur generowania stanu:

$$P_g = \{p_{g1}, \dots, p_{gz}\}.$$

Każda z procedur p_{g_j} powoduje ustalenie stanu systemu operacyjnego, obowiązującego do zakończenia realizacji następnej procedury generowania stanu. Generowanie stanu polega na wymianie pewnych fragmentów w programie realizującym funkcje zarządcy, co znacznie skraca czas jego pracy.

Przykładowo, w jednej z faz zarządcy systemu SYPRUCZ [1] mogą być wykonywane następujące procedury:
- obsługi klawiatury DZM, po wykonaniu procedury dołączania klawiatury do systemu,
- pusta, po wykonaniu procedury odłączania klawiatury DZM podczas przetwarzania,
- zawieszenie realizacji programu, po wykonaniu procedury realizującej odpowiednią komendę operatorską.

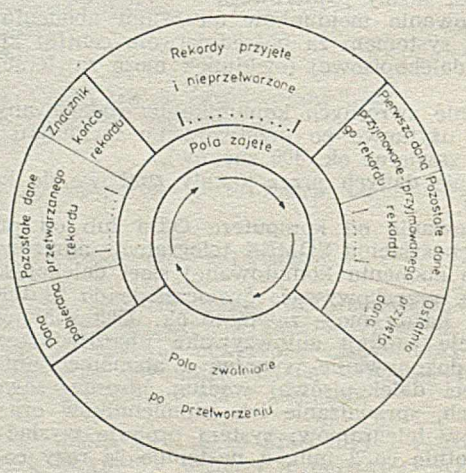
W fazie tej zarządca nie sprawdza wartości wskaźników, lecz wykonuje jedną z procedur zależnie od wygenerowanego stanu. Pracuje on więc w sposób „inteligentny", uzależniając sposób „dedukcji" od istniejących warunków zewnętrznych.

Taka konstrukcja umożliwia dogodne wkomponowanie interpretera kodu pośredniego w ciało programu zarządcy. Na przykład w systemie SOMEK, w jednej z faz jest realizowana jedna z procedur:
— zapisania wartości argumentu według adresowania natychmiastowego do rejestru systemowego **R**, jeśli wartość bitu **A1 = 0** i **A2 = 1**,
— ustalenie bezpośredniego adresu argumentu instrukcji i zapisanie tego argumentu do **R**, gdy wartość bitu **A1 = 1** i **A2 = 1**.

Stosowanie kodu pośredniego i równoległej realizacji funkcji zarządcy umożliwia m.in.:
— tworzenie efektywnych programów użytkowych,
— pełną kontrolę realizowanego programu użytkowego (żaden błąd w programie użytkowym nie może spowodować wymknięcia się programu spod pełnej kontroli systemu),
— rozszerzenie zarządcy o nowe funkcje, bez wydłużania czasu pracy programów użytkowych,
— sterowanie pracą programu w trybie pracy krokowej (wstrzymanie dalszej realizacji po wykonaniu każdej instrukcji) lub ciągłej,

Po każdym wstrzymaniu dalszej realizacji programu jest możliwe:
- kontynuowanie w dowolnym trybie,
- uzyskiwanie informacji o wartości rejestrów programowych pól w pamięci operacyjnej, zawartości buforów,
- korekta tekstu programu, ponowna translacja i ponowne jego uruchomienie,
- wykonywanie komend operatorskich.

Istotne korzyści można wyciągnąć również ze skrócenia cyklu przetwarzania do jednej fazy. W opracowanych systemach cykl ten nie przekracza 50 ms. Dzięki temu można było opracować efektywną wymianę danych między urządzeniami zewnętrznymi a procesorem — w systemie SYPRUCZ na zasadzie sprawdzania gotowości urządzeń, a w systemie SOMEK na zasadzie wymiany z przerwaniami. Przyjęcie przerwania w systemie SOMEK jest możliwe tylko na początku każdego cyklu, w dalszej części cyklu przerwania są maskowane. Dzięki takiemu rozwiązaniu nie muszą być pamiętane wartości rejestrów procesora i ich regeneracja, co znacznie zwiększa efektywność obsługi przerwania. Równoległa realizacja funkcji zarządcy i podział na fazy okazały się również bardzo korzystne przy uruchamianiu systemów operacyjnych. Wstawienie jednego rozkazu stopu pozwoliło kontrolować przebieg wszystkich funkcji zarządcy z dokładnością do jednej procedury.



Rys. 2. Zajmowanie i zwalnianie pól bufora zorganizowanego cyklicznie

CYKLICZNA ORGANIZACJA BUFORÓW URZĄDZEŃ WEJŚCIOWYCH

Przyjmowanie i przetwarzanie danych za pośrednictwem buforów zorganizowanych cyklicznie, zrealizowano przy użyciu tablic, z których każda zawiera:
— adres początku bufora,
— adres końca bufora,
— adres pola, do którego zostanie przyjęta następna dana,
— adres danej pobieranej do przetwarzania,
— adres początku przyjmowanego rekordu,
— znacznik końca rekordu,
— znacznik kasacji rekordu.

Cykliczna organizacja bufora polega na tym, że procesy wprowadzania i pobierania znaków po dotarciu do końca bufora rozpoczynają się od jego początku. System udostępnia do przetwarzania rekordy, które w całości zostały przyjęte lub które wypełniły cały bufor. Przyjmowane dane zajmują, a przetworzone zwalniają kolejne pola bufora w sposób zilustrowany na rys. 2. Taka organizacja wydaje się bardzo korzystna, gdy przyjmowane dane muszą być natychmiast przetwarzane, a długości rekordów mogą być bardzo zróżnicowane.

LITERATURA

[1] Gąsiorek W.: Programowanie minikomputerów MERA 300 w systemie SYPRUCZ. Skrypt. Politechnika Krakowska, Kraków, 1983

[2] Gąsiorek W., Piotrowski S.: Opracowanie projektu i oprogramowanie systemu operacyjnego oraz translatora języka typu ASSEMBLER dla minikomputera MERA 306. Praca naukowo-badawcza. Politechnika Krakowska, Kraków, 1984

[3] Gąsiorek W., Piwkowski A., Puchała M.: Pakiet programów R7 dla sporządzania wykazu wagonów w składzie pociągu na minikomputerze MERA 303. Praca naukowo-badawcza. Politechnika Krakowska, Kraków, 1983.

Robotron na LFM '87

W dniach od 15 do 27 marca 1987 r., a więc bezpośrednio po hanowerskim CeBIT, odbyła się w ramach Wiosennych Targów Lipskich (LFM '87) kolejna w międzynarodowym kalendarzu, wielka ekspozycja przemysłu informatycznego. W ekspozycji tej tradycyjnie dominowali gospodarze, którzy dzięki przeprowadzonej na przestrzeni kilkunastu lat konsekwentnej koncentracji tego przemysłu w ramach kombinatu Robotron, osiągnęli w tej dziedzinie imponujące wyniki i wysunęli się pod względem wszechstronności oferty oraz możliwości eksportowych niewątpliwie na czołowe miejsce wśród krajów RWPG. Wyraźny wzrost rozmiarów ekspozycji w porównaniu z rokiem poprzednim można było stwierdzić w ofertach Węgier, Czechosłowacji i ZSRR, a nawet Rumunii, natomiast oferty Polski i Bułgarii były wyjątkowo skromne i znacznie poniżej faktycznych możliwości tych krajów. Zauważalne było ponowne pojawienie się w Lipsku nie tylko niektórych producentów z czołówki światowej, jak Siemens, Hewlett-Packard i Xerox, ale również nowych, głównie japońskich, wytwórców sprzętu mikrokomputerowego, jak Epson i Toshiba, oraz znacznej liczby mniejszych firm z Zachodu, które nie odstraszone wszechstronnością oferty Robotronu uznały za celowe i opłacalne pojawienie się również w Lipsku.

Szczupłość miejsca, jakie możemy przeznaczyć na relację z LFM '87, nie pozwala na szczegółowe omówienie wszystkich najważniejszych punktów wyjątkowo obszernej w tym roku ekspozycji, którą zlokalizowane nie tylko w trzech dużych halach tematycznych (nr 15, 17 i 18), ale również w niektórych pawilonach i stoiskach narodowych (np. ZSRR). Ze względu na wszechstronność oferty, odzwierciedlającą w pełni współczesne tendencje rozwojowe informatyki, oraz wieloletnią tradycję obecności sprzętu produkcji NRD na rynku polskim, skoncentruję się głównie na scharakteryzowaniu oferty kombinatu Robotron.

Scenariusz tej oferty, prezentowanej w całości w hali nr 15, został podporządkowany zastosowaniom CAD/CAM, automatyzacji prac biurowych oraz problematyce sieciowej. Mimo zdalnego wspomagania niektórych demonstracji dużymi systemami komputerowymi Jednolitego Systemu, w ekspozycji oczywiście był widoczny wyłącznie sprzęt mikrokomputerowy.

Czołowym nowym produktem była ulepszona wersja 16-bitowego mikrokomputera A 7100, który dzięki wprowadzeniu procesora arytmetycznego oraz rozszerzeniu pojemności pamięci zewnętrznej (dysk sztywny do 50 MB), znacznie zwiększył swą moc obliczeniową w porównaniu z wersją pierwotną. Komputer wyposażono w dwa systemy operacyjne: CDP 3.11 (zgodny z MS-DOS) oraz MUTOS 1700 (zgodny z Unix). Systemy te zapewniają dostęp do olbrzymiego obszaru istniejącego oprogramowania narzędziowego i użytkowego, w tym graficznego oraz do przetwarzania tekstów. Akcentowano ukierunkowanie A 7100 na obszar zastosowań CAD/CAM, dla których oferowano urządzenia peryferyjne również produkcji Robotronu: plotery formatu A2 i A3, stolik graficzny formatu A4 oraz drukarkę graficzną K 6313/14. Takie ukierunkowanie mikrokomputera powoduje, że A 7100 jest określany przez producenta jako komputerowe stanowisko pracy (niem. Arbeitsplatzcomputer).

Rozwój sprzętu 16-bitowego nie wyeliminował oczywiście dotychczasowej linii konstrukcji 8-bitowych, które zaprezentował ulepszony mikrokomputer osobisty 1715 W z rozszerzoną do 256 KB pamięcią operacyjną, ukierunkowany głównie na obszar zastosowań biurowych. Wyraziło się to we wprowadzeniu do klawiatury specjalnych, dostosowanych do obsługi procesora tekstów, klawiszy funkcyjnych oraz niemieckich znaków narodowych, w użyciu nowego modelu drukarki rozetkowej 1152/257, a także w znacznym rozszerzeniu specjalistycznego oprogramowania narzędziowego i użytkowego.

Trzecim, zasadniczym akcentem ekspozycji Robotronu, był podsystem graficzny EC 7945, dostosowany jak wskazuje symbol tego urządzenia, do bezpośredniej współpracy z komputerami Jednolitego Systemu w zakresie zastosowań grafiki. Trzonem podsystemu jest terminal graficzny, złożony z ekranu o dużej rozdzielczości oraz specjalizowanej klawiatury. W zależności od potrzeb użytkownika, istnieje możliwość rozszerzania podsystemu o kompletny asortyment graficznych urządzeń peryferyjnych: tablicę graficzną, koordynatograf oraz dwa rodzaje ploterów.

Na podkreślenie zasługuje fakt, że wszystkie bez wyjątku składniki podsystemu są produktami Robotronu, co jest najlepszym potwierdzeniem olbrzymich możliwości konstrukcyjnych i produkcyjnych tej firmy. EC 7945 został już wszechstronnie sprawdzony we współdziałaniu z systemami komputerowymi EC 1055.M oraz EC 1057. Istnieje również możliwość jego przyłączenia do innych modeli drugiej i trzeciej generacji komputerów JS.

Interesującym akcentem ekspozycji Robotronu była demonstracja działania lokalnej sieci ROLANET 1, zbudowanej z mikrokomputerów PC 1715 oraz minikomputera K 1630. Tematem demonstracji była automatyzacja prac biurowych. Prezentowana sieć jest pierwszym produktem perspektywicznego planu rozwoju sieci komputerowych, w realizacji którego Robotron ściśle współdziała z kombinatem przemysłu środków łączności Nachrichtenelektronik.

Oprócz systemów mikrokomputerowych Robotron zaprezentował wiele nowości z coraz bogatszego asortymentu drukarek. Po raz pierwszy eksponowano nową rodzinę graficznych drukarek mozaikowych K 6320 o szybkości do 165 zn./s, której np. model K 6328 pozwala na druk wysokiej jakości (siatka 18×36 punktów). Również w grupie drukarek rozetkowych serii 1152 pokazano nowy model 257 o szybkości 55 zn./s. Coraz bardziej interesująca jest także oferta w dziedzinie elektronicznych maszyn do pisania, których cechy są już konkurencyjne w stosunku do podobnych konstrukcji zachodnich.

Na równi z ofertą sprzętową Robotron prezentował imponujący asortyment nowoczesnego oprogramowania, obejmując praktycznie wszystkie jego rodzaje. Na podkreślenie zasługuje fakt, że opiera się ono na rozwiązaniach własnych, z pełnym jednak uwzględnieniem światowych trendów rozwoju. Godna uznania jest bardzo starannie opracowana dokumentacja, a także liczne placówki usług konsultacyjnych.

Niewątpliwa dominacja ekspozycji NRD nie mogła przysłonić interesujących, lecz z trudem docierających do naszego kraju osiągnięć pozostałych naszych sąsiadów. Wymienić tu należy radziecki 16-bitowy mikrokomputer SM 1810, wyposażony w dyski sztywne. Wystawiona konfiguracja tego komputera akcentowała silnie współpracę w ramach RWPG, którą dokumentowało użycie monitora produkcji polskiej oraz robotronowskiej drukarki. Drugim akcentem ekspozycji radzieckiej była nowa drukarka wierszowa EC 7040 przeznaczona dla komputerów Jednolitego Systemu drugiej i trzeciej generacji.

Głównym akcentem ekspozycji czechosłowackiej był 32-bitowy minikomputer SM 52-12 o bardzo interesującej charakterystyce. Mimo, że od pewnego czasu jest on podstawowym szlagierem firmy KOVO, na naszym rynku jest praktycznie nieznany.

WŁADYSŁAW KLEPACZ

PIOTR ZIELCZYŃSKI
Środowiskowe Centrum Obliczeniowe
CYFRONET
Otwock-Świerk

Turbo Prolog

Jednym z ostatnich produktów wprowadzonych na rynek przez firmę Borland International (1986 rok) jest Turbo Prolog [7] — system zawierający kompilator Prologu wraz ze środowiskiem programowania w tym języku. Turbo Prolog jest przeznaczony dla mikrokomputerów typu IBM PC z systemem operacyjnym MS-DOS/PC-DOS (wersja 2.0 wzwyż). Jedynym wymaganiem sprzętowym jest pamięć o pojemności powyżej 384 KB. Produkt jest dostarczany na dwóch dyskietkach. Pierwsza z nich zawiera podstawowy plik PROLOG.EXE, komunikaty o błędach i pliki systemowe. Na drugiej dyskietce znajduje się kilkadziesiąt przykładowych programów oraz biblioteka wykorzystywana przy łączeniu modułów.

Niniejszy artykuł zawiera charakterystykę Turbo Prologu i próbę jego oceny na tle kilku innych translatorów Prologu dostępnych na IBM PC.

OGÓLNA CHARAKTERYSTYKA

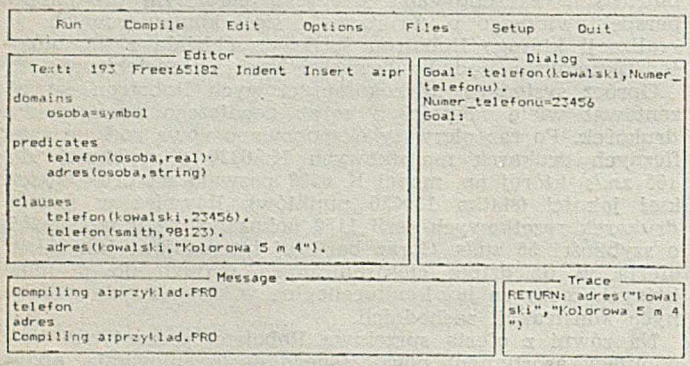
Korzystanie z Turbo Prologu jest bardzo wygodne dzięki obecności list menu oraz systemu okien. Kilka przykładowych okien przedstawiono na rys. 1:

— okno **Editor**, w którym pisze się i poprawia program,

— okno **Dialog** do wywoływania programu i prowadzenia konwersacji,

— okno **Message**, w którym pojawiają się różne komunikaty (o błędach, dotyczące ładowania, kompilacji itp.),

— okno **Trace** wykorzystywane w czasie śledzenia programu.



Rys. 1. Okna w Turbo Prologu

Program w Turbo Prologu składa się z kilku sekcji identyfikowanych za pomocą słowa kluczowego. Najczęściej stosowane są sekcje: **predicates, clauses, domains i goal** (rys. 2).

```
domains
    pracownik=symbol
    pensja=real
predicates
    zarobki(pracownik,pensja)
    szczesliwy(pracownik)
goal
    szczesliwy(Kto),write(Kto).
clauses
    szczesliwy(Pracownik) if
        zarobki(Pracownik,X) and
        X > 50000 .
    zarobki(wojtek,25000).
    zarobki(rysiek,51000).
    zarobki(karol,39000).
```

Rys. 2. Program w Turbo Prologu

W sekcji **predicates** podaje się spis predykatów sprawdzających zachodzenie określonej relacji między argumentami. Przy deklarowaniu predykatu należy podać dziedziny jego argumentów, np.:

zarobki(symbol, real)

deklaruje predykat „zarobki", którego pierwszym argumentem jest symbol, drugim — liczba rzeczywista.

Zasadniczą sekcją programu jest sekcja **clauses**, w której definiuje się wszystkie klauzule. Dzielą się one na fakty i reguły. Fakty określają relacje między obiektami, np. fakt:

zarobki(wojtek, 25000)

informuje, że obiekt „wojtek" jest w relacji „zarobki" z liczbą 25000.

Reguły opisują warunkowe zachodzenie relacji, np.:

```
szczęśliwy(Pracownik) if
  zarobki(Pracownik, X) and
  X > 50000.
```

Powyższa reguła w tłumaczeniu na język potoczny może brzmieć: pracownik jest szczęśliwy, jeśli jego zarobki są większe niż 50000. Zamiast słów kluczowych „if", „and" oraz „or" można używać odpowiednio znaków ":=", "," oraz ";". W celu odróżnienia stałych od zmiennych zastosowano następującą konwencję: zmienne rozpoczynają się dużymi literami po których następuje ciąg liter, cyfr lub znaków podkreślenia "_", stałe rozpoczynają się małymi literami. Konwencja ta jest taka sama, jak np. w Prologu V [4], natomiast różni się ona znacznie od konwencji w MicroPrologu [2, 5], gdzie do oznaczenia zmiennych używa się określonych liter (x, y, z,...).

Sekcja **domains** służy do deklarowania dziedzin argumentów predykatów, np. poniższa sekwencja deklaruje dziedzinę „tytuł książki" jako napis oraz listę napisów, liczb całkowitych i symboli (gwiazdka oznacza złożoną z danych elementów):

```
tytuł_książki = string
lista_książek = tytuł_książki *
lista_zarobków = integer *
lista_pracowników = symbol *
```

Dziedzin zadeklarowanych w tej sekcji można używać następnie do deklarowania predykatów, np.:

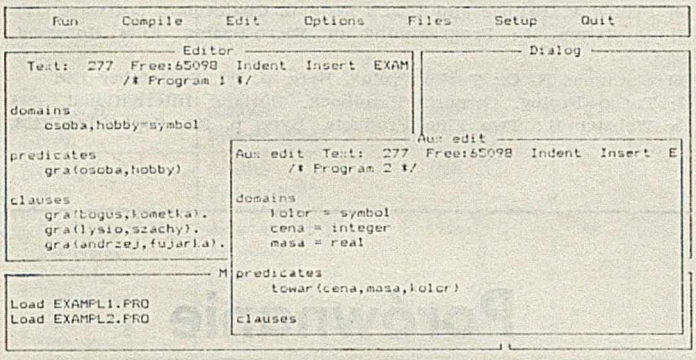
```
biblioteka (lista_książek)
księgowość (lista_zarobków, lista_pracowników).
```

W sekcji **goal** wywołuje się program, zadając pytanie, na które system powinien odpowiedzieć korzystając z informacji zapisanych w klauzulach, np.:

zarobki(wojtek,30000).	Czy zarobki Wojtka wynoszą 30000?
zarobki(Kto,25000).	Kto zarabia 25000?
zarobki(_,50000).	Czy ktokolwiek zarabia 50000?
zarobki(karol,X) and X > 40000.	Czy Karol zarabia ponad 40000?
zarobki(rysiek,Ile).	Ile zarabia Rysiek?

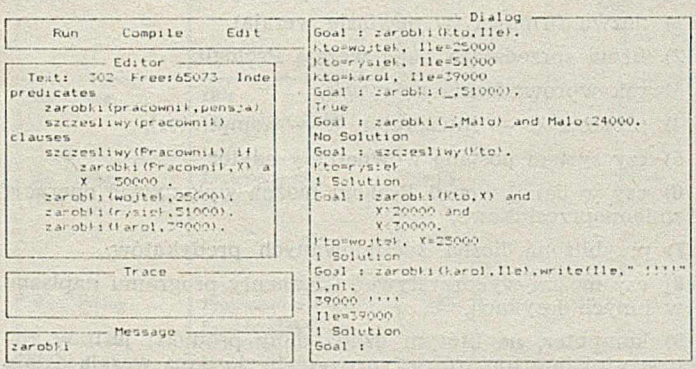
Oprócz omówionych sekcji mogą wystąpić jeszcze sekcje **global domains** i **global predicates** (do deklaracji dziedzin i predykatów w programach złożonych z modułów) oraz sekcja **database** deklarująca predykaty, które mogą być wpisywane i usuwane z bazy danych.

Nowy program można wpisywać do pamięci za pomocą edytora, wywołanego opcją **Edit** z głównego menu. W edytorze dostępnych jest wiele wygodnych poleceń (np. różne warianty poruszania kursorem, operacje na blokach, wpisywanie, kasowanie, znajdowanie i poprawianie ciągu znaków, skok do określonej linii itp.). Dostępny jest również edytor pomocniczy, którego można użyć na przykład do wstępnego poprawienia tekstu przed wprowadzeniem go do głównego edytora (rys. 3). Można także korzystać z opcji **help**, powodującej wyświetlenie się spisu podstawowych poleceń. Osoby przyzwyczajone do edytora WordStar lub Turbo Pascala mogą używać występujących w nich poleceń, których większość jest zgodna z poleceniami edytora Turbo Prologu. Wyjście z edytora następuje po naciśnięciu klawisza ESC lub F10.



Rys. 3. Edytor pomocniczy

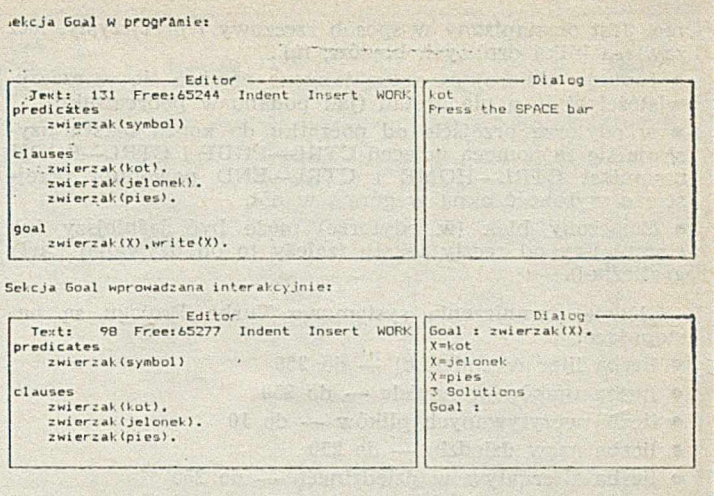
Po napisaniu programu można przystąpić do kompilowania. Skompilowany program zapisuje się bezpośrednio do pamięci lub też tworzy się niezależny plik o rozszerzeniu EXE (lub OBJ). W wypadku błędu w czasie kompilacji wyświetlany jest odpowiedni komunikat, a kursor wskazuje miejsce w programie, gdzie został wykryty błąd. Sterowanie jest przekazywane wówczas automatycznie do edytora, aby osoba uruchamiająca program mogła od razu wnieść poprawkę.



Rys. 4. Przykład konwersji w oknie Dialog

Jeśli sekcja **goal** występuje w programie, to program wykonuje się bezpośrednio po kompilacji. Jeśli nie, to po skompilowaniu programu w oknie **Dialog** ukazuje się napis „Goal:". Wówczas program wywołuje się interakcyjnie (rys. 4). Działanie programu jest zależne od tego czy sekcja **goal** zostanie umieszczona w programie, czy będzie wprowadzona interakcyjnie. W wypadku umieszczenia sekcji **goal** w programie — wyświetlana jest tylko jedna odpowiedź, a interakcyjnego wywołania — wszystkie możliwe odpowiedzi (rys. 5).

W trakcie uruchamiania programu warto używać śledzenia. Służy do tego instrukcja „trace" umieszczona przed programem. Można śledzić wszystkie lub tylko wybrane predykaty. W trakcie śledzenia kursor wskazuje klauzulę, która jest aktualnie wywoływana, a w oknie **Trace** wypisują się kolejne wywołania predykatów (rys. 1). Naciskając klawisz F10 można przejść do kolejnego kroku rozwiązywania. Standardowe wymiary okna **Trace** są zbyt małe, co powoduje załamywanie wyświetlanego tekstu i zmniejsza jego czytelność. Warto więc poszerzyć to okno na całą



Rys. 5. Różnica pomiędzy zamieszczeniem sekcji goal w programie i wprowadzeniem jej interakcyjnie

szerokość ekranu. Aby zmienić wymiary lub położenie okna, należy z głównego menu wybrać opcję **Setup**, a następnie **Windows**, po czym klawiszami kursora i klawiszem SHIFT ustawić położenie narożników.

PRÓBA OCENY SYSTEMU

Zaletą Turbo Prologu jest duża liczba standardowych predykatorów umożliwiających m.in. wykonywanie operacji na plikach, bezpośredni dostęp do pamięci, operacje bitowe definiowanie okien, syntezę dźwięku, operowanie napisami itp. Zdefiniowane są podstawowe funkcje matematyczne: logarytmy, pierwiastek, e do potęgi x, reszta i część całkowita z dzielenia, wartość bezwzględna, funkcje trygonometryczne. W programach graficznych można stosować zarówno układ kartezjański, jak i grafikę żółwia. Zapewniona jest również możliwość łączenia modułów napisanych w innych językach (np. C, Pascal, Fortran). Dzięki istnieniu predykatu „edit" wiele funkcji edytora jest dostępnych nawet po utworzeniu niezależnego od środowiska programu binarnego. Z wnętrza programu można wywołać również polecenia systemu DOS.

Niestety, nie wszystkie predykaty występujące w poprzednich wersjach Prologu dla komputerów IBM PC (np. Prolog 1 [1], Prolog V [4], MicroProlog [2, 5]) są zdefiniowane w Turbo Prologu. Brakujące predykaty można zwykle w niezbyt skomplikowany sposób zdefiniować we własnym zakresie, np. występujący w Prologu V predykat
name(Wyraz, Lista-liter),
który zmienia wyraz na listę występujących w nim liter, może być zdefiniowany w Turbo Prologu za pomocą predykatu frontchar(Wyraz,Pierwsza-litera,Reszta) dzielącego wyraz na pierwszy znak i resztę. Wiele predykatów Turbo Prologu ma nazwy inne niż analogiczne predykaty w pozostałych Prologach, co pogłębia niezgodność poszczególnych wersji. Praktycznie uniemożliwia to adaptowanie w Turbo Prologu dużych, skomplikowanych programów napisanych w innych odmianach języka.

Sama składnia klauzul jest bardzo zbliżona do standardu proponowanego przez Clocksina i Mellisha [3] oraz zastosowanego w Prologu V. Różni się jednak znacznie od składni MicroPrologu. Przykładowo, klauzula pytająca o wszystkich chłopców będących dziećmi Jacka wygląda w poszczególnych wersjach następująco:

Turbo Prolog: ojciec(jacek,Kto) and chłopiec(Kto).
Prolog V: ojciec(jacek,Kto) and chłopiec (Kto)?
MicroProlog: which(x: Jacek ojciec x & x chłopiec).

Zdecydowaną przewagę nad pozostałymi wersjami Prologu ma środowisko Turbo Prologu umożliwiające wykonywanie wielu czynności bez konieczności wychodzenia z systemu (redagowanie programu, jego uruchamianie i testowanie, zapisywanie i odczytywanie z dysku itp.). System okien umożliwia, na przykład, jednoczesne obserwowanie treści programu i efektów jego wykonania. Wymiary, położenie i kolory okien można dopasować do własnych potrzeb.

Podręcznik Turbo Prologu [7] zawiera pełny opis systemu i wiele przykładów ilustrujących technikę programowa-

nia. Jest on napisany w sposób rzeczowy i przejrzysty, lecz zawiera kilka drobnych błędów, np.:

- funkcja abliczająca arcus tangens nazywa się w rzeczywistości atan, a nie arctan (jak podano w podręczniku),
- w edytorze przejście od początku do końca tekstu uzyskuje się za pomocą poleceń CTRL—PGUP i CTRL—PGDN, natomiast CTRL—HOME i CTRL—END powodują przejście o wysokość okna w górę i w dół,
- oznaczony blok (w edytorze) może być jaśniejszy lub ciemniejszy od reszty tekstu (zależy to od używanej karty graficznej).

Główne ograniczenia systemowe Turbo Prologu są następujące:

- liczba liter w zmiennej — do 250
- liczba znaków w napisie — do 250
- liczba wczytywanych plików — do 10
- liczba nazw dziedzin — do 250
- liczba alternatyw w dziedzinach — do 250
- liczba nazw predykatów — do 300
- liczba parametrów w predykacie — do 50
- liczba zmiennych w klauzuli — do 100
- liczba klauzul dla predykatu — do 500.

Powyższe wartości są na tyle duże, że nie stanowią istotnego ograniczenia w większości praktycznych zadań. W razie konieczności ominięcia któregoś z tych ograniczeń, można posłużyć się pewnym prostym wybiegiem. Na przykład, jeśli w dużym systemie ekspertowym trzeba użyć więcej niż 500 klauzul dla jednego predykatu, to deklarując dwa równoważne predykaty o różnych nazwach:

predykat1(Argumenty):—predykat2(Argumenty).

uzyskuje się zwiększenie dopuszczalnej liczby klauzul do 999.

W Turbo Prologu zmienne całkowicie mogą przyjmować wartości od —32786 do +32786 (podobnie, jak w Prologu V i większości innych). Czasami zakres ten jest niewystarczający. Wówczas trzeba zadeklarować odpowiednią zmienną jako liczbę rzeczywistą, co pozwala operować liczbami, które w zapisie potęgowym mają wykładnik z przedziału ⟨—308,+308⟩.

Programy skompilowane za pomocą Turbo Prologu wykonują się znacznie szybciej od analogicznych programów uruchamianych za pomocą interpreterów Prologu (np. wspomnianego Prologu 1, Prologu V i MicroPrologu). Skompilowane programy mają dostęp do większego stosu i wykorzystują większą głębokość rekurencji [6]. Kompilator Turbo Prologu jest jednak zbyt czuły na pewne mało istotne błędy (np. sygnalizuje błąd, gdy nie zostanie użyty predykat zadeklarowany w sekcji **predicates**). Innym mankamentem kompilatora jest niejasność niektórych komunikatów o błędach (nie są one opisane nawet w podręczniku).

Podstawową wadą Turbo Prologu jest konieczność definiowania typów zmiennych. Umożliwia to m.in. korzystanie z list mieszanych (zawierających kilka różnych typów zmiennych). Problem ten można częściowo ominąć, deklarując obiekt złożony „element", który składa się z funktora określającego typ zmiennej i samej zmiennej w nawiasach, np.:

```
domains
  element = z(char),c(integer),r(real),  l(string),s(symbol)
  lista = element *
```

gdzie z oznacza znak, c — liczbę całkowitą, r — liczbę rzeczywistą, l — napis, s — symbol. Każdy element takiej listy musi być poprzedzony funktorem określającym dziedzinę danego elementu, np.:

[z(a),c(2)r(3.33),l(„czwarty element"), ostatni_element)].

W Turbo Prologu jest również niepełna obsługa dynamicznej bazy danych. Można wprawdzie wypisywać i usuwać z pamięci fakty, nie można natomiast wykonywać tych operacji na regułach [8]. Predykatem służącym do wpisywania faktów do bazy danych jest „asserta". Dopuszczalne jest na przykład:

asserta(ssak(lew))

lecz niedozwolone:

asserta(zwierzak(X):—ssak(X)).

Pewnym mankamentem Turbo Prologu jest również brak mechanizmów dealokacji pamięci (ang. garbage collection), co powoduje szybsze zapełnienie się dostępnej pamięci.

Pomimo wymienionych wad, Turbo Prolog jest wygodnym i mocnym narzędziem programowania logicznego na mikrokomputerach IBM PC, używając go musimy jednak pamiętać o wielu wymienionych powyżej ograniczeniach.

LITERATURA

[1] Burnham W. D., Hall A. R.: Prolog Programming and Applications. Macmilian Education, 1985
[2] Clark K. L., Ennals J. R., McCabe F. G.: Micro-Prolog, 1985
[3] Clocksin W. F., Mellish C. S.: Programming in Prolog. Springer, 1984
[4] Prolog V. Primer and User's Manual. Chalcedony Software, 1985
[5] de Saram H.: Programming in Micro-Prolog. Ellis Horwood, 1985
[6] Shammas N. C.: Turbo Prolog, Byte, p. 293, September 1986
[7] Turbo Prolog. Owner's Handbook. Borland International, 1986
[8] Webster B.: Two Fine Products. Byte, p. 355, September 1986.

Porównanie dostępnych wersji Prologu

W tabeli porównano kompilatory i interpretery języka Prolog dostępne aktualnie na rynku. Kolejne rubryki oznaczają:

1) nazwa firmowa (w nawiasie wersja),
2) firma sprzedająca daną wersję Prologu,
3) rok wprowadzenia na rynek,
4) procesor (i — interpreter, k — kompilator),
5) czy system posiada wewnętrzny edytor,
6) czy w danej wersji Prologu można wykonywać operacje zmiennoprzecinkowe,
7) przybliżona liczba zdefiniowanych predykatów,
8) czy można wykorzystywać fragmenty programu napisane w innych językach,
9) komputer, na którym działa dany produkt; jeśli w rubryce tej figuruje „różne", oznacza to, że dana wersja działa na dowolnym komputerze posiadającym odpowiedni system operacyjny,
10) system operacyjny,
11) wymagana pamięć (w kilobajtach),
12) cena (w dolarach USA).

Minus w rubryce oznacza brak danych.

Wszystkie wymienione wersje Prologu, oprócz VM/Prologu i pierwszej wersji MicroPrologu, mają standardową składnię. Różnią się jednak między sobą możliwościami i wbudowanymi predykatami. Na przykład: Arity Prolog, Turbo Prolog, Prolog 86 Plus i MP Prolog mają rozwinięte operacje napisowe. Możliwości definiowania okien występują, np. w MAC Prologu, IF Prologu, Prologu 86 Plus, Turbo Prologu i Prologu 2. Grafika wbudowana jest między innymi w Prolog/m współpraca z myszką), Prolog/i, IF/Prolog, Turbo Prolog i MP Prolog. Sigma Prolog ma możliwość emulacji innych interpreterów. Quintus Prolog ma wbudowany mechanizm sprawdzania stylu (ang. style checker). Wiele wersji Prologu posiada możliwość śledzenia wykonania programu (np. Prolog-1, Arity Prolog, Turbo Prolog, ED Prolog). Niektóre produkty są rozwinięciem innych produktów tej samej firmy, np. Prolog/i jest kontynuacją Pro-

Nazwa (wersja)	Firma	Rok	Procesor	Edytor	Zmiennoprzecinkowy	Liczba predykatów	Inne języki	Komputer	System operacyjny	Wymagana pamięć	Cena
Prolog-1 (2.2)	Expert Systems International (USA)	1983	i	T	T	150 175	T	IBM PC VAX różne	DOS VMS CP/M-80	128 — 64	95 3 275 95
Prolog-2 (1.21)	Expert Systems International (USA)	1985	k+i	T	T	320	T	IBM PC	DOS	384	450 (i) 895 (k)
PC-Prolog (1.1)	SU-TVT Infologics (Szwecja)	1984	i	T	T	130	N	IBM PC VAX 11/780 SUN-2	DOS UNIX 4.2 UNIX 2.0	192 — —	295 1 500 1 500
Micro Prolog	Programming Logic Systems (USA)	1982	i	T	T	70	T	IBM PC różne C64	DOS,CPM86 CP/M-86 CP/M-80	128 128 64 64	250 250 125 125
		1986	k+i	T	T	100	T	IBM PC	DOS	256	—
Prolog-86 (2.1)	Solution Systems (USA)	1983	i	T	N	130	N	IBM PC	DOS	256	95
Prolog-86 Plus (1.0)	Solution Systems (USA)	1986	i	T	T	200	N	IBM PC	DOS	256	250
PD Prolog (1.9J) ED Prolog FS Prolog VMI Prolog VML Prolog VMA Prolog	Automata Design Associates (USA)	1985	i	N	T	220	T	IBM PC	DOS	256 256 256 320 320 320	0 29,95 49,95 100 200 250
Arity Prolog (4.0)	Arity Corporation (USA)	1985	k+i	N	T	200	T	IBM PC	DOS	512	795 (i+k) 350 (i)
Prolog-KABA	ASTEC, Inc. (Japonia)		i	T	N	200	N	wiele	DOS	256	400
Turbo Prolog (1.0)	Borland International (USA)	1986	k	T	T	75	T	IBM PC	DOS	384	99,95
Prolog/i (1.11)	Chalcedony Software (USA)	1986	i	T	T	120	T	IBM PC	DOS	256	69,95
Prolog/m (1.11)	Chalcedony Software (USA)	1986	i	T	T	120	T	Apple McIntosh	własny	—	99,95
IF/Prolog (3.0)	Interface Computer (RFN)	1983	k+i	N	T	250	T	IBM PC MicroVAX Apollo, Sun Tektronix HP 150	DOS AOS/VS3 UNIX DOS	256 512 512 256	550 (i) 2 700 2 700 (i) 4 050 (k) 550 (i)
VM/Prolog	IBM Corporation (USA)	1985	i	N	T		T	IBM 370 IBM 43xx IBM 30xx	IBM VM/SP	1400	8 000
MP Prolog (2.2)	Logicware, Inc. (Kanada)	1984	k+i	T	T	150	T	IBM 30xx IBM 43xx VAX 7xx Sun, Apollo IBM PC	VM/CMS MVS/TSO VMS,UNIX UNIX DOS	2000 2000 2000 2000 512	24 000 24 000 16 900 5 650 495
MAC Prolog	Programming Logic Systems (USA)	1985	k+i	T	T	100	N	Apple McIntosh	własny	512	495
Sigma Prolog	Programming Logic Systems (USA)	1982	i	T	T	90	T	oparte na 68000 VAX	UNIX VMS,UNIX	768 768	695 2 700
Quintus Prolog	Quintus Computer Systems (USA)	1985	k+i	T	T	175	T	Sun, Xerox VAX, Apollo	UNIX	1000	—

logu V i Prologu V-Plus, Prolog 2 jest kompatybilny z Prologiem 1. Kolejne wersje interpreterów firmy Automata Design Associates rónią się między sobą nowymi możliwościami: PD Prolog jest najprostszą wersją rozprowadzaną za darmo, ED Prolog ma mechanizm śledzenia, FS Prolog ma możliwość zapisu klauzul w dowolne miejsce pamięci, a pozostałe wersje wykorzystują pamięć wirtualną.

Opracował: PIOTR ZIELCZYŃSKI na podstawie książki Susan Garavaglia „Prolog. Programming Techniques and Applications", Harper and Row Publishers, New York, 1987

ZDZISŁAW PŁOSKI
Instytut Informatyki
Uniwersytet Wrocławski

Chiwriter — procesor tekstu (2)

W drugiej części artykułu dokończono omawianie właściwości programu oraz przedstawiono jego ocenę na tle innych procesorów tekstu.

Formatowanie

Działanie formatowania odnosi się do jednostki tekstu zwanej akapitem. W *Chiwriterze* akapit (ang. paragraph) oznacza porcję tekstu pomiędzy dwoma kolejnymi znakami CR wprowadzonymi przez użytkownika (przyciśnięciami klawisza ENTER). Znaki te są odpowiednio symbolizowane przez program w prawej skrajnej kolumnie ekranu i odróżniają się od znaków nowych linii wprowadzanych i nadzorowanych automatycznie. Efekt automatycznego formatowania jest widoczny już przy pisaniu kolejnych wierszy, chyba że za pomocą operacji CTRL-J zakaże się wyrównywania prawego marginesu (rysunek). Przy oddziaływaniu tej operacji (równanie kolumn), w wierszu statusów, na ekranie jest wyświetlany napis JUST. Wyrównywanie marginesów polega na proporcjonalnym zwiększaniu odstępów między słowami. Dlatego też odstępy wpisane przez użytkownika (ang. hard spaces) są — dla informacji — zaznaczane na ekranie małymi kropeczkami, w celu odróżnienia od odstępów, którymi zarządza program. Odstępy wprowadzone przez użytkownika (także wielokrotne) nigdy nie zostaną usunięte przez program automatycznie.

```
Zaczynasz z wielkim entuzjazmem i aprobatą, wszystko
jest wtedy cudowne. Potem zauważasz, że inne
obiecują więcej ( - ). Powstaje pytanie czy warto
zrywać związki, w które zainwestowałeś miesiące, dla
jakiejś nowej sztuczki z przestawianiem słów,...
```

```
Zaczynasz z wielkim entuzjazmem i aprobatą, wszystko jest wtedy cudowne. Potem zauważasz, że inne
obiecują więcej ( - ). Powstaje pytanie czy warto
zrywać związki, w które zainwestowałeś miesiące, dla
jakiejś nowej sztuczki z przestawianiem słów,...
```

Formatowanie z wyrównaniem (JUST) i bez wyrównania marginesów

Aby sformatować akapit używa się funkcji CTRL-F. Formatowaniu ulega część akapitu leżąca na prawo od kursora aż do końca akapitu. Szerokość kolumny i jej miejsce na ekranie (co wpływa na późniejsze położenie tekstu na papierze) zależy od ustawienia marginesów. Przy formatowaniu akapitu w opisywanej wersji giną ewentualne dodatkowe wysuwy dodane przez użytkownika w pierwszym lub ostatnim wierszu akapitu. Dyktuje to kolejność działań, gdy chcemy mieć odpowiednie światło między kolejnymi akapitami.

Formatowanie w połączeniu z operacją ALT-*n* ma zastosowanie do zmiany wysuwu w obrębie całego akapitu (por. — operowanie blokami tekstu).

Stronicowanie

Chiwriter automatycznie dzieli tekst na strony. Liczba rządków na stronie jest parametryzowalna i może, na przykład, wynosić 130 (większej liczby nie zaleca się ze względu na format typowego papieru — A4). Granice stron są zaznaczane na ekranie liniami kropkowanymi. Zakończenie strony można wymusić za pomocą operacji CTRL-P w wierszu, w którym występuje kursor. *Chiwriter* oznacza to na ekranie linią ciągłą. W razie zmiany decyzji, powtórne użycie CTRL-P przywraca stan poprzedni. Doraźnie, na przykład, aby uniknąć pozostawienia u dołu strony samego tytułu lub na początku nowej strony nieestetycznie się prezentującego ostatniego, niepełnego wiersza akapitu, można przerzucić linię automatycznej paginacji o wiersz wyżej za pomocą operacji CTRL-N. Można z tego się wycofać naciskając powtórnie CTRL-N.

Operowanie blokami tekstu

Operacja ALT-M powoduje, iż dalsze manipulacje kursorem umożliwia zaznaczenie fragmentu tekstu jako pewnej całości, na której można będzie wykonywać dalsze całościowe przekształcenia. Zaznaczanie to objawia się w formie ukazywania tekstu w negatywie i zwie się *oświetlaniem*. W razie niebacznego wykonania funkcji ALT-M, klawisz ESC odwołuje jej skutki.

Znakom oświetlonej porcji tekstu można automatycznie zmienić krój, wciskając odpowiedni klawisz F*n*. W oświetlonym bloku można całościowo zmienić wielkość odstępu między wierszami — ALT-*n*. W szczególności, można w ten sposób zmienić wysuw w kilku akapitach na raz (całym tekście!), jednak z utratą dodatkowych przestrzeni wprowadzonych za pomocą operacji CTRL-A.

Tekst oświetlony można usunąć (dotyczy to oczywiście również części wydzielonego tekstu niewidocznych na ekranie) za pomocą operacji ALT-C lub przesłać do bufora ALT-D. Również przy usuwaniu tekst trafia do bufora — choćby na „wszelki wypadek". Replikę tekstu przechowywanego w buforze można otrzymać w dowolnym miejscu (decyduje pozycja kursora) za pomocą operacji ALT-P.

Przeszukiwanie i zastępowanie

Po wykonaniu funkcji ALT-S, *Chiwriter* prosi o wprowadzenie tekstu-wzorca. Wciśnięcie klawisza ESC rozpoczyna w tej sytuacji przeszukiwanie pliku w celu odnalezienia zadanego wzorca. Kolejne operacje ALT-S powodują odnajdywanie następnych wystąpień wzorca w pliku. Poszukiwanie odbywa się tylko w części pliku leżącej na prawo (w dół) od kursora. Ustalony wzorzec usuwa się za pomocą operacji ALT-F. Operacja ALT-R umożliwia dodatkowo automatyczne (wybiórcze lub globalne) zastępowanie odnalezionego wzorca przez nowy, uprzednio określony.

Przykład:

Następujący ciąg operacji:

CTRL-HOME ALT-R *CHIWRITER* ESC Chi? OK! ESC a

spowoduje zastąpienie w całym tekście (opcja a, ang. all) napisu *CHIWRITER* przez napis „Chi? OK!".

Tworzenie makrodefinicji

Istnieje możliwość ujmowania pod wspólną nazwą dowolnych ciągów uderzeń w klawisze. Definiowanie rozpoczyna operacja CTRL-D, po której podaje się nazwę makrodefinicji (zakończoną wciśnięciem klawisza ENTER). Dalej — uważnie — należy wcisnąć definiowany ciąg klawiszy i zakończyć go powtórną operacją CTRL-D. Stan definiowania jest sygnalizowany na ekranie napisem DEF. Wywołanie zdefiniowanej sekwencji odbywa się za pomocą operacji CTRL-K, po której należy podać nazwę makrowywołania zakończoną znakiem CR (klawisz ENTER). W razie nieuważnego spowodowania odpowiednich operacji, stan pierwotny przywraca się wciśnięciem klawisza ESC.

Przykład:

W celu określenia nazwą „c" ciągu *CHIWRITER*, należy przycisnąć po kolei następujące klawisze:

CTRL-D c ENTER F3 F3 CAPS-LOCK chiwriter CAPS-LOCK F1 F1 CTRL-D

Późniejsze odwołanie się do sekwencji

CTRL-K c ENTER

spowoduje zamierzone skutki. Zwróćmy uwagę na dbałość o ustalenie typowego kontekstu w końcowej części treści makrodefinicji — krój F1, powrót do rejestru małych liter (zakładamy milcząco, że definiowanie w takich warunkach się rozpoczęło).

Za pomocą makrodefinicji można określić złożone z kilku elementów symbole matematyczne, na przykład, wysokie nawiasy, symbol pierwiastka itp. Swoje własne makrodefinicje — a także ukształtowane wedle własnej wygody inne cechy programu — użytkownik może przechować za pomocą polecenia Save Parameters w prywatnym pliku. Aby plik taki zastąpił standardowy plik parametrów *Chiwritera*, należy na początku sesji podać w poleceniu **CW** jego nazwę poprzedzoną znakiem minus, na przykład CW —ZPLPARAM.

Inne operacje

Operacja CTRL-C powoduje ześrodkowanie zawartości wiersza, w którym znajduje się kursor. Operacja ALT-L usuwa wiersz oznaczony kursorem. Dzielenie wiersza najprościej wykonać wciskając klawisz ENTER (lecz uwaga, kończy się tym samym akapit — por. formatowanie). Łączenie wierszy uzyskuje się przez ustawienie kursora na początku drugiego wiersza i użycie klawisza DEL.

Wskaźnik zapełnienia pamięci

Szereg informacji kontrolnych jest wyświetlanych stale podczas redagowania w tzw. wierszu statusowym (u góry ekranu). Niektóre z nich omówiono w poprzednich punktach, warto jednak zwrócić uwagę na jeszcze jedną — wskaźnik zapełnienia pamięci przez wpisany tekst. Wyraża się go w procentach w rubryce **FULL** i nie jest wskazane przekraczanie go powyżej 90% (a jest to już pokaźna liczba stron tekstu, rzędu 40—50). Manipulacja długimi plikami może być niezbyt wygodna ze względu na długi czas składowania, toteż lepiej wykorzystać zdolność *Chiwritera* do łączenia krótszych plików w jeden przy drukowaniu.

Zmianę wartości wskaźnika **FULL** można najłatwiej zaobserwować w eksperymencie polegającym na przechowaniu w buforze całego tekstu (za pomocą operacji ALT-D).

OBSERWACJE UŻYTKOWNIKA

Autor tego artykułu posługuje się *Chiwriterem* od stycznia 1987 roku, „przelewając" za jego pomocą na dysk kilkaset tysięcy bajtów tekstu. W tym czasie nie doszło praktycznie do żadnych załamań w działaniu programu. Miał jednak miejsce precedens, który doprowadził do niespodziewanej blokady programu. I nie należy tego mylić z nieokreślonymi zachowaniami spowodowanymi wadami sprzętu (syndrom kompatybilności). Tym ważniejsze jest, aby od początku wejść w nawyk wykonywania okresowych składowań przy dłużej pracy. Można to osiągnąć przez zmianę w parametrach systemu odstępu czasu między automatycznymi składowaniami (polecenia **C, P** i dalej **M 15** — składowanie co kwadrans). Warto też dostroić program do naszych przyzwyczajeń, zmieniając odpowiednio zawartość innych wpisów w pliku parametrów. Można w ten sposób określić — na przykład — preferowany rozmiar strony, wysuw papieru, sposób numeracji stron itp. Jeśli w poleceniu **CW** nie określono inaczej, *CHIWRITER* czyta parametry z pliku **DEFAULT.PAR.**

Przed drukowaniem nie zawadzi zadbać o wielkość lewego marginesu na drukarce (polecenia **P, O, L** *liczba dodatkowych odstępów*). Warto zawsze sprawdzić przesunięcie papieru na początek strony i ustawienie prowadnic papieru względem położenia spoczynkowego głowicy.

Dobrze jest też przed przystąpieniem do pisania ustalić roboczy skorowidz, w którym będą składowane pliki (polecenia **C, D**), aby ich nie zapisywać do systemowego skorowidza *Chiwritera*, co może mieć znaczenie przy liczniejszym gronie użytkowników danej instalacji.

Do cech uciążliwych można zaliczyć trudności w odnajdowaniu określonych miejsc w dłuższym pliku. Operacje PGUP, PGDN działają zbyt wolno, podobnie mają się sprawy z funkcją wyszukiwania CTRL-S. W wersji drugiej dołączono polecenie odnajdywania strony według jej numeru.

CHIWRITER NA TLE OGÓLNEJ CHARAKTERYSTYKI PROCESORÓW TEKSTU

Odwołam się teraz powtórnie do oceny oprogramowania przeznaczonego do przetwarzania tekstów wyrażonej przez autorów *Whole Earth Software Catalog*. Według propozycji Charlesa Spezzano można wydzielić trzy sfery zastosowań procesorów tekstu:

- waga lekka — doraźna korespondencja, notatki,
- waga średnia — częste pisanie listów, raportów lub artykułów, bez wymagań na zaawansowane techniki w rodzaju automatycznych nagłówków, podziału ekranu; niezbyt wielka długość dokumentów (ponad 25 stron),
- waga ciężka — w pełni uzupełniające się, zaawansowane konstrukcje oprogramowania, za pomocą których można z powodzeniem przygotować trudne artykuły, skomplikowane (o zmiennych formatach) raporty wszelkiego rodzaju — z książkami włącznie.

Według tej gradacji ze spokojem można zaliczyć *Chiwritera* do wagi półciężkiej. Dokładniej można to określić według wieloaspektowej charakterystyki, którą przytaczam za cytowanym już Stewartem Brandem. Przy kryteriach dwuwartościowych występowanie bądź brak cechy w *Chiwriterze* zaznaczają odpowiednie znaki „+" i „—":

- dostępność na komputerach zgodnych z IBM PC/XT/AT,
- swoboda sporządzania kopii (+),
- zapotrzebowanie na pamięć (wartość minimalna) — 256 KB,
- maksymalna wielkość dostępnej pamięci roboczej — brak danych handlowych (opisana wersja współpracuje z pamięcią 640 KB),
- liczba wierszy redagowanego tekstu na ekranie — jest to współczynnik krytyczny dla wielu programów; tzw. „tunelowe widzenie" jest głównym ograniczeniem komputerowego pisania, w *Chiwriterze* wskaźnik ten wynosi (dla minimalnego wysuwu) 33 wiersze,
- maksymalny rozmiar pliku — około 40—45 stron o wymiarach 38 wierszy po 64 znaki (około 200—300 słów na stronie),
- dostępność korektora ortograficznego (—); tylko na zasadzie współpracy z odrębnym (za osobne pieniądze) programem,
- łatwość telekomunikacji (—); dotyczy w równym stopniu *Chiwritera*, co większości zainstalowanych u nas komputerów; dodajmy, że cały problem uzdatniania procesorów tekstu do celów telekomunikacji jest — zdaniem światowych specjalistów — „wstydliwym aspektem przetwarzania tekstów",
- możliwość przygotowywania tekstów programów (+); wielu użytkowników procesorów tekstu nie stroni od używania ich (jeśli tylko na to pozwalają) do przygotowywania tekstów programów, *Chiwriter* pozwala na takie zastosowanie (polecenia **L, A** i **S, A**),
- możliwość wyboru i zmiany skorowidza na dysku (+),
- dostępność polecenia wycofywania pochopnej decyzji skasowania tekstu (ang. undo) — *Chiwriter* umożliwia automatyczne okresowe składowanie, które w pewnym stopniu zaspokaja ten wymóg,
- automatyczne formatowanie (+),
- numerowanie stron na ekranie i wizualne zaznaczanie na ekranie końców stron (+); ponadto, są podawane na ekranie współrzędne bieżącego położenia kursora (numer rządka — **ROW**, numer kolumny — **COL**),
- podział ekranu umożliwiający oglądanie i redagowanie dwóch (kilku) plików na raz (—); niemniej, *Chiwriter* umożliwia łączenie plików (przez wykorzystanie bufora),
- możliwość drukowania wprost z pamięci operacyjnej, bez uprzedniego składowania (+),
- automatyczne składowanie (+); rzecz wielce istotna, zwłaszcza dla roztargnionych, a o to nie trudno przy pisaniu (por. *casus* Alfreda Lee),
- możliwość makrodefiniowania (+),
- użycie myszy (—); dodajmy za autorem stosowanej tu kwalifikacji, że mysz — acz użyteczna przy częstych skokach po ekranie — zabiera piszącemu połowę palców,

- maksymalna szerokość strony w znakach — 77,
- wyrównanie liczb w kolumnach; możliwości kalkulatorowe (—),
- możliwość organizacji notek, nagłówków itp. (+),
- operacje na szpaltach (—),
- możliwość redagowania w czasie drukowania (—); jest to, przy niezbyt dużej szybkości działania typowych drukarek mozaikowych, druga istotna niedogodność; przykładowo — tekst elaboratu na 32 strony maszynopisu jest wyprowadzany (w trybie szkicowym) w czasie sięgającym trzech kwadransów,
- drukowanie proporcjonalne, czyli kształtowanie liter na podobieństwo składu drukarskiego — „i" jest węższe niż „m" itp. (—).

CZY CHIWRITER JEST DOBRYM PROGRAMEM?

Aby uniknąć subiektywizmu, o który w przypadku procesorów tekstu bardzo łatwo, sięgnijmy znów do metod oceny sugerowanych przez wydawców WESC. Czytelników, którzy mają cień wątpliwości co do prywatnej opinii o *Chiwriterze* niżej podpisanego, zapewniam, że program ten posłuży mi jeszcze przez długi czas, nim dam się skusić na coś innego; dla wielu tekstów jest to narzędzie nader zgrabne, a używając frazeologii taksatorów zza oceanu można rzec, że: „*Chiwriter* nosi się znakomicie".

Można zatem, zdaniem fachowców z *WESC*, rozważać następujące kryteria doskonałości oprogramowania:

- Dobre oprogramowanie wykonuje ważne zadania dobrze. Slogan ten znajduje pełne odzwierciedlenie przy zastosowaniu *Chiwritera* do tekstów takich jak niniejszy, tzn. zawierających znaki z typowej klawiatury komputera, wzbogacone o kilka krojów liter, a nawet w odniesieniu do tekstów matematycznych o średniej złożoności.
- Dobre oprogramowanie jest „przezroczyste", to znaczy nakład pracy niezbędny do nauczenia się wygodnego posługiwania się tym oprogramowaniem jest niewielki w porównaniu do stwarzanych przez nie możliwości. W wypadku *Chiwritera* jest tak niewątpliwie. Firmowy samouczek zajmuje zaledwie 10 stron maszynopisu. Dla porównania podręcznik *Wordstara* stanowi już pokaźny skrypt, a dla takiego programu jak TEX jest już całą książką.
- Dobre oprogramowanie ma strukturę cebuli. To kryterium trudno odnieść wprost do oprogramowania takiego jak *Chiwriter*. Zważmy jednak, że wykazy jego poleceń (listy *menu*) są rozplanowane i oznaczone w sposób bardzo przejrzysty i właśnie — warstwowy.
- Dobre oprogramowanie daje się łatwo mieszać z innym oprogramowaniem. To kryterium dotyczy standaryzacji plików i funkcji. Przewodzą tu pakiety zintegrowane w rodzaju *Symphony*. Możliwość dowolnego definiowania krojów pisma sprawia, że pliki sporządzone w jednej instalacji mogą okazać się nie w pełni czytelne w innej instalacji. Można się przy tym bronić przesyłając adresatowi, prócz tekstu, własną konfigurację *Chiwritera*. Lepsza byłaby jednak standaryzacja, o którą warto zabiegać, lecz którą trudno wymusić.
- Dobre oprogramowanie ma dobre wsparcie i reklamę. To kryterium dotyczy w większym stopniu realiów rynków zagranicznych. Chodzi o to, że do programu szeroko rozpowszechnionego, działającego na wielu komputerach i z serwisem wzywanym telefonicznie można mieć większe zaufanie. Rzecz jednak w tym czy zawsze szeroką reklamę itp. zyskuje właśnie to „najlepsze" oprogramowanie? W odniesieniu do *Chiwritera* warunek ten jest spełniany.
- Dobre oprogramowanie nie jest chronione przed kopiowaniem (!). Podkreślam, że nie jest to mój wymysł, lecz specjalistów z *WESC*. Powód takiego stanowiska jest dwojaki. Po pierwsze, ochrona przed kopiowaniem utrudnia użytkowanie oprogramowania w pewnych sytuacjach. Celne jest porównanie: „oprogramowanie chronione podobne jest do tego, jakby książkę obłożyć prawem czytania jej wyłącznie w mgnieniach flesza" — Richard Dalton, *WESC*. Naruszenie prawa powstaje przecież tylko w sytuacji sprzedaży nie swojego oprogramowania. Po drugie, warto zawsze się zwrócić po oprogramowanie do producenta, zyskując dodatkowe atuty — dobrą i ładną dokumentację (czy u nas również?), pomoc i konsultacje w użytkowaniu. *Chiwriter* w podstawowej konfiguracji jest dostępny bez ochrony.
- Ostatnie kryterium — dobre oprogramowanie jest dość drogie (?). Kryterium to autorzy *WESC* uważają za kontrowersyjne. Podkreślają, dla przykładu, że wśród procesorów tekstu ceny kształtują się od 600 dolarów (większość) do 10 dolarów (jeden z najlepszych w puli skatalogowanych w *WESC — PC-WRITE*).

Chiwritera nie było w stawce procesorów tekstu ocenianych w *WESC 86*. Nie jest to zbyt dziwne, zważywszy że liczba programów tego rodzaju osiąga już w świecie setki. Tym bardziej godne uwagi wydaje się spostrzeżenie R. Daltona, który pisze: „Dziś bardzo łatwo sięgać po coraz to nowe i nowe „generacje" oprogramowania. Czyniąc tak, łatwo popaść w stan ciągłego przygotowywania się do pracy — miast pracy samej." *Chiwriter*, procesor tekstu z pewnością pozwala trochę popracować.

Podziękowania

Podjęcie tematu stało się możliwe dla autora niniejszego opracowania dzięki uprzejmości następujących osób: prof. dra hab. Stefana Paszkowskiego z Instytutu Niskich Temperatur i Badań Strukturalnych PAN we Wrocławiu, orędownika komputerowych technologii opracowywania tekstów; doktora Zbigniewa Leyka z Instytutu Informatyki Uniwersytetu Warszawskiego, autora polskiego liternictwa i dodatkowych udogodnień w opisanej wersji programu oraz mgra Zdzisława Jarzębowskiego z Instytutu Informatyki Uniwersytetu Wrocławskiego, obdarzonego intuicją i zaradnością w odnajdywaniu ciekawej literatury.

Stan i tendencje rozwoju technologii programowania w Kombinacie Robotron

dokończenie ze s. 5

5. Inne produkty

PSU — środowisko wspomagające projektowanie i programowanie (zgodny z Unixem system narzędziowy Jednolitego Systemu).

UNI-3 — system zarządzania dla bibliotek i komponentów bibliotecznych Jednolitego Systemu.

DSB — biblioteka struktur danych Jednolitego Systemu.

PLET — prekompilator tablic decyzyjnych dla języka PL/I Jednolitego Systemu.

MOSY 1600 — modułowy system do wspomagania wytwarzania oprogramowania dla mikrokomputerów (system operacyjny MOOS 1600).

STRUCTUR-MAC — programowanie strukturalne w językach MACRO 1600 i MACRO-SM (system operacyjny MOOS 1600).

* * *

Dalszy rozwój technologii programowania Robotronu będzie odbywał się z uwzględnieniem aspektów takich, jak:
— włączenie języków wyższego poziomu w istniejący zestaw narzędzi (Fortran 77, MODULA-2), przy jednoczesnym zapewnieniu akceptowania jednakowych kompilatorów dla komputerów różnych klas,
— stworzenie jednolitego środowiska do rozwoju oprogramowania (Unix) dla komputerów różnych klas (Jednolity System, SKR, PC),
— rozszerzenie środków wspomagania prac rozwojowych w fazach specyfikowania i projektowania,
— uwzględnienie narzędzi zapewniających dobrą jakość i wspomagających kierowanie,
— podwyższenie stopnia przenośności wszystkich narzędzi programowych i ciągłe dopasowywanie do nowych systemów operacyjnych (np. MS-DOS i PC-DOS),
— tworzenie możliwości pracy sieciowej produktów programowych,
— opracowywanie norm państwowych w celu ujednolicenia nazw i określeń oraz sprzężeń.

Tłumaczyła: MAŁGORZATA RÓŻAŃSKA

ANNA SYFERT
ANDRZEJ RAZNOWIECKI
(oprac.)
Warszawa

Struktura systemu operacyjnego PC-DOS (4)

W tej części opisano wywołania, które nie są zgodne z systemem operacyjnym CP/M, ponieważ w istocie wykonują funkcje specyficzne dla PC-DOS. Zostają wykonane przez instrukcję dalekiego wywołania CALL do adresu 050H w bloku prefiksu segmentu programu lub przez przerwanie programowe INT 21H.

W obydwu wypadkach, kod funkcji zostaje przekazany w rejestrze AH. Funkcje te są charakterystyczne dla drugiej wersji PC-DOS i nie wiążą się ani z CP/M ani z Xenixem.

FUNKCJE DOTYCZĄCE PLIKÓW

Poniższa grupa funkcji występuje w innej formie również w systemie operacyjnym CP/M. Umożliwiają one obsługę plików o dostępie swobodnym.

Funkcja 27H — odczyt bloków danych o dostępie swobodnym.
Funkcja ta odpowiada w zasadzie funkcji 21H, jednak w odróżnieniu od niej odczytuje kilka bloków danych równocześnie i zapamiętuje je w buforze dysku (DTA). Liczba odczytywanych rekordów danych daje się obliczyć z wielkości bloku danych (wyrównanie 0EH i 0FH). Wtedy rekordy danych są odczytywane począwszy od numeru wybranego rekordu znajdującego się w komórkach o wyrównaniu 21H—24H (numer ten musi zostać właściwie ustawiony przed wykonaniem funkcji). Po dokonaniu odczytu uaktualniane są odpowiednio do wykonania funkcji komórki 0CH i 0DH (aktualny blok danych), 20H (aktualny rekord danych przy dostępie sekwencyjnym) oraz 21H—24H (aktualny rekord danych przy dostępie swobodnym).

Parametry wejściowe:
AH — 27H,
CX — liczba odczytywanych bloków danych (dla CX=0 odczyt nie zostanie przeprowadzony; rozkaz NOP),
DS:DX — adres otwartego bloku FCB.

Parametry wyjściowe:
AL-00H, jeśli proces odczytu został zakończony bez błędu,
AL-01H, jeśli osiągnięto koniec pliku (EOF) i nie zostały odczytane żadne dane,
AL-02H, jeśli proces odczytu nie został przeprowadzony, ponieważ w buforze (DTA) nie ma dostatecznie dużego obszaru wolnej pamięci dla rekordu danych,
AL-03H, jeśli podczas odczytu rekordu został osiągnięty koniec pliku, rekord danych został uzupełniony do pełnej oczekiwanej długości przez dołączenie 00H (koniec rzeczywiście odczytanych danych jest oznaczany przez CTRL-Z lub musi zostać obliczony z wielkości pliku (wyrównanie od 10H do 13H w FCB),
CX — liczba rzeczywiście odczytanych bloków danych.

Funkcja 28H — zapis bloków danych o dostępie swobodnym.
Tak jak w wypadku odczytu bloku danych o dostępie swobodnym, należy najpierw ustawić w komórkach o wyrównaniu od 21H do 24H bloku FCB względny numer pierwszego zapisanego rekordu danych. Liczba zapisywanych rekordów danych zostaje obliczona z wielkości bloku danych (wyrównanie 0EH i 0FH) oraz zapisana począwszy od rekordu danych wyspecyfikowanego w komórkach o wyrównaniu 21H do 24H. Zapisywane dane znajdują się w buforze dysku. Po dokonaniu zapisu zostają uaktualnione, odpowiednio do wykonania funkcji, komórki o wyrównaniu 0CH—0DH (aktualny blok danych), 20H (aktualny rekord danych przy dostępie sekwencyjnym) oraz 21H—24H (aktualny rekord danych przy dostępie swobodnym). Podobnie jak w wypadku dostępu sekwencyjnego można dokonać zapisu do pliku z ochroną zapisu!

Parametry wejściowe:
AH — 28H,
CX — liczba zapisanych bloków danych (0 nie powoduje zapisu),
DS:DX — adres otwartego bloku FCB.

Parametry wyjściowe:
AL-00H, jeśli proces zapisu został zakończony bez błędu,
AL-01H, jeśli nośnik danych jest całkowicie zapełniony (proces zapisu nie został przeprowadzony),
AL-02H, jeśli proces zapisu nie został przeprowadzony, ponieważ w buforze (DTA) nie ma dostatecznie dużo wolnego miejsca do zapisu bloku danych,
CX — liczba rzeczywiście zapisanych bloków danych.

Funkcja 29H — oddzielenie nazwy pliku w linii polecenia.
Funkcja przeszukuje ciąg znaków linii polecenia, w celu wydzielenia ważnej specyfikacji pliku (nie uwzględniając ścieżki katalogowej). Nazwa pliku może zawierać znaki „*" lub „?". Funkcja przeszukiwania jest przerywana, jeśli zostanie znaleziony znak CTRL-Z lub separator pliku (w zależności od sposobu pracy funkcji przeszukiwania określonego w AL).

Parametry wejściowe:
AH — 29H,
AL — sposób przeszukiwania,
DS:SI — przeszukiwany ciąg znaków linii polecenia,
ES:DI — adres założenia nieotwartego bloku FCB (w obszarze tym może znajdować się nieotwarty FCB, zawierający informacje nie przekazywane w linii polecenia).

Parametry wyjściowe:
AL-00H, jeśli nazwa pliku w linii polecenia nie zawiera znaków „*" i „?",
AL-01H, jeśli nazwa pliku w linii polecenia zawiera te znaki,
AL-FFH, jeśli zawarta w linii polecenia specyfikacja dysku jest niewłaściwa,
DS:SI — adres nie przeszukiwanej części linii polecenia,
ES:DI — adres wygenerowanego nieotwartego bloku FCB (jeśli nie znaleziono dowolnej nazwy pliku, to ES:DI+1 zawiera spacje, 20H).

Funkcja 2FH — określenie adresu bufora dysku.
Funkcja ta jest przeciwieństwem funkcji 1AH. Ponieważ w wypadku funkcji zgodnych z systemem Xenix użycie bufora dysku nie jest widoczne dla użytkownika, funkcję tę zaliczono do charakterystycznych dla PC-DOS.

Parametr wejściowy:
AH — 2FH.

Parametr wyjściowy:
ES:BX — adres użytego bufora dysku (DTA).

POZOSTAŁE FUNKCJE

Do tej grupy zalicza się funkcje dotyczące ustawiania przerwań, odczytywania czasu oraz informacji specyficznych dla danego kraju.

Funkcja 25H — ustawienie wektora przerwań.
Wektory przerwań nigdy nie powinny być zmieniane bezpośrednio w obszarze pamięci od 0000H:00000H do 0000:0400H, ponieważ w tym wypadku PC-DOS nie miałby nad nim żadnej bezpośredniej kontroli, co byłoby w szczególności niekorzystne dla przyszłej wielozadaniowej wersji tego systemu operacyjnego.

Parametry wejściowe:
AH — 25H,
AL — numer przerwania podlegającego zmianie (od 00 do FFH),
DS:DX — wektor przerwania (adres początkowy procedury obsługi tego przerwania).

Funkcja 2AH — odczytanie daty z zegara systemowego.

Parametr wejściowy:
AH — 2AH

Parametry wyjściowe:
AL — dzień tygodnia (0 — niedziela, 6 — sobota)
DL — dzień (1, ..., 31)
DH — miesiąc (1, ..., 12)
CX — rok (1980, ..., 2099)

Funkcja 2BH — zmiana daty.

Parametry wejściowe:
AH — 2BH
DL — dzień (1, ..., 31)
DH — miesiąc (1, ..., 12)
CX — rok (1980, ..., 2099)

Parametry wyjściowe:
AL — 00H, jeśli przekazana data była prawidłowa,
AL — FFH, jeśli przekazana data była nieprawidłowa.

Funkcja 2CH — odczytanie czasu systemowego.

Parametr wejściowy:
AH — 2CH

Parametry wyjściowe:
CH — godziny (0, ..., 23)
CL — minuty (0, ..., 59)
DH — sekundy (0, ..., 59)
DL — setne sekundy (0, ..., 99)

Funkcja 2DH — ustawienie czasu systemowego.

Parametry wejściowe:
AH — 2DH
CH — godziny (0, ..., 23)
CL — minuty (0, ..., 59)
DH — sekundy (0, ..., 59)
DL — setne sekundy (0, ..., 99)

Parametry wyjściowe:
AL — 00H, jeśli przekazany czas był prawidłowy,
AL — FFH, jeśli przekazany czas był nieprawidłowy.

Funkcja 2EH — zmiana wskaźnika weryfikacji dysku.

W PC-DOS istnieje możliwość odczytania informacji z nośnika informacji (dysku) po ich zapisaniu, w celu upewnienai się, że proces zapisu został zakończony bez wystąpienia błędów. Aczkolwiek to kontrolne czytanie wydłuża czas dostępu do nośnika informacji, jednak w ogólności zalecane jest używanie tej funkcji, ponieważ z reguły błędy dysku dotyczą najcenniejszych danych.

Parametry wejściowe:
AH — 2EH
AL — 00H w celu wyłączenia funkcji kontrolnej,
AL — 01H w celu włączenia funkcji kontrolnej.

Funkcja 30H — ustalanie numeru wersji systemu operacyjnego.

Parametr wejściowy:
AH — 30H

Parametry wyjściowe:
AL — główny numer wersji, np. 2 dla PC-DOS 2
AH — poboczny numer wersji, np. 0BH dla MS-DOS 2.11
BH — numer fabryczny dla wersji OEM (ang. original equipment manufacturer)
BL.CX — numer fabryczny kolejny

Funkcja 33H — zmiana reakcji na CTRL-C.

W PC-DOS istnieje możliwość przerwania wykonywania funkcji przy użyciu klawisza CTRL-C. Standardowo CTRL-C funkcjonuje jedynie dla funkcji we—wy dla urządzeń standarowych (funkcje 01H, ..., 0CH). Przez powyższe wywołanie systemowe reakcja na CTRL-C może zostać rozszerzona na wszystkie funkcje, dla których przerwanie nie jest explicite wykluczone. W wypadku, gdy odczytywana jest reprezentacja CTRL-C w kodzie ASCII (03H) przy użyciu funkcji 06H i 07H, to sprawdzanie za pomocą powyższej funkcji musi zostać uprzednio wyłączone!

Parametry wejściowe:
AH — 33H
AL — 00H oznacza przekazanie aktualnego stanu rozpoznawania CTRL-C
AL — 01H oznacza zmianę rozpoznawania CTRL-C
DL — 00H oznacza wyłączenie kontroli CTRL-C (AL — 01H)
DL — 01H oznacza włączenie kontroli CTRL-C (AL — 01H)

Parametry wyjściowe:
AL — FFH, jeśli AL zawiera w momencie wywołania funkcji nieprawidłowy kod
DL — 00H, jeśli kontrola CTRL-C została wyłączona
DL — 01H, jeśli kontrola CTRL-C została włączona

Funkcja 35H — odczytanie wektora przerwań.

Parametry wejściowe:
AH — 35H
AL — numer odczytywanego wektora przerwań (00H-FFH)

Parametr wyjściowy:
ES:BX — wektor przerwań (adres początkowy procedury obsługi tego przerwania)

Funkcja 36H — odczytanie informacji o organizacji dysku. Powyższa funkcja dostarcza informacji o organizacji nośnika danych (liczba sektorów na niepodzielną jednostkę alokacji, wielkość sektora, całkowita i rozporządzalna pojemność dysku).

Parametry wejściowe:
AH — 36H
DL — numer dysku (0 — bieżący dysk, 1 — dysk A itd.)

Parametry wyjściowe:
AX — liczba sektorów na niepodzielną jednostkę alokacji (najmniejsza jednostka, na jakie PC-DOS dzieli obszar pamięci dyskowej; np. 1 lub 2 dla dyskietek, 8 dla dysków 10 MB),

Budowa obszaru z informacjami charakterystycznymi dla danego kraju przy wywołaniu funkcji 38H

Wyrównanie	Znaczenie
00H—01H	Format daty i czasu: 0=USA (hh:mm:ss miesiąc/dzień/rok) 1=Europa (hh:mm:ss dzień/miesiąc/rok)
02H—06H	Skrót oznaczenia waluty kraju (zakończony przez 00H) np. DM dla RFN
07H—08H	Separatory do przejrzystego dzielenia liczb w bloki po tysiąc (np. kropka dla RFN, 1.000.000); ciąg znaków zakończony przez 00H
09H—0AH	Separator między pozycjami przed i po przecinku dziesiętnym (np. kropka dla USA, 10.25); ciąg znaków jest zakończony przez 00H
0BH—0CH 0DH—0EH	Separator między częściami reprezentacji daty Separator między poszczególnymi częściami reprezentacji czasu
0FH—0FH	Najbardziej znaczące 6 bitów nie jest wykorzystywane. Bit 1: 0 oznacza brak spacji między wartością i oznaczeniem waluty, a 1 oznacza spację między wartością i oznaczeniem waluty. Bit 0: oznaczenie waluty stoi przed jej wartością (np. $100) lub za wartością (np. 40,75DM)
10H—11H	Liczba miejsc po przecinku zwykle podawanych dla należności w danej walucie (np. RFN 2 miejsca, tzn. 32,40DM)
12H—12H	Reprezentacje czasu: 00H — format 12-godzinny (np. 3:45 pm) 01H — format 24-godzinny (np. 15.45)
13H—15H	Adres wyrównanie i segment procedury, która dla znaków specjalnych charakterystycznych dla danego kraju (kod ASCII: 80H ... FFH) przedstawionych jako małe dostarcza kody ASCII odpowiednich znaków przedstawionych jako duże; znaki są przekazywane w AL
16H—17H	Separator oddzielający elementy list; ten ciąg znaków jest zakończony przez 00H
18H—1EH	Zarezerwowane

AX — FFFFH, jeśli numer dysku w DL jest nieprawidłowy,
BX — liczba wolnych jednostek alokacji,
CX — liczba bajtów na sektor (w ogólności 512),
DX — całkowita pojemność dysku (w jednostkach alokacji).

Funkcja 38H — odczytanie reprezentacji czasu i daty charakterystycznej dla danego kraju.
Powyższa funkcja stwierdza, jaka reprezentacja została wybrana wcześniej przez użytkownika (przy użyciu tej funkcji lub w pliku CONFIG.SYS). W tabeli przedstawiono budowę obszaru pamięci zawierającego informacje charakterystyczne dla danego kraju.

Parametry wejściowe:
AH — 38H
AL — kod kraju; podanie 00H oznacza, że należy użyć wcześniej określonego kodu kraju;
DS:DX — adres początkowy obszaru o długości 32 bajtów, w którym PC-DOS powinien odłożyć informacje charakterystyczne dla danego kraju.
DS:DX — FFFFH oznacza, że kod kraju przekazany w AL powinien zostać przyjęty przez system operacyjny.

Parametr wyjściowy:
DS:DX — adres początkowy 32-bajtowego obszaru pamięci zawierającego informacje charakterystyczne dla danego kraju.

Funkcja 54H — odczytanie wskaźnika weryfikacji dysku por. z funkcją 2EH.

Parametr wejściowy:
AH — 54H

Parametr wyjściowy:
AL — 00H, jeśli nie jest wykonywane kontrolne odczytanie po zapisie,
AL — 01H, jeśli jest wykonywane kontrolne odczytanie po każdym zapisie.

Funkcja 57H — zmiana lub odczyt czasu i daty ostatniego odwołania do pliku.
Powyższa funkcja umożliwia odczytanie lub zmianę daty i czasu ostatniego odwołania do pliku. W tym celu plik należy wcześniej otworzyć.

Parametry wejściowe:
AH — 57H
AL — 00H oznacza odczytanie czasu i daty ostatniego odwołania
AL — 01H oznacza ustawienie daty i czasu ostatniego odwołania
BX — numer kanału otwartego pliku
CX — ustawiany czas (jeśli AL=01H)
DX — ustawiana data (jeśli AL=01H)

Parametry wyjściowe (bit C nie ustawiony):
CX — czas ostatniego odwołania (jeśli AL=000H)
DX — data ostatniego odwołania (jeśli AL — 00H)

Komunikaty błędów (bit C ustawiony):
AX — 01H, jeśli AL zawiera nieprawidłowy kod rozkazowy (nie 00H lub 01H)
AX — 06H, jeśli plik nie został otwarty.

VI Ogólnopolskie Targi Wynalazków

Wzorem lat ubiegłych Ośrodek Postępu Technicznego w Katowicach organizuje kolejne Targi Wynalazków. Odbędą się one **w dniach 9—13 maja 1988 r. w Katowicach, na terenie Ośrodka Postępu Technicznego.** Patronat nad imprezą objął Minister-Kierownik Urzędu Postępu Naukowo-Technicznego i Wdrożeń, Konrad Tott.

Celem Targów jest upowszechnianie najlepszych projektów wynalazczych z różnych dziedzin techniki. Przedmiotem Targów mogą być wynalazki, wzory użytkowe i projekty racjonalizatorskie zgłoszone przez zakłady pracy, instytucje, organizacje oraz twórców indywidualnych, a także zapotrzebowanie na rozwiązanie tematów (problemów) zgłaszanych przez przemysł.

W ramach imprez towarzyszących Targom przewiduje się spotkanie wynalazców i racjonalizatorów z kierownictwem Urzędu Postępu Naukowo-Technicznego i Wdrożeń oraz Urzędu Patentowego PRL, celem omówienia aktualnych problemów wynalazczości, oraz branżowe spotkania przedstawicieli uczelni i instytutów naukowo-badawczych z przedstawicielami przemysłu. Ponadto przewiduje się udzielanie porad przez dyżurujących rzeczników patentowych i projekcje filmów technicznych.

Wszystkie rozwiązania prezentowane na Targach będą uczestniczyć w konkursie na najlepszy projekt wynalazczy, a oceny dokona jury Konkursu.

Informacje o problemach (tematach) wymagających rozwiązania, zgłoszonych przez przemysł, zostaną umieszczone w katalogu, a informacje o rozwiązaniach prezentowanych na Targach — w numerze specjalnym Biuletynu Projektów Wynalazczych wydanym z okazji Targów.

Rozwiązania prezentowane na Targach, jak również problemy (tematy) do rozwiązania, objęte będą działającym w OPT systemem promocji osiągnięć naukowo-technicznych. Działające w ramach tego zespoły specjalistów dokonają wyboru najwartościowszych rozwiązań do prezentowania na innych wystawach w kraju i za granicą.

Termin nadsyłania zgłoszeń uczestnictwa w Targach minął 10 lutego br.

Szczegółowych informacji udzielają organizatorzy:

Ośrodek Postępu Technicznego
ul. M. Buczka 1 B
40-955 Katowice
skr. poczt. 454
telefon: 59-60-61 (do 7) w. 194, 225
teleks: 0312458 opt pl.

Harmonogram szkoleń ZETO — Łódź

Centrum Szkolenia Informatycznego ZETO — Łódź
90-558 Łódź, ul. Hutora 69, tel. 32-50-70, teleks 885208
informuje o kursach organizacyjnych w pierwszej połowie 1988 r., obejmujących tematy mikrokomputerowe.

Zgłoszenia uczestnictwa powinny być dokonywane na piśmie (lub teleksem), najpóźniej na miesiąc przed rozpoczęciem szkolenia. Należność za kursy należy wpłacać na konto: ZETO Łódź, NBP I OM Łódź, nr 47018-2219, odnotowując ten fakt na karcie zgłoszenia. Na przelewie należy podać nazwę kursu, jego termin oraz nazwisko uczestnika.

Budowa i projektowanie systemów mikroprocesorowych
- System mikroprocesorowy MCS-80
  Część 1. Mikroprocesor i pamięć
  11—15 kwiecień, cena 13 800 zł;
  Część 2. Układy wejścia—wyjścia
  25—27 maj, cena 13 800 zł;
- Projektowanie systemów wspomagane RTDS-8
  25—29 kwiecień, cena 14 200 zł.

Użytkowanie i obsługa mini- i mikrokomputerów
- System operacyjny CP/M 2.2 — podstawy użytkowania
  6—10 czerwiec, cena 14 200 zł;
- System operacyjny CP/M 2.2 — zasady funkcjonowania
  2—6 maj, cena 14 200 zł;
- System operacyjny CP/M 3.0 — podstawy użytkowania
  27 czerwiec — 1 lipiec, cena 14 200 zł;
- System operacyjny MS DOS — podstawy użytkowania
  28 marzec — 1 kwiecień, 9—13 maj, cena 14 800 zł;
- System operacyjny MS DOS — zasady funkcjonowania
  25—29 kwiecień, 6—10 czerwiec, cena 15 800 zł;
- Podstawy programowania mikrokomputerów
  11—15 kwiecień, 6—10 czerwiec, cena 14 800 zł;
- Basic z podstawami programowania
  5—15 kwiecień, cena 27 600 zł;
- Basic — mikrokomputery ośmiobitowe
  13—17 czerwiec, cena 14 200 zł;
- Basic IBM PC/XT
  18—22 kwiecień, 20—24 czerwiec, cena 14 800 zł;
- Fortran 80
  20—24 czerwiec, cena 14 200 zł;
- Pascal MT+
  18—29 kwiecień, cena 27 600 zł;
- Pascal TURBO — kurs podstawowy
  21 marzec — 1 kwiecień, 2—13 maj, 27 czerwiec — 8 lipiec, cena 29 600 zł;
- Pascal TURBO — rozszerzenia (dla zaawansowanych)
  23—27 maj, cena 15 800 zł;
- Język C
  5—15 kwiecień, cena 29 600 zł;
- Assembler Z-80
  16—27 maj, cena 27 600 zł;
- Assembler 8080
  9—20 maj, cena 27 600 zł;

JAN BIELECKI
Instyt Informatyki
Politechnika Warszawska

Język C

Wskazania, adresy, konwersje

Powszechnie wiadomo, co to jest adres. Osoby programujące w języku asemblerowym zgodzą się zapewne, że umiejętne posługiwanie się adresami rozkazów i danych stanowi sedno ich działalności.

W językach wysokiego poziomu rolę adresu spełnia wskazanie (ang. pointer). Co prawda, w niektórych prymitywnych językach różnica między adresem i wskazaniem nie jest istotna, jednak w języku C jest ona tak znacząca, że może stanowić sprawdzian znajomości języka.

Ponieważ pojęcie adresu jest obce językowi C, zdefiniowanie go w kategoriach tego języka jest trudne i może zostać sprowadzone do stwierdzenia, że wskazanie jest daną lokalizującą pewien **obiekt** (taki jak np. zmienna albo funkcja), natomiast adres jest daną lokalizującą jedynie **miejsce** w pamięci operacyjnej.

Poprzestając na tym co już wyrażono, można przystąpić do rozpatrzenia przykładów, które zapewne lepiej wyjaśnią sedno sprawy niż dyskusyjne definicje.

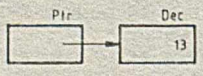
```
#include <stdio.h>

int Dec = 13,
    *Ptr = &Dec;

main()
{
  printf("%d", *Ptr);
}
```

Wydruk 1. Wskazania proste

Rys. 1



Na wydruku 1 przedstawiono najprostsze użycie wskazania. Jak wynika z rys. 1, zmiennej Ptr przypisano daną wskazującą zmienną Dec, a zmiennej Dec przypisano daną typu (int). Wykonanie programu powoduje wyprowadzenie liczby 13.

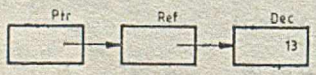
```
#include <stdio.h>

int Dec = 13,
    *Ref = &Dec,
    **Ptr = &Ref;

main()
{
  printf("%d", **Ptr);
}
```

Wydruk 2. Wskazania pośrednie

Rys. 2



Na wydruku 2 przedstawiono istotę wskazania pośredniego. Jak wynika z rys. 2, zmiennej Ptr przypisano wskazanie zmiennej Ref, zmiennej Ref przypisano wskaznie zmiennej Dec, a zmiennej Dec przypisano daną typu (int). Wykonanie programu powoduje wyprowadzenie liczby 13.

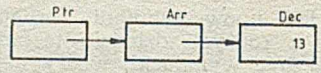
```
#include <stdio.h>

int Dec = 13,
    *Arr[1][1] = { &Dec },
    *(*Ptr)[1] = Arr;

main()
{
  printf("%d", ***Ptr);
}
```

Wydruk 3. Złożone wskazania pośrednie

Rys. 3



Na wydruku 3 przedstawiono program zawierający zestaw deklaracji, w którym rolę zmiennej Ref z poprzedniego wydruku, pełni jednoelementowa, dwuwymiarowa tablica Arr. Jak wynika z porównania rys. 2 i rys. 3, układ powiązań między zmiennymi jest niemal taki sam. Różnica polega na tym, że zmiennej Ptr z wydruku 2 jest przypisane wskazanie zmiennej prostej, natomiast zmiennej Ptr z wydruku 3 jest przypisane wskazanie tablicy. Z tego powodu, w celu wyprowadzenia danej przypisanej zmiennej Dec, należy w instrukcji printf posłużyć się wyrażeniem ***Ptr a nie **Ptr.

Uzasadnienie, że wykonanie programu z wydruku 3 istotnie powoduje wyprowadzenie liczby 13 jest następujące:

- zmiennej Dec przypisano daną o wartości 13,
- jedynemu elementowi tablicy Arr przypisano wskazanie zmiennej Dec,
- zmiennej Ptr przypisano wskazanie (jednoelementowego) pierwszego wiersza tablicy Arr,
- wyrażenie ***Ptr jest interpretowane jak *(*(*(Ptr))),
- wyrażenie Ptr jest nazwą zmiennej prostej,
- wyrażenie *Ptr jest nazwą pierwszego wiersza tablicy Arr, a więc jest nazwą wskazania elementu Arr[0][0],
- wyrażenie **Ptr jest nazwą elementu Arr[0][0],
- wyrażenie ***Ptr jest nazwą danej wskazywanej przez element Arr]0][0[, a więc jest nazwą zmiennej Dec,
- w rozpatrywanym programie napis ***Ptr mógłby zostać zastąpiony napisem Dec.

Należy nadmienić, że ci wszyscy, którzy traktują wskazania jak adresy uznaliby zapewne, że wykonanie programu z wydruku 3 odpowiada schematowi z rys. 4 i powoduje wyprowadzenie liczby znajdującej się w słowie pamięci o adresie 13. Interpretacja taka jest oczywiście błędna i wynika stąd, że niektórzy autorzy nazywają operację wyłuskania * (gwiazdka) adresowaniem pośrednim. Ofiarami takich stwierdzeń padają niekiedy także implementatorzy

kompilatorów. Można np. odnotować, że stało się tak w wypadku kompilatora języka C implementowanego przez firmę Supersoft, w wersji dla mikrokomputerów 8-bitowych (por. Informatyka nr 1, 1986).

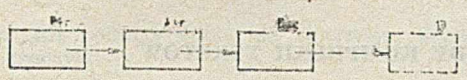
```
#include <stdio.h>

int Dec = 13,
    *Arr[1][1] = { &Dec },
    *(*Ptr)[1] = Arr;

main()
{
  printf("%d", **(int **)Ptr);
}
```

Wydruk 4. Konwersja równoważna operatorowi wyłuskania

Rys. 4



Zwolennicy mówienia o adresach bardzo często nie doceniają znaczenia konwersji między typami wskazującymi, uważając operatory konwersji za niebyłe. W wielu wypadkach nie prowadzi to do żadnych negatywnych skutków. Istnieją jednak programy, w których operator konwersji może być traktowany tak jak pewien inny operator, co z pewnością musi już być istotne. Przykład takiego programu przedstawiono na wydruku 4. W programie tym, którego wykonanie powoduje wyprowadzenie liczby 13, operator (int **) jest równoważny operatorowi * (gwiazdka).

```
#include <stdio.h>

int Arr[ ][2] = { { 1},12,13 },
    (*Ptr)[2] = Arr + 1;

main()
{
  printf("%d", (**Arr)[*Ptr]);
}
```

Wydruk 5. Nietrywialne wyłuskiwanie i indeksowanie

Na zakończenie proponuję zagadkę dla bardziej wnikliwych Czytelników. Niech będą uprzejmi przeanalizować program z wydruku 5 i szczegółowo wyjaśnić, dlaczego wykonanie tego programu powoduje wyprowadzenie liczby 13.

Rozwiązanie

• Przez domniemanie przyjmuje się, że inicjator tablicy Arr ma postać:

{{1,0}, {12,13}}

a więc, że tablica składa się z dwóch wierszy.

• Zmiennej Ptr zostaje przypisane wskazanie na wektor składający się z drugiego wiersza tablicy Arr.

• Wyrażenie **Arr jest nazwą elementu Arr[0][0]. Wynika to stąd, że Arr jest nazwą tablicy, a więc jest nazwą wskazania wektora składającego się z pierwszego wiersza tablicy Arr. Wyrażenie *Arr jest nazwą wspomnianego wektora, a więc jest nazwą wskazania pierwszego elementu tego wektora, tj. nazwą wskazania elementu Arr[0][0]; **Arr jest więc istotnie nazwą elementu Arr[0][0].

• Wyrażenie *Ptr jes nazwą wektora składającego się z drugiego wiersza tablicy Arr, a więc jest nazwą wskazania pierwszego elementu tego wektora, tzn. nazwą wskazania elementu Arr[1][0].

• Wyrażenie (**Arr)[*Ptr] jest zatem równoważne każdemu z następujących wyrażeń:

(Arr[0][0]) [&Arr[1][0]]
*(Arr[0][0] + &Arr[1][0])
*(1 + &Arr[1][0])
*(&Arr[1][1])
Arr[1][1]

• Ponieważ elementowi Arr[1][1] przypisano daną o wartości 13, wykonanie rozpatrywanego programu powoduje wyprowadzenie liczby o takiej właśnie wartości.

ROBERT CHYŁA
Centrum Obliczeniowe
Politechnika Wrocławska
KRZYSZTOF MUSIAŁ
Biblioteka Główna
i Ośrodek Informacji Naukowo-Technicznej
Politechnika Wrocławska

Metody kompresji komunikatów

Wiele zadań przetwarzanych przez komputer wiąże się z przechowywaniem w pamięci znacznej liczby danych. Często zachodzi też potrzeba przesłania tych danych przez sieć komputerową na duże odległości. Pojemność potrzebnej pamięci lub też czas przesłania danych są czynnikami, które w dużej mierze wpływają na koszt przetwarzania. Koszt ten można wydatnie zmniejszyć, stosując kompresję danych.

Niestety nie istnieje metoda zapewniająca maksymalną kompresję dla różnych rodzajów danych. Opracowano wiele metod przybliżonych, tj. takich, które pozwalają uzyskiwać stopień kompresji zbliżony do maksymalnego. Wybór metody odbywa się na podstawie charakteru danych, które mają być poddane kompresji. Pewne metody mogą okazać się znakomite w wypadku kompresji tekstów, ale dawać fatalne rezultaty w kompresji liczb, i odwrotnie. Przy wyborze metody istotne są także czasy kompresji oraz dekompresji (tj. czasy uzyskania danych w postaci pierwotnej). Tak więc systemy, w których na bieżąco są wykonywane operacje kompresji i dekompresji (np. danych wprowadzanych przez operatora terminala oraz wyświetlanych komunikatów) muszą charakteryzować się małym czasem kompresji i dekompresji. Natomiast dokonywanie kompresji danych w celu archiwizacji nie nakłada już tak ostrych wymagań czasowych.

W niniejszym artykule przedstawiono metody kompresji komunikatów polegające na zastąpieniu często powtarzających się fraz tekstu ich identyfikatorami o stałej długości. Bardzo duży wpływ na skuteczność kompresji ma odpowiedni wybór fraz do zamiany. Opisano dwie metody wyboru fraz będących przedrostkami wyrazów.

PODZIAŁ METOD KOMPRESJI

Ogólnie metody kompresji można podzielić na dwie grupy:
- metody odwracalne,
- metody nieodwracalne.

Metody odwracalne pozwalają na dokładne odtworzenie danych, które były poddane kompresji, natomiast metody nieodwracalne nie mają takiej właściwości. Metodę nieodwracalną można zastosować, np. do kompresji liczb, gdy nie zależy nam na tym, aby były one pamiętane z maksymalną dokładnością.

Metody odwracalne można z kolei podzielić na dalsze dwie grupy:
- metody semantycznie zależne;
- metody semantycznie niezależne.

Metody semantycznie zależne wykorzystują do kompresji znaczenie danych poddawanych kompresji. W metodach semantycznie niezależnych dane są traktowane wyłącznie jako ciągi znaków lub bitów — nie jest natomiast istotne, co te ciągi oznaczają.

Metoda semantycznie zależna może być wykorzystana, na przykład, przy kompresji daty. Wiedząc, że w miesiącu jest maksymalnie 31 dni, a rok składa się z 12 miesięcy oraz że data dotyczy tylko dwudziestego wieku, można zapamiętać datę na dwóch bajtach: dzień—na pięciu, miesiąc—na czterech, a dwucyfrową końcówkę roku na siedmiu bitach. Ten sposób kompresji daty jest wykorzystywany w systemie operacyjnym PC-DOS.

Bogaty przegląd tych metod przedstawił H. K. Reghbati w [2].

METODY KOMPRESJI TEKSTÓW

Istnieje wiele metod kompresji tekstów. Najpopularniejsza metoda polega na utworzeniu słownika zawierającego często powtarzające się frazy (ciągi znaków), a następnie na zmianie fraz występujących w tekście na kody będące identyfikatorami tych fraz w słowniku.

Kompresja tekstów według powyższej zasady sprowadza się do rozwiązania dwóch problemów:
- wyboru fraz do słownika,
- zastępowania fraz w tekście identyfikatorami tych fraz w słowniku.

W pracy [3] wykazano, że czas optymalnego wyboru fraz nie zależy w sposób wielomianowy od wielkości zbioru. Celowe jest więc poszukiwanie algorytmów przybliżonych. Kilka z nich zaprezentowano w [1]. Algorytmy te różnią się od siebie złożonością obliczeniową. Nie zawsze jednak algorytmy bardziej złożone dają lepsze rezultaty.

Kandydatami do umieszczenia w słowniku mogą być następujące rodzaje fraz:
- całe wyrazy,
- grupy sąsiednich wyrazów,
- przedrostki wyrazów,
- przyrostki wyrazów.

Możliwe są też kombinacje powyższych rodzajów fraz. W wypadku niewielkich tekstów zadowalające wyniki może dać nieautomatyczny wybór fraz.

Rozwiązanie drugiego z wymienionych wyżej problemów kompresji wcale nie jest oczywiste. Sytuację komplikuje fakt, że frazy ze słownika mogą mieć części wspólne. Na przykład, mając dany fragment tekstu:

STRONA TEKSTU

oraz trzy frazy ze słownika:

ST, RONA, TRON

można uzyskać różne stopnie kompresji, zależnie od kolejności wykonywania zamiany fraz na ich kody. Jeżeli najpierw nastąpi kompresja frazy TRON, to później możliwa będzie tylko kompresja frazy ST. W rezultacie dany fragment tekstu zostanie skrócony do dziewięciu znaków (zakładamy, że identyfikator frazy zajmuje jedną pozycję znakową). Jeżeli natomiast kompresja fraz nastąpi w innej dopuszczalnej kolejności, a mianowicie najpierw, ST, a potem RONA, to uzyska się zmniejszenie długości tekstu do ośmiu znaków, a więc będzie on o jeden znak krótszy.

Problem optymalnej zamiany fraz w tekście przy ustalonym słowniku został już rozwiązany. Odpowiedni algorytm przedstawiono w [4].

ALGORYTMY WYBORU I ZAMIANY FRAZ

Poniżej przedstawiono dwa algorytmy wyboru fraz do słownika, które zostały zastosowane przez autorów do kompresji komunikatów. W algorytmach tych zakłada się, że frazami są pojedyncze wyrazy lub ich przedrostki. Obydwa

algorytmy dokonują wyboru fraz z utworzonego wcześniej zbioru fraz kandydujących. Zbiór ten zawiera:

- wszystkie wyrazy występujące w tekście,
- wszystkie wyrazy wraz z dołączonymi do nich z prawej strony spacjami,
- wszystkie przedrostki wyrazów o długości większej niż jeden.

Niech dla każdej frazy x ze zbioru **F**, p_x oznacza jej liczbę wystąpień w tekście, natomiast q_x — zysk wynikający z zastąpienia frazy x jej identyfikatorem. W wypadku, gdy identyfikator jest jednobajtowy, $q_x=(|x|-1)*p_x$. Ponadto, niech **M** będzie liczbą elementów słownika **S**.

Algorytm I wyboru fraz

Do słownika należy wybrać **M** fraz o największych wartościach q_x.

Ponieważ frazy w słowniku mogą być przedrostkami innych fraz ze słownika, rzeczywisty zysk osiągnięty przez zastosowanie danej frazy x może być mniejszy od wartości q_x.

Drugi algorytm dobiera frazy do słownika, biorąc pod uwagę zysk rzeczywisty.

Algorytm II wyboru fraz

Do słownika należy wybrać frazy, dla których bieżąca wartość q_x jest maksymalna. Po każdym wybraniu danej frazy do słownika należy zmodyfikować wartości p_y i q_y dla fraz y, które nie zostały jeszcze wzięte do słownika.

```
for z in zbior fraz kandydujacych do
begin
   lz := |z| - 1;
   cz := pz;
end.
i := 0;
while i < M do
begin
   dolacz do S fraze x, dla ktorej zysk qx jest maksymalny;
   i := i + 1;
   for y not in S do
      if y jest przedrostkiem x then
         if nie istnieje w S taki przedrostek z, ze y
           jest przedrostkiem z, a z jest przedrostkiem x
         then
            cy := cy - cx;
            qy := cy * ly;
      else
         if x jest przedrostkiem y
         then
            if ly > |y| - |x|
            then
               ly := |y| - |x|; qy := cy   ly;
end.
```

Wydruk 1. Algorytm II wyboru fraz. Oznaczenia: M — pojemność słownika (maksymalna liczba fraz w słowniku); lz — efektywna długość frazy z, kandydującej do umieszczania w słowniku S; cz — efektywna liczba wystąpień fazy z; i — efektywna liczba faz w S

Algorytm ten został przedstawiony szczegółowo na wydruku 1. Uwzględniono w nim fakt, że fraza może być przedrostkiem innej frazy. Modyfikacja parametrów określających efektywną długość frazy, efektywną liczbę wystąpień oraz efektywny zysk dotyczy wszystkich fraz należących do zbioru fraz kandydujących, lecz jeszcze nie wybranych do słownika fraz. Dzięki temu, jeżeli w pewnym momencie zostanie wybrana fraza y, to określony dla niej zysk q_y będzie zyskiem rzeczywistym, tzn. liczbą bajtów, jaka rzeczywiście zostanie zaoszczędzona po podstawieniu jednobajtowego kodu w istniejące wystąpienia tej frazy.

```
procedure drukuj(t)
begin
   for c := kolejne znaki z t do
      if c jest kodem frazy then
         drukuj(element slownika o kodzie c);
      else
         drukuj_znak(c);
end.
```

Wydruk 2. Procedura dekompozycji komunikatu t

Opracowany przez autorów algorytm zmiany fraz w komunikatach na ich kody w słowniku fraz polega na tym, że każdy komunikat jest przeglądany osobno, tj. niezależnie od pozostałych. Po natrafieniu na frazę należącą do słownika, w miejsce tej frazy jest podstawiany kod wskazujący na jej pozycję w słowniku. Frazy w słowniku są wyszukiwane w sposób sekwencyjny. Kolejność doboru fraz nie jest optymalizowana. Odwrotna do powyższego algorytmu procedura dekompresji komunikatu t jest przedstawiona na wydruku 2.

KOMPRESJA KOMUNIKATÓW

Komunikaty wyprowadzane przez współczesne systemy komputerowe powinny być jak najpełniejsze oraz dostępne użytkownikowi na bieżąco. Z uwagi na konieczność szybkiego dostępu do komunikatów muszą być one przechowywane w pamięci operacyjnej. Obszar pamięci zajmowany przez komunikaty z reguły jest duży, dlatego ich kompresja jest bardzo istotna. Wskutek występowania podobnych fraz, komunikaty nadają się jednak szczególnie do kompresji. Użyta metoda kompresji musi charakteryzować się krótkim czasem uzyskania pierwotnej treści komunikatu, natomiast czas kompresji nie jest ważny, gdyż dokonywana jest ona w zasadzie jednokrotnie na etapie generowania systemu.

Autorzy dokonali kompresji angielskiej i polskiej wersji komunikatów kierowanych do użytkownika w systemie FRAMEWORK na IBM PC.

Wyniki kompresji komunikatów A — zbiór komunikatów angielskich, P — zbiór komunikatów polskich, AI — algorytm I wyboru fraz do słownika, AII — algorytm II wyboru fraz do słownika

Zbiór i algorytm		Wielkość zbioru przed kompresją	Wielkość słownika	Wielkość zbioru po kompresji	Współczynnik kompresji
Zbiór A	AI	17 307	1324	11 439	1,36
	AII	17 307	1694	9 623	1,53
Zbiór P	AI	19 056	1219	12 260	1,41
	AII	19 056	1639	10 606	1,56

W tabeli przedstawiono wyniki kompresji uzyskane dla dwóch różnych zbiorów komunikatów oraz dla słowników utworzonych za pomocą dwóch wyżej przedstawionych algorytmów wyboru fraz. Kompresji dokonano z zastosowaniem opisanego algorytmu zamiany fraz. Liczba elementów słownika M była równa 128. Wykorzystano fakt, że znakom, z których złożone były komunikaty, odpowiadały kody mniejsze od 128, tak że pozostało jeszcze do wykorzystania 128 wolnych kodów mieszczących się w jednym bajcie. Kompresji poddano nie tylko komunikaty, ale także same frazy ze słownika, postępując tak samo jak w wypadku komunikatów.

Dekompresja komunikatów wymaga stałej obecności słownika w pamięci operacyjnej. Dlatego w tabeli współczynnik kompresji jest ilorazem wielkości zbioru przed kompresją przez sumę wielkości zbioru po kompresji i wielkości słownika.

Algorytm II wyboru fraz do słownika pozwala uzyskać lepsze wyniki kompresji, lecz jest bardziej złożony od algorytmu I. W wypadku kompresji komunikatów nie ma to jednak większego znaczenia. Obydwa algorytmy dają lepsze współczynniki kompresji dla zbioru komunikatów polskich. Jest to naturalne, ze względu na większą rozwlekłość języka polskiego niż angielskiego.

Dodatkowym argumentem za stosowaniem kompresji komunikatów jest to, że są one wówczas przechowywane w pamięci w postaci niejawnej, co stanowi pewne utrudnienie dla osób starających się je samowolnie zmienić.

LITERATURA

[1] Fraenkel A. S., Mor M., Perl Y.: Is Text Compression by Prefixes and Suffixes Practical? Acta Informatica, Vol. 20, pp. 371—389, 1983

[2] Reghbati H. K.: An Overview of Data Compression Techniques. Computer, No. 1, pp. 71—75, 1981

[3] Storer J. A., Szymański T. G.: Data Compression via Textual Substitution. Journal of the ACM, Vol. 29, pp. 928—951, 1982

[4] Wagner R. A.: Common Phrases and Minimum-Space Text Storage. Communications of the ACM, No. 3, pp. 148—152, 1973.

Systemy personalne IMB PS/2 — nowość czy tylko face-lifting?

Po sukcesie serii mikrokomputerów IBM PC nadszedł czas wprowadzenia na rynek nowego produktu. W tydzień po zaanonsowaniu przez firmę Apple nowego mikrokomputera MAC II, opartego na mikroprocesorze Motorola 68020, firma IBM zaprezentowała nową serię mikrokomputerów osobistych opartych na mikroprocesorach serii Intel 86. Nowa rodzina została określona równie chwytliwą co poprzednia nazwą IBM PS/2 (Personal System).

W serii nowych systemów personalnych (chyba musimy zacząć się przyzwyczajać do tego określenia — nową terminologię IBM przedstawiono na str. 24 w nr. 10, 1987 (INFORMATYKI) zaprojektowano nowe mikrokomputery z nowym rozwiązaniem centralnej magistrali. Układy grafiki, sterowanie portem szeregowym oraz równoległym, sterownik dysków, a także zegar są umieszczone na głównej płycie. Zmieniono też technologię rozmieszczenia elementów, ułatwiając ich składanie i ewentualną naprawę. Systemy te są wyposażone w adaptery dyskietek 3,5-calowych oraz w dyski stałe (co najmniej 20 MB) oraz w łącze do dysków optycznych IBM 200 MB. Oferowana jest też — po raz pierwszy przez IBM — mysz z dwoma przyciskami.

Modele rodziny PS/2

Wykaz poszczególnych modeli serii oraz ich główne cechy charakterystyczne zamieszczono w tabeli 1, zaczerpniętej z Byte'a (nr 6, 1987). Należy przede wszystkim zwrócić uwagę na nowe rozwiązanie centralnej magistrali — zwanej Micro Channel, a umożliwiającej zależnie od wersji, równoległe przesyłanie 16 lub 32 bitów danych. Oprócz wymienionych podstawowych modeli przewidywane są ich mutacje wyposażone w większe, szybsze lub dodatkowe dyski stałe. Dostępny będzie — na przykład — adapter ESDI (ang. enhanced small device unit — jaki eufemizm!) umożliwiający szybki transfer danych z (lub do) dysków stałych. Jako dodatkowe proponowane są zewnętrzne adaptery dla dyskietek 5,25-calowych. Zwiększenie częstotliwości zegara (od dawna stosowane w komputerach kompatybilnych z IBM PC) oraz nowe rozwiązania współpracy z dyskami pozwalają firmie IBM na stwierdzenie, że Model 30 jest 2,5 raza szybszy od IBM PC/XT, a pozostałe od 2 do 3,5 raza w stosunku do IBM PC/AT. Warto jednak zwrócić uwagę, że większe modele systemu nie są jeszcze dostępne. Ciekawe jest również porównanie proponowanych cen podstawowych konfiguracji — bez uwzględnienia ceny monitora (od 250 do 685 dolarów) oraz ceny nowego oprogramowania.

Systemy współpracy z monitorami

W systemach PS/2 dostępne są trzy nowe rozwiązania współpracy z monitorami (tabela 2):

- **MCGA** (ang. multicolor graphics system)
Blok systemu zawiera dwuportową pamięć 64 KB RAM, pamięć 16 KB RAM przeznaczoną na generator znaków oraz układy realizacji palety 256 K kolorów. System może pracować w jednym z trzech trybów oraz przy dodatkowym wyposażeniu w tzw. trybie „720". MCGA może emulować również dotychczas istniejące systemy CGA oraz EGA (ten ostatni — po dołączeniu dodatkowego adaptera).

- **VGA** (ang. videa graphics system)
Blok systemu jest złożony z 12 750 bramek. Realizuje on wszystkie tryby systemu MCGA, systemu EGA oraz dwa tryby dodatkowe. System VGA jest arbitrem pomiędzy pamięcią a procesorem oraz pamięcią ekranu a układami wyświetlania.

- **VGA** (ang. video graphics system)
Dodatkowy adapter graficzny wykorzystuje zewnętrzne wyjście wideo systemu przez 20-stykowe połączenie z 16-bitową magistralą Micro Channel, zastępując tym samym układy wideo z głównej płyty systemu. Zwiększenie do 256 liczby kolorów wybieranych z 256 K kolorów palety wymaga dodatkowej karty rozszerzenia pamięci. W trybie zmniejszonej rozdzielczości dostępne są dodatkowe możliwości programowania kształtów znaków, proporcjonalnego rozmieszczania oraz wypełniania pól wzorkiem.

Oba systemy CMGA i VGA mogą współpracować z jednym z czterech monitorów analogowych:

Model 8503 — monochromatyczny o średniej rozdzielczości z możliwością wyświetlania punktów w 64 odcieniach szarości (z systemu MCGA i VGA).

Tabela 1. Charakterystyka modeli systemu PS/2

Model	30—021	50—021	60—041	80—041	80—111
Mikroprocesor	8086	80286	80286	80386	80386
Szybkość procesora	8 MHz	10 MHz	10 MHz	16 MHz	20 MHz
Koprocesor	8087	80287	80287	80387	80387
Szerokość magistrali	16 bitów	16 bitów	16 bitów	32 bity	32 bity
Standard RAM	640 KB	1 MB	1 MB	1 MB	2 MB
Maksimum RAM	640 KB	7 MB	15 MB	16 MB	16 MB
Standard ROM	64 KB	128 KB	128 KB	128 KB	128 KB
Liczba złączy	3	4	8	8	8
Rodzaj złączy	IBM PC	Micro Channel	Micro Channel	Micro Channel	Micro Channel
Pojemność dyskietki	720 KB	1,44 MB	1,44 MB	1,44 MB	1,44 MB
Pojemność dysku stałego (maksymalna)	20 MB	20 MB (20 MB)	44 MB (88 MB)	44 MB (88 MB)	115 MB (230 MB)
Tryb grafiki	MCGA	VGA, EGA, MCGA	VGA,EGA MCGA	VGA, EGA MCGA	VGA, EGA MCGA
Monitor 1024×768 pikseli	brak	dołączany	dołączany	dołączany	dołączany
System operacyjny	DOS. 3.3	DOS 3.3, OS/2	DOS 3.3, OS/2	DOS 3.3, OS/2	DOS 3.3, OS/2
Dostępność	czerwiec 1986	czerwiec 1986	czerwiec 1986	lipiec 1987	listopad 1987
Cena podstawowa	2295 dolarów	3595 dolarów	5295 dolarów	6995 dolarów	10995 dolarów

Tabela 2. Charakterystyka trybów grafiki

System/tryb	Liczba kolumn	Rozdzielczość	Pole znaku	Liczba kolorów
MCGA				
tekst	80	640×400	8×16	16 z 256 K
grafika	80	320×200	8×8	256 z 256 K
grafika	80	640×200	8×8	2 z 256 K
grafika	80	640×480	8×16	2 z 256 K
tekst	80	720×400	9×16	—
VGA				
tekst	80	720×400	9×16	—
grafika	80	640×480	—	16 z 256 K
8514/A				
grafika	—	1024×768	—	256 z 256 K
grafika	—	640×480	—	—

Model 6512 — kolorowy o średniej rozdzielczości,

Model 8513 — kolorowy o średniej rozdzielczości,

Model 8514 — kolorowy o wysokiej rozdzielczości.

Nowy „stary BIOS"

Model 30 rodziny PS/2 ma BIOS identyczny z wersją działającą na mikrokomputerach serii IBM PC/XT. Zachowana została więc pełna możliwość przeniesienia programów dotychczas istniejących, a nie korzystających z dodatkowych chwytów przy dostępie do pamięci ekranu lub — zależnych od szybkości procesora.

Pozostałe modele mogą wykorzystywać jedną z dwóch nowych rozszerzonych wersji BIOS-a. Pierwsza z nich zwana CBIOS (ang. Compability BIOS) umożliwia dostęp do 1 MB pamięci, druga — ABIOS (ang. Advanced BIOS) dostarcza operacji wielozadaniowości oraz umożliwia dostęp do 16 MB pamięci. Wszystkie programy BIOS rezydują w pamięci stałej typu 27256 ROM oraz mają jednakowy adres wejścia F000:FFFE. W implementacji funkcji BIOS-a zmieniono nieco nazwy lub działanie niektórych przerwań, przy czym według stwierdzeń firmy IBM nie ma to większego znaczenia dla istniejących już programów.

* * *

Równocześnie z nowym systemem PS/2 firma IBM anonsuje nowe lub unowocześnione urządzenia dodatkowe oraz oprogramowanie. Reklamowany jest na przykład dysk optyczny 200 MB oraz trzy rodzaje drukarek, IBM Proprinter II, X24, XL24 czy Quietwriter III, dających kopie o wysokiej jakości z wieloma już sprawdzonymi opcjami. O rozszerzeniu systemu operacyjnego DOS 3.3 oraz nowym systemie OS/2 (Operating System/2) — w następnym numerze. Oczywiście, wraz z ujawnieniem nowych, konkretnych planów firmy IBM, pojawi się masa opracowań, analiz, a wreszcie książek w rodzaju „Inside... without tears". Niestety, będą znacznie większe trudności w poznaniu wnętrza tych systemów, gdyż nie opublikowano schematów komputerów, opisów funkcjonalnych systemów graficznych i magistrali systemowej oraz wydruku BIOS-ów. Tym samym znacznie bardziej utrudnione będzie tworzenie komputerów kompatybilnych oraz różnego rodzaju „ekstra-super-turbo" rozszerzających i przyspieszających układów i programów. Na przykład, wobec umieszczenia układów portów szeregowych na głównej płycie, wystąpią kłopoty z dołączeniem autonomicznych kart do różnego rodzaju komunikacji. Co prawda twierdzi się, że wykorzystywane w PS/2 mikroprocesory są standardowe, bez tajnych zmian w maskach, ale „autoalarmy" można umieścić w innych miejscach. Myślę jednak, że dotychczasowe doświadczenia „kopistów" będą procentować.

Podając informacje o PS/2 zaczyna lowi 80-041. Na razie jednak brak jest istniejącemu, 32-bitowemu mikrokomputerowi MAC II. Komputer ten odpowiada jeszcze niedostępnemu modelowi 80-041. Na razie jednak brak jest dokładnych danych porównawczych. Porównuje się jedynie ceny. A więc za 6996 dolarów można kupić gotowego do pracy MAC-a II, a za 6995 dolarów (1 dolar taniej) standard modelu 80-041. Do pełnej odpowiedniości funkcjonalnej z komputerem MAC brakuje jeszcze monitora, koprocesora oraz systemu operacyjnego (w sumie za 1800 dolarów). Z kolei, do MAC-a można kupić za 1500 dolarów kartę AST Research Mac286 umożliwiającą wykonywanie programów napisanych dla systemu operacyjnego PC-DOS.

Oczywiście, trudno jest jednoznacznie odpowiedzieć na pytanie zawarte w tytule. Mikroprocesory serii Intel 80xxx są obarczone wieloma genetycznymi cechami umożliwiającymi wykonywanie programów napisanych dla ich starszych wersji, ale tym samym mającymi dużo funkcjonalnych niejednorodności. Podobnie, oprogramowanie nowego systemu zakłada prawie pełną przenośność już istniejących programów, a więc często również tych, które działały pod nadzorem systemu CP/M. Oba te czynniki powodują, że „potężny" ze względu na wielkość pamięci i mocy obliczeniowej system PS/2 oraz działający na nim system operacyjny OS/2 (lub DOS 3.2) mają nałożone więzy mocno je ograniczające. Z kolei, model 30 jest tylko zmodernizowaną wersją PC/XT, która już od dawna istnieje pod postacią wielu kompatybilnych odmian. Mogę więc zaryzykować stwierdzenie, że PS/2 jest tylko naturalnym krokiem ku unowocześnieniu sprawdzonego w milionach egzemplarzy systemu. Tym niemniej znowu został dokonany postęp, a my tak naprawdę jeszcze nie poznaliśmy (a co gorsze, nie wdrożyliśmy) dotychczasowych rozwiązań.

Opracował
WACŁAW ISZKOWSKI
na podstawie Byte,
June 1987

Język C — propozycja polskiej terminologii (2)

W bieżącym numerze publikujemy dokończenie proponowanego nazewnictwa dla języka C (Red.).

operator

Operatory języka C można podzielić na: operatory nawiasowe ({ } i [ ]), strukturowe (—› i :), arytmetyczne (+ — * / % ++ ——), porównywania (‹ ‹= › ›= == !=), logiczne (! && ||), bitowe (~ & | ^), przesunięć (‹‹ ››), przypisania (m.in. = += —=), konwersji (nazwa typu), i rozmiaru (sizeof).

opracowanie

Pojęcie opracowanie dotyczy zinterpretowania deklaracji albo wyrażenia. W następstwie zinterpretowania deklaracji są ustalane atrybuty obiektu reprezentowanego przez identyfikator, a w następstwie opracowania wyrażenia powstaje dana stanowiąca rezultat tego opracowania.

oznacznik

```
struct Cplx{
  float re,im;
};
struct Cplx var;
```

Identyfikator var reprezentuje zmienną typu (struct{ float re, im; }) zadeklarowaną za pomocą oznacznika Cplx.

parametr

```
fun (max,min)
{ return max == min; }
```

Parametrami funkcji fun są identyfikatory max i min.

pole

```
struct{
  int var;
  float arr[4];
  struct{
    float re,im;
  } complex;
}Str;
```

Polami struktury Str są var, arr i complex. Polami struktury complex są re i im.

pole bitowe

```
struct{
  unsigned head : 2;
  unsigned      : 3;
  unsigned tail : 2;
}Str;
```

Struktura Str składa się z trzech pól bitowych.

prolog

Prolog jest to zestaw czynności wykonywanych tuż przed podjęciem wykonywania instrukcji grupującej, albo tuż przed podjęciem wykonywania programu. Stosownie do sytuacji mówi się o prologu instrukcji grupującej albo prologu programu.

```
static var;
main(){
  int Var = 4;
  printf ("%d", var + Var);
}
```

W prologu programu zmienna var otrzymuje wartość początkową 0. W prologu instrukcji grupującej stanowiącej ciało funkcji main, zmienna Var otrzymuje wartość początkową 4.

przeciążenie

```
struct{
  int var;
} str;
int var;
```

Identyfikator var jest przeciążony, gdyż reprezentuje zarówno składową struktury str jak i zmienną prostą var.

przesłonięcie

```
fun(par)
  char par;
{ int par = 5;
  return par;
};
```

W obrębie instrukcji grupującej deklaracja parametru par jest przesłonięta przez deklarację zmiennej par.

przypisanie

Przypisanie jest czynnością polegającą na związaniu ze zmienną ściśle określonej danej. Dana taka ma zawsze wartość, ale wartość ta może być nieokreślona. Jawne przypisanie danej odbywa się za pomocą operacji przypisania. Niejawne przypisanie odbywa się podczas kojarzenia parametrów funkcji z jej argumentami oraz podczas wykonywania prologu instrukcji grupującej i programu.

pseudodeklarator

```
sizeof (int *[3])
```

Argumentem operatora sizeof jest napis zawierający pseudodeklarator *[3].

punkt początkowy deklaracji

Punktem początkowym deklaracji jest punkt, od którego rozpoczyna się jej zakres. Punkt ten występuje bezpośrednio po deklaratorze identyfikatora.

```
fun(par)
  int par;
{   char chr = par;
    return chr == par;
}
```

Punktem początkowym deklaracji parametru jest punkt przed pierwszym średnikiem. Punktem początkowym deklaracji zmiennej chr jest punkt przed pierwszym znakiem „=".

rezultat

Rezultatem opracowania wyrażenia jest dana. Rezultatem opracowania wyrażenia zawartego w instrukcji return jest dana, która poddana ewentualnej konwersji na daną takiego typu jak identyfikator funkcji stanowi rezultat wywołania funkcji.

rozmiar

```
int arr[3][4]
```

Rozmiar drugiego wymiaru tablicy arr wynosi 4.

skojarzenie

```
main(){
  fun(5);
}
fun(par)
  int par;
{ print ("%d,, , par); }
```

Parametr par funkcji fun zostaje skojarzony z argumentem reprezentowanym przez literał 5. Nie ujawniony tu drugi parametr funkcji printf zostaje skojarzony z argumentem par reprezentowanym przez parametr par funkcji fun.

skutek uboczny

```
int var = 5;
main(){
  printf ("%d %d", fun(),var);
}
fun()
{ return var = 8, 2; }
```

Z wykorzystaniem funkcji fun jest związany skutek uboczny w postaci zmiany wartości zmiennej globalnej var. W następstwie wykonania programu są wyprowadzane liczby 2 i 5 albo liczby 2 i 8.

stała
Stałą jest obiekt przetwarzania, którego wartość nie może ulec zmianie. Stałymi w języku C są obiekty reprezentowane przez literały arytmetyczne i znakowe.

synonim typu
```
typedef struct Cplx{ float re,im; } cplx;
struct complex { float re,im; } ;
```
Typy (struct Cplx), cplx i (struct complex) są synonimami.

sytuacja wyjątkowa
Sytuacją wyjątkową jest odstępstwo od spodziewanego przebiegu wykonania programu. Nie zawsze jest ono przejawem błędu. Przykładem sytuacji wyjątkowej jest dzielenie przez zero, powstanie nadmiaru, odwołanie się do tablicy z niewłaściwymi indeksami itp.

typ danej
Typem danej jest taki zestaw jej atrybutów, który w pełni i jednoznacznie opisuje wszystkie właściwości danej — poza jej wartością. Określeniem typ obejmuje się także zbiór danych o ustalonych wartościach, mówiąc że dana jest pewnego typu, gdy jej wartość należy do tego zbioru.

wiązanie
Ponieważ wiązanie operatora „+" (plus) jest lewostronne, a wiązanie operatora „=" (znak równości) jest prawostronne, wyrażenie
```
a = b = c + d + e
```
jest interpretowane tak, jak wyrażenie
```
a = (b = ((c + d) + e))
```

widoczność
Identyfikator jest widoczny w zasięgu jego deklaracji.

wskazanie
Wskazaniem jest dana wskazująca.
```
char Name [] = "jb";
main()
{ printf ("%s", Name); }
```
Wyrażenie Name występujące w wywołaniu funkcji printf reprezentuje wskazanie pierwszego znaku tablicy Name.

wskazanie plikowe
```
main()
{ fclose (fopen ("kaja","w")); }
```
Argumentem funkcji fclose jest wskazanie plikowe.

wyrażenie
Wyrażeniem jest napis stanowiący opis czynności, których wykonanie powoduje utworzenie danej. Mówi się wówczas, że wyrażenie reprezentuje tę daną.
```
char arr[] = "JanB",
    *ref = arr;
main(){
  printf ("%c%c%c%c%c", *arr,
                        *++ref,
                        1 [ref],
                        arr [3])
}
```
Argumentami wywołania funkcji printf są wyrażenia.

wyrażenie stałe
Każde wyrażenie stałe reprezentuje stałą. Nie każde wyrażenie reprezentujące stałą jest wyrażeniem stałym. W zasięgu deklaracji
```
int arr[2][3];
```
wyrażeniami stałymi są: arr, arr[0] i & arr[0][0].

zakres deklaracji
```
main(){
  char var = 'b';
  { char var = 'j';
    putchar(var);
  }
  putchar(var);
}
```
Zakresem deklaracji pierwszego identyfikatora var jest fragment programu od następującego po tym identyfikatorze znaku „=" aż do ostatniego nawiasu klamrowego.

zasięg deklaracji
```
main(){
  char var = 'b';
  { char var = 'j'
    putchar(var);
  }
  putchar(var);
}
```
Zasięgiem deklaracji pierwszego identyfikatora var jest fragment programu od następującego po tym identyfikatorze znaku „=" aż do ostatniego nawiasu klamrowego, ale z wykluczeniem fragmentu wewnętrznej instrukcji grupującej od napisu var do najbliższego nawiasu klamrowego włącznie.

zmienna
Zmienną jest obiekt przetwarzania, którego wartość może ulec zmianie. Zmiana taka dokonuje się w następstwie przypisania danej. Zmienne dzielą się na zmienne proste i argumenty zmiennych.

zmienna wskazująca
Zmienną wskazującą jest zmienna, której można przypisywać dane wskazujące.
```
main(){
  char *Ptr;
  Ptr = "jb";
  printf("%s", Ptr);
}
```
Zmiennej Ptr przypisano daną wskazującą pierwszy znak tablicy reprezentowanej przez literał "jb".

znak widoczny
Znakiem widocznym jest znak spacji oraz każdy znak, który w sposób widoczny ujawnia się na ekranie monitora lub drukarce.

Z OSTATNIEJ CHWILI

Pojawienie się pierwszych implementacji zgodnych ze standardem ANSI, w tej liczbie implementacji języka Turbo C, przyczyniło się do rozszerzenia słownika terminów.

model
Obranym modelem pamięci może być tiny, small, compact, medium, large i huge. Stosownie do obranego modelu i deklaracji, wskazania mogą być realizowane jako bliskie (near), dalekie (far) albo odległe (huge).

modyfikator
Język podstawowy, w którym występuje jedynie modyfikator unsigned, rozszerzono o następujące modyfikatory: signed, cdecl, pascal, const, volatile, near, far, huge, interrupt.

odrzucenie danej
Jawne odrzucenie danej następuje na skutek poprzedzenia wyrażenia operatorem (void).

prototyp
Prototypem jest deklarator funkcji, w której jawnie określono typy wszystkich parametrów funkcji.

punkt charakterystyczny
Punktem charakterystycznym jest punkt, poza który nie sięgają skutki uboczne wykonania programu. Przyjmuje się, że punktami charakterystycznymi są: miejsce, w którym zakończono opracowywanie argumentów funkcji, miejsce wystąpienia operatora koniunkcji, alternatywy, warunki i połączenia, oraz miejsce, w którym zakończono opracowywanie kompletnego wyrażenia.

wskazanie adresowe
Wskazaniem adresowym jest wskazanie typu (void *).

zmienna ustalona
Zmienną ustaloną jest zmienna, która została zadeklarowana z modyfikatorem const.

JAN BIELECKI

Kreczmar A.: Języki obiektowe (1)
INFORMATYKA 1988, nr 1, s. 2
Pierwsza część charakterystyki języków obiektowych zawierająca wyjaśnienie pojęcia obiektu oraz sposobu jego użycia w językach programowania.

Kreczmar A.: Object languages (1)
INFORMATYKA 1988, No. 1, p. 2
First part of object language characteristics, which contains explanation of object conception and method of its use in programming languages.

Kreczmar A.: Objektsprachen (1)
INFORMATYKA 1988, Nr. 1, S. 2
Erster Teil einer Charakteristik von Objektsprachen, der Begriffserläuterung des Objektes und Methode für seine Anwendung in Programmiersprachen umfasst.

Baumbach H. D.: Stan i tendencje rozwoju technologii programowania w kombinacie Robotron
INFORMATYKA 1988, nr 1, s. 4
Charakterystyka zasad stosowanych w kombinacie Robotron przy produkcji oprogramowania.

Baumbach H. D.: Actual situation and development trends of software technology in Robotron combine
INFORMATYKA 1988, No. 1, p. 4
Characteristics of rules applied for software production in Robotron combine.

Baumbach H. D.: Heutiger Stand und Entwicklungstrends der Programmiertechnologie im Kombinat Robotron
INFORMATYKA 1988, Nr. 1, S. 4
Eine Charakteristik von Grundlagen der Softwareerzeugung im Kombinat Robotron.

Paprocki A.: ADA/SM — kompilator podzbioru Ady dla komputerów SM-4
INFORMATYKA 1988, nr 1, s. 6
Charakterystyka opracowanego w Instytucie Maszyn Matematycznych w Warszawie kompilatora języka Ada dla komputerów typu SM-4.

Paprocki A.: ADA/SM — an Ada-subset compiler for SM-4 computers
INFORMATYKA 1988, No. 1, p. 6
Characteristics of Ada-subset compiler for SM-4 computers, which was elaborated in the Institute of Mathematical Machines in Warsaw.

Paprocki A.: ADA/SM — ein Ada-Untermenge-Kompilierer für SM-4-Rechner
INFORMATYKA 1988, Nr. 1, S. 6
Eine Charakteristik des Ada-Untermenge-Kompilierers für SM-4-Rechner, der im Institut für Mathematische Maschinen in Warschau erarbeitet wurde.

Popko J.: Jednostka centralna Mazovii 1016
INFORMATYKA 1988, nr 1, s. 9
Charakterystyka sprzętowa jednostki centralnej mikrokomputera Mazovia 1016 oraz ocena jej zgodności z wzorcem IBM PC.

Popko J.: CPU of Mazovia 1016
INFORMATYKA 1988, No. 1, p. 9
Hardware characteristics of the Mazovia 1016 CPU and evaluation of its compatibility with IBM PC.

Popko J.: Zentraleinheit von Mazovia 1016
INFORMATYKA 1988, Nr. 1, S. 9
Hardwareorientierte Charakteristik von Mazovia 1016-Zentraleinheit und Bewertung ihrer Kompatibilität mit IBM PC.

Gąsiorek W.: Pewne problemy konstrukcji i oprogramowania systemowego dla Mery 300
INFORMATYKA 1988, nr 1, s. 11
Charakterystyka uniwersalnych rozwiązań prowadzących do wzrostu efektywności działania systemów programowania na minikomputerach.

Gąsiorek W.: Some problems on hardware and system software for Mera 300
INFORMATYKA 1988, No. 1, p. 11
Characteristics of universal solutions, which assure productivity intensification of minicomputer programming systems.

Gąsiorek W.: Gewisse Hard- und Systemsoftwareprobleme für Mera 300
INFORMATYKA 1988, Nr 1, S. 11
Eine Charakteristik der universellen Lösungen, die zur Effektivitätssteigerung der Programmierungssysteme für Minicomputer beitragen.

Zielczyński P.: Turbo Prolog
INFORMATYKA 1988, nr 1, s. 14
Ogólna charakterystyka kompilatora języka Turbo Prolog oraz jego porównanie z innymi, dostępnymi na IBM PC, translatorami tego języka.

Zielczyński P.: Turbo Prolog
INFORMATYKA 1988, No. 1, p. 14
General characteristics of the Turbo Prolog compiler and its comparison with other available on IBM PC translators for this language.

Zielczyński P.: Turbo Prolog
INFORMATYKA 1988, Nr. 1, S. 14
Allgemeine Charakteristik von Turbo Prolog-Kompilierer und sein Vergleich mit anderen auf IBM PC erreichbaren Übersetzern für diese Sprache.

Płoski Z.: Chiwriter — procesor tekstu (2)
INFORMATYKA 1988, nr 1, s. 18
Dokończenie charakterystyki procesora tekstów Chiwriter oraz ocena jego wartości użytkowej w porównaniu z innymi programami tego typu.

Płoski Z.: Chiwriter — a text processor (2)
INFORMATYKA 1988, No. 1, p. 18
Termination of the Chiwriter text processor characteristics and evaluation of its usability in comparison with other similar programs.

Płoski Z.: Chiwriter — ein Textprozessor (2)
INFORMATYKA 1988, Nr. 1, S. 18
Beendigung einer Charakteristik von Chiwriter-Textprozessor und Bewertung seiner Anwendungsfähigkeit im Vergleich mit anderen ähnlichen Programmen.

Syfert A., Raznowiecki A.: Struktura systemu operacyjnego PC-DOS (4)
INFORMATYKA 1988, nr 1, s. 21
Czwarta część charakterystyki systemu operacyjnego PC-DOS zawierająca omówienie wywołań funkcji specyficznych dla tego systemu.

Syfert A., Raznowiecki A.: Structure of the PC-DOS operating system (4)
INFORMATYKA 1988, No. 1, p. 21
Fourth part of the PC-DOS operating system characteristics, which contains discussion of calls for system's specific functions.

Syfert A., Raznowiecki A.: Struktur des PC-DOS-Betriebssystems (4)
INFORMATYKA 1988, Nr. 1, S. 21
Vierter Teil einer Charakteristik von PC-DOS-Betriebssystems, der eine Besprechung von Aufrufen spezifischer Funktionen dieses Systems umfasst.

Bielecki J.: Język C — wskazania, adresy, konwersje
INFORMATYKA 1988, nr 1, s. 24
Zasady oraz praktyczne wskazówki posługiwania się wskazaniami, adresami i konwersjami w języku C.

Bielecki J.: C language — pointers, addresses, conversions
INFORMATYKA 1988, No. 1, p. 24
Principles and practical advices for using pointers, addresses and conversions in the C programming language.

Bielecki J.: C-Sprache — Zeiger, Adressen und Umwandlungen
INFORMATYKA 1988, Nr. 1, S. 24
Grundsätze und praktische Hinweise für Anwendung von Zeigern, Adressen und Umwandlungen in der C-Programmiersprache.

Chyła R., Musiał K.: Metody kompresji komunikatów
INFORMATYKA 1988, nr 1, s. 26
Klasyfikacja i charakterystyka najczęściej używanych metod kompresji tekstów oraz ich praktycznego zastosowania do kompresji komunikatów.

Chyła R., Musiał K.: Message compression techniques
INFORMATYKA 1988, No. 1, p. 26
Classification and characteristics of generally applied techniques for text compression and its practical use for message compression.

Chyła R., Musiał K.: Methode für Meldungenverdichtung
INFORMATYKA 1988, Nr. 1, S. 26
Klassifikation und Charakteristik der meist verwendeten Methoden für Textverdichtung und ihre praktische Anwendung für Meldungenverdichtung.

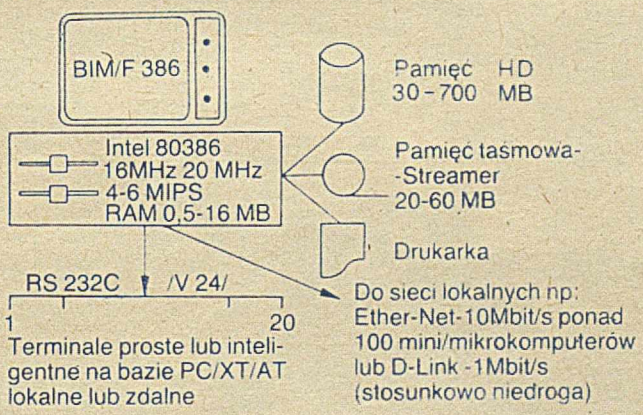
PZ globo®

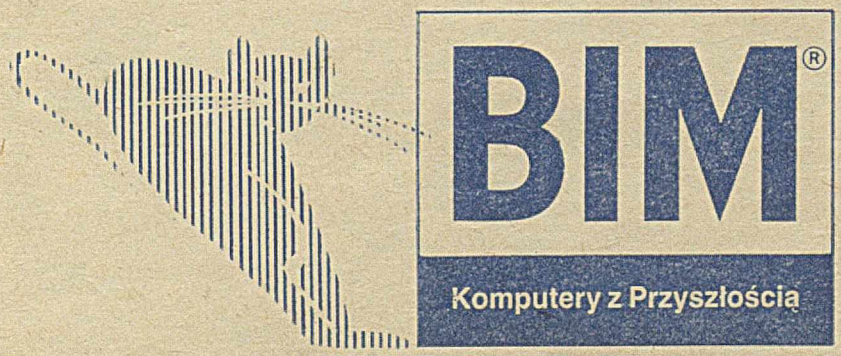
Licencjonowany producent firmy BIM
Koplin 73 200 Choszczno Tel. 7550 Telex 0445413

Oferuje nowoczesne, niezawodne, bardzo wydajne:

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

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





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

EO/614/87

6-letnie doświadczenie software'owe sprawdzone w ponad 1000 zakładach pracy

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

System CSK/32 to:

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

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

WDROŻENIA, SZKOLENIA
— zespoły ds. wdrożeń i szkoleń pracują w:
Gdyni tel. 248 018, tlx 054792 • Krakowie tel. 373 573 • Poznaniu tel. 676 271
Wrocławiu tel. 481 679 • Warszawie tel. 258 528, tlx 816711
Sopocie — w wyspecjalizowanym Salonie Sprzedaży, ul. Kombatantów 1

computer studio kajkowscy ™

EO/74/88

MIĘDZYWOJEWÓDZKA SPÓŁDZIELNIA PRACY „SIÓDEMKA"

ŁÓDŹ, AL. KOŚCIUSZKI 93,

ZAKŁAD INFORMATYKI I SYSTEMÓW KOMPUTEROWYCH

poleca po **najniższych cenach** w kraju:

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

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

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

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

Proponujemy również na wszelkiego rodzaju mikrokomputery programy wspomagające zarządzanie przedsiębiorstwem:

— system finansowo-księgowo-kosztowy
— system zbytu i zaopatrzenia
— system technicznego przygotowania produkcji
— system materiałowy
— system kadrowy i kadrowo-płacowy (również dla pracowników akordowych)

Wymienione systemy pracują w wersjach sieciowych i wielodostępnych.

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

EO/1005/87