mikroKLAN 1



1 1984

# informatyka

MODULA-2
Komputeryzacja prac inżynierskich
CACM

WYDAWNICTWO SIGMA

# informatyka

1 1984

Nr 1
Miesięcznik Rok XIX
Styczeń 1984

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





Materiałów nie zamówionych redakcja nie zwraca

Redakcja: 00-041 Warszawa, ul. Jasna 14/16, pok. 243 i 244, tel. 27-71-40 lub 26-82-61 w. 184

Zakł. Graf. „Tamka". Zam. 2352. Obj. 4,9 ark. druk. Nakład 3650 egz. T-25.

INDEKS 36124

Cena egzemplarza zł 75,—
Prenumerata roczna zł 900,—

WYDAWNICTWO NACZELNA ORGANIZACJA TECHNICZNA SIGMA CZASOPISM I KSIĄŻEK TECHNICZNYCH


W NUMERZE:



*Witamy w oczekiwanym — 1984 roku. Zapowiedzi wróżbitów, proroków, literatów i publicystów zapewne nie spełnią się... Tym goręcej życzymy wytrwałości na własnej drodze ku dobremu.*

*Redakcja*

# Nasz mały reformizm

Zmienialiśmy się już, zmieniamy, będziemy zmieniać... Mimo wysiłków nie udaje się jednym ruchem wyskoczyć z INFORMATYKĄ poza zaścianek. Trudno, uznajemy siłę realiów — pozostaniemy na miarę otoczenia, także tego swojsko-informatycznego.

Ale poprawiać się będziemy. Skoro nie uda się od razu stworzyć pisma na znacznie wyższym poziomie kompetencji i edytorskiej sprawności, skoro nie można nim zaraz starych murów skruszyć, porządek zaprowadzić gdzie bałagan — będziemy drążyć po maleńku. Teraz jakby nie czas na rewolucyjne przekształcenia.

Postanowiliśmy — wobec wyraźnej (i u nas już!) erupcji mikroinformatyki — oddać nieco łamów amatorom techniki mikroprocesorowej (mikroKLAN). Zrozumiały to pomysł w jedynym ogólnopolskim czasopiśmie informatycznym. Liczyliśmy przy tym, że przezorny wydawca zwiększy może objętość, poprawi warunki poligrafii. Takie infantylne, przyznajmy, marzenia.

Objętość pisma zmniejszono (tymczasowo!) o osiem stron i żadne nasze racje nie okazały się przekonywające — „trudności na odcinku poligraficznym" i już. Oczywiście, innowacje edytorskie są także nie do przeprowadzenia — pod groźbą usunięcia z drukarni.

Budżet pisma nie wynika z jego potrzeb czy ewentualnych (potencjalnych) możliwości, lecz z sytuacji finansowej wydawcy.

Wprawdzie INFORMATYKA przez ostatni rok przynosiła deficyt, niemniej łatwo byłoby — dzięki kilku śmielszym posunięciom — nadać jej rozmach i w efekcie — zacząć zarabiać...

Ale — jak już deklarowałem — będziemy się poprawiać. Namawiać ludzi, by wbrew wszystkiemu pisali, wojować z drukarnią o bardziej urozmaicony druk, zawracać dysponentom głowę o lepszy papier, wprowadzać na łamy grafikę (nie tylko komputerową), podwyższać zawodową wagę pisma i drążyć jego społeczny kontekst. Wprawdzie bez nadziei na znalezienie się poza zaściankiem, ale z przekonaniem, że walka ze zgodą na jego trwanie ma sens, że już sama w sobie jest zapowiedzią lepszego.

Mimo zmniejszenia łamów, postanowiliśmy oddać mikroKLANOWI osiem stron. Nazywamy go wkładką, gdyż zajmuje środkowe strony pisma, dając się oddzielić od reszty. Jest to jednak integralna część INFORMATYKI, zawierająca się w docelowym charakterze czasopisma. I w jego cenie.

Resztę zmuszeni jesteśmy maksymalnie ścieśnić, wyżąć całą wodę, jaka chciałaby przepłynąć przez te łamy. A to oznacza w gruncie rzeczy zwiększenie trudności. I aby ta praca mogła przynieść korzystne efekty, będziemy musieli rozszerzyć krąg stałych współpracowników redakcji i autorów. Starając się stworzyć dogodne tło dla rozsądnych wypowiedzi i rzetelnych dokonań „informatycznego pisarstwa". Tylko tą drogą.

Nie będziemy się naiwnie odwoływać do wigoru i zapału dysponentów. Odwołamy się raczej do przytomnych umysłów „informatycznych", które nie bacząc na ograniczenia, bezwład i ogólne zniechęcenie, dopatrzą się w informatyce szansy na lepsze życie. Również w wymiarze społecznym.

Tylko rozrosłe biurokratyczne potęgi mogą się lękać (mikro)informatycznych przeobrażeń, obawiać rozwoju drobnych, ale rzutkich, ruchliwych konkurentów. Dla człowieka coś tworzącego, kreatywnego, który szuka nie zabezpieczających go instytucjonalnych form, lecz przede wszystkim — poczucia sensu własnego działania, wsparcie komputerowe może mieć niebagatelne znaczenie. Może nadać społeczną (kulturalną, organizacyjną, gospodarczą, technologiczną...) wagę pojedynczemu wysiłkowi, wzmocnić indywidualne dążenie wobec chaosu ruchów zbiorowych. Po prostu siła jednostki dobrze organizującej swoje prace może być większa od nieoperatywnie działających instytucji.

Podaję to wszystko raczej na wiarę, nie mając jeszcze dostatecznie wiarygodnych dowodów. Sądzę jednak, że kto dostrzega tę komputerową ścieżkę na tle naszych bezdroży, temu jej jasno oświetlać nie trzeba. Kto nie widzi — ten i tak nas nie zasili.

ZBIGNIEW GLUZA

PIOTR FUGLEWICZ
CNPSS MERASTER
Katowice

MODULA-2 — język lat osiemdziesiątych (1)

# Charakterystyka języka

Pod koniec lat sześćdziesiątych nie istniał żaden język programowania odpowiedni dla przejrzystego zapisu technik określonych mianem programowania strukturalnego. Do nauki podstaw programowania strukturalnego nie obarczonego nieregularnościami i wadami istniejących języków Niklaus Wirth zaprojektował nowy i elegancki język — PASCAL. Po dziesięciu latach badania problemów związanych z PASCALEM stworzył język MODULA-2, który ma go zastąpić.

## EWOLUCJA PASCALA

Pomimo początkowego braku poparcia ze strony przemysłu komputerowego, PASCAL, dzięki łatwości wyrażania algorytmów i struktur danych, szybko wszedł na rynek zastosowań komercjalnych. Popularność PASCALA rosła w ciągu lat siedemdziesiątych i obecnie jest on jednym z najczęściej używanych języków dla mini- i mikrokomputerów.

Wśród różnorodnych opinii na temat tego języka pojawiały się często wzajemnie sprzeczne. Brian Kernighan uznał PASCALA za „odpowiedni dla małych, ograniczonych programów o znikomym oddziaływaniu z otoczeniem i nie korzystających z programów pisanych przez kogokolwiek innego; bowiem PASCAL jest narzędziem, przydatnym do nauczania, ale nie do prawdziwego programowania" [2]. Natomiast Hoare uważał, że „największą siłą PASCALA jest to, że ma on tak mało nieistotnych właściwości, iż nie ma praktycznie potrzeby tworzenia jego podzbiorów. Dlatego właśnie język jest dostatecznie silny, aby można go stosować w specjalizowanych rozszerzeniach".

Producenci sprzętu, oprogramowania oraz naukowcy, uznając słuszność uwag zarówno Kernighana, jak i Hoare'a, zaczęli sami tworzyć własne, rozszerzone wersje PASCALA. Niestety, poszczególne odmiany miały ze sobą niewiele wspólnego, a i sposób projektowania rozszerzeń na ogół nie był zgodny z założeniami PASCALA.

Alarmujący wzrost liczby odmian rozszerzonego PASCALA doprowadził w naturalny sposób do powstania idei zdefiniowania standardowego zbioru rozszerzeń języka. Spośród kilku prób normalizacji w zasadzie żadna się nie powiodła, jako że czas stworzenia normy minął już parę lat temu. Obecnie rozpowszechnionych jest tak wiele rozszerzeń, że nawet jeśli np. ANSI opracuje swój standard, nie należy liczyć, że zostanie on powszechnie przyjęty.

W 1977 roku Wirth stwierdził: „Jeśli język okazuje się być tylko marginalnie przydatny do pewnych zastosowań. to oczywiście znaczy, że wyobraźnia jego twórcy była niedostateczna i że trzeba mieć odwagę zbudować nowe, naprawdę adekwatne narzędzie, zamiast rozbudowywać istniejące" [5]. Zdanie to padło w artykule opisującym nowy język — MODULA, który był wynikiem prac nad problemami związanymi z wieloprogramowaniem i obsługą urządzeń wejścia—wyjścia.

MODULA składa się z minimalnego podzbioru PASCALA, do którego dodano strukturę modularną, ulepszoną składnię oraz możliwość wieloprogramowania i dostępu do niskiego poziomu maszyny. MODULA pozbawiona była, natomiast, pascalowej organizacji plików oraz wskaźników. O ile brak pierwszej z tych cech stanowił znaczną niewygodę, to modularna budowa, w porównaniu z blokową organizacją PASCALA, stanowiła krok naprzód, taki jak bloki w porównaniu z fortranowskim wywołaniem procedury.

Jakkolwiek MODULA jest jeszcze używana na niektórych wyższych uczelniach, to Wirth zaprzestał prac nad tym językiem po spędzeniu roku w Xerox Palo Alto Research Center, gdzie pracował na komputerze osobistym ALTO, programując w języku MESA. Po powrocie do Szwajcarii rozpoczął prace nad opracowaniem podobnego, lecz prostszego systemu komputerowego, obejmującego zarówno sprzęt, jak i oprogramowanie. Jak pisze Wirth — „System ten (później nazwany LILITH) został oprogramowany w jednym tylko języku, który w związku z tym musiał zaspokoić wymagania zarówno opisu systemu na wysokim poziomie, jak i programowania tych fragmentów, które ściśle współpracują z konkretnym sprzętem. MODULA-2 jest wynikiem rozważań nad językiem, który zawiera wszystkie elementy PASCALA rozszerzone o różne koncepcje modularności oraz (...) wieloprogramowości" [6].

## PODSTAWOWE WŁAŚCIWOŚCI MODULI-2

MODULA-2 jest językiem uniwersalnym, ukierunkowanym na implementacje systemów, językiem zawierającym najlepsze elementy swych poprzedników. Struktury instrukcji oraz typy danych są rozszerzeniem pascalowych, a modularność, dostęp do sprzętu i wieloprogramowość stanowią ulepszenie odpowiednich właściwości MODULI.

W porównaniu z PASCALEM, MODULA-2 charakteryzuje się m.in. następującymi właściwościami:

- modularnością, a w szczególności — możliwością podziału programu na część definicyjną i implementacyjną
- istnieniem współprogramów (podstawa wieloprogramowości)
- dostępem do sprzętu na poziomie języka maszynowego
- istnieniem typu procedurowego (możliwość przyporządkowywania zmiennej całej procedury).

### Modularność

Każdy program jest modułem, który sam może składać się z kilku modułów. Obiekty (stałe, typy, zmienne, procedury) zadeklarowane i zdefiniowane w jednym module — mogą być użyte w innym, jeśli zostaną do niego zaimportowane — ang. import (przykład 1).

```
(* Przykład 1 — Prosty moduł pisania na monitorze. Procedura
WriteInt jest importowana ze standardowego modułu InOut *)

MODULE ProstyProgram;
FROM InOut IMPORT WriteInt;
BEGIN
WriteInt(999)
END ProstyProgram.
```

Moduły, z których mogą być importowane obiekty (np InOut) nazywane są modułami bibliotecznymi, inne, np. programy użytkowe — modułami klienta, albo klientami (ang. client). Każdy program biblioteczny jest nie tylko podporządkowany innym modułom, ale może również sam zawierać podporządkowane moduły, tzn. może importować obiekty z innych modułów bibliotecznych. Powstaje w ten sposób hierarchia modułów, przy czym moduły klienta stanowią poziom najwyższy. Moduł, który nie zawiera żadnej listy importowej jest na poziomie najniższym.

W odróżnieniu od innych języków programowania (FORTRAN, PASCAL), kompilator MODULI-2 musi mieć dostęp do opisu importowanych obiektów. Tym sposobem

unika się błędów syntaktycznych (np. błędny parametr w wywołaniu procedury) oraz semantycznych (błędna nazwa). Gdyby w przykładzie 1 nie istniała importowana procedura lub parametr byłby niewłaściwy — np. WriteInt ("999") — sygnalizacja błędu nastąpiłaby w czasie kompilacji. Rozłączenie części definicyjnej i implementacyjnej modułu bibliotecznego umożliwia zmianę modułu bez konieczności pisania na nowo (powtórnej kompilacji), modułów importujących. Pozwala to też na odpowiednią ochronę oprogramowania przed piractwem programistycznym. Te obiekty, które mają być widziane poza modułem (a więc te, które mają być importowane) należy zadeklarować w części definicyjnej i wymienić je w liście eksportowej. Użytkownik, który importuje obiekty z tego modułu, musi znać tylko część definicyjną. Część implementacyjna może istnieć tylko w postaci skompilowanej, a więc nieczytelnej dla osób postronnych.

Moduł tworzy ścianę wokół swoich obiektów. Poza jawnie zadeklarowanymi jako eksportowane (ang. export), obiekty modułu są izolowane od otaczającego programu. Zdolność do kapsułkowania (ang. encapsulation) wewnętrznych obiektów czyni moduły tak silnym narzędziem, że stają się one odpowiednikiem programowym „czarnej skrzynki".

Moduły dzielą duży program na logicznie powiązane części, ze sprzęgami zdefiniowanymi przez identyfikatory importowe i eksportowe. Program napisany w MODULI-2 jest wskutek tego czytelniejszy, łatwiejszy do zrozumienia i mniej podatny na efekty uboczne, niż pascalowski program o strukturze blokowej.

Zdolność modułów do podziału programu na prawie niezależne części jest zapewniona przez oddzielną kompilację. W przykładzie 2 przedstawiono trzy osobno kompilowane moduły.

```
(* Przykład 2 — Trzy oddzielnie kompilowane moduły: biblioteczny SystemPlikow, jego moduł implementacyjny oraz moduł, który z niego korzysta. Klient nie ma swojego modułu definicyjnego, ponieważ niczego nie importuje *)

DEFINITION MODULE SystemPlikow;
  EXPORT QUALIFIED
    FILE, Otworz, Zamknij, Czytaj, CzytajSlowo,
    Pisz, Blad;
  TYPE FILE;
  PROCEDURE Otworz (VAR f: FILE; nazwa: ARRAY OF CHAR);
  PROCEDURE Zamknij (VAR f: FILE);
  POCEDURE Czytaj (f: FILE; VAR ch: CHAR);
  PROCEDURE Pisz (f: FILE; ch: CHAR);
  PROCEDURE CzytajSlowo (f: FILE, i: CARDINAL);
. . . . . .
  PROCEDURE Blad (f: FILE): BOOLEAN;
END SystemPlikow.
(* ------------------------------------------------------------ *)
IMPLEMENTATION MODULE SystemPlikow;
FROM SYSTEM IMPORT ADDRESS, ADR;
CONST DlugoscNazwy = 20;
TYPE rozmiarbloku = [0..511];
TYPE FILE = POINTER TO RECORD
      . . . . . .
      END (* FILE *);
PROCEDURE Otworz (VAR f: FILE; nazwa: ARRAY OF CHAR);
  (* Tresc procedury *)
END (* Otworz *);
. . . . . .
BEGIN
  (* Inicjowanie zmiennych *)
END SystemPlikow.
(* ------------------------------------------------------------ *)
MODULE ProgramGlowny;
  FROM SystemPlikow IMPORT
    FILE, Otworz, Zamknij, Czytaj, Blad;
VAR f1, f2: FILE;
    ch: CHAR;
BEGIN
  Otworz (f1, "zbior.dat");
  . . . . . .
  Czytaj (f1, ch);
  IF Blad (f1) THEN HALT END;
  . . . . . .
  Zamknij (f1);
END ProgramGlowny.
```

Moduł biblioteczny SystemPlikow realizuje proste operacje wejścia—wyjścia, eksportując typ FILE i operacje plikowe. Typ FILE jest typem „nieprzezroczystym" (ang. opaque), co oznacza, że pola rekordu są dostępne wewnątrz modułu implementacyjnego, tylko nazwa typu jest wydawana na zewnątrz modułu.

Jeśli lista eksportowa zawiera słowo QUALIFIED, momoduł klienta musi poprzedzić wszystkie odwołania do identyfikatorów modułu bibliotecznego jego nazwą, np. identyfikator Otworz powinien być wywoływany jako SystemPlikow.Otworz. Eksport kwalifikowany umożliwia uniknięcie kolizji nazw modułu bibliotecznego i modułu klienta. W module klienta można uniknąć powtarzania nazwy modułu bibliotecznego przez użycie symbolu FROM oraz identyfikatora modułu bibliotecznego przed listą importową.

Moduł ProgramGlowny, importuje SystemPlikow dla dokonania operacji wejścia—wyjścia. Równocześnie może on być wywołany przez inny program jako podprogram (wywołania podprogramów są wykonywane przez procedurę CALL importowaną ze standardowego modułu Program).

**Kompilacja rozłączna**

W większości obecnie używanych języków programowania możliwa jest oddzielna kompilacja części programu. Na ogół w jej trakcie nie jest sprawdzana zgodność typów. Stanowi to poważny problem przy integracji dużych systemów. MODULA-2 umożliwia kontrolę zgodności typów między modułami. Skompilowana postać modułu definicyjnego zawiera informację o typach obiektów, która jest dostępna podczas kompilacji zarówno odpowiedniego modułu implementacyjnego, jak i modułów klientów.

Rozdzielenie części definicyjnej i implementacyjnej umożliwia zmianę treści modułu implementacyjnego bez konieczności rekompilacji modułów klienta. Jeśli moduł definicyjny pozostaje bez zmian, wymagana zgodność typów zostaje zachowana. Pozwala to uniknąć rekompilacji całego programu, co jest szczególnie wygodne przy dużych programach. Zmiana części definicyjnej modułu bibliotecznego czyni niepoprawnymi wszystkie moduły klientów bezpośrednio lub pośrednio importujące dany moduł, ponieważ zmienia się sprzęg między modułem a klientem.

Dla rozwiązania tego problemu, MODULA-2 ma możliwość kontroli zgodności modułów na poziomie egzekutora. W trakcie kompilacji modułu definicyjnego kompilator przypisuje mu niepowtarzalną wartość określoną jako klucz modułu. W trakcie kompilacji odpowiadający moduł implementacyjny otrzymuje klucz modułu definicyjnego. Moduł klienta importujący moduł biblioteczny ma wpisywaną kopię klucza modułu importowanego do swojej postaci wynikowej — w czasie kompilacji. W trakcie wykonywania programu, program ładujący sprawdza, czy klucze zapisane w modułach klientów są zgodne z kluczami modułów bibliotecznych. Ponieważ moduły implementacyjne otrzymują swoje klucze z odpowiadających modułów definicyjnych, część implementacyjna może być optymalizowana, modyfikowana lub nawet wymieniona bez żadnego wpływu na moduł klienta.

Jeśli jednak rekompilowany jest moduł definicyjny (i co za tym idzie — odpowiadający mu moduł implementacyjny), np. wskutek dodania nowej zmiennej, otrzymuje on nowy klucz modułu. Po wykonaniu modułu klienta z nieaktualną biblioteką sygnalizowany jest „błąd wersji", co oznacza, że moduł klienta musi być rekompilowany.

Korzyści dla implementacji systemów pisanych przez wielu programistów są oczywiste. Jeśli program jest zdefiniowany dostatecznie dobrze i w projekcie określono powiązania między jego poszczególnymi częściami, można rozpowszechnić moduły biblioteczne w postaci skompilowanej na wstępie pracy. Programiści piszący te moduły mogą w nich następnie dokonywać poprawek, nie zmuszając klientów do rekompilacji swoich modułów po wprowadzeniu każdej zmiany.

**Biblioteki**

W PASCALU operacje wejścia—wyjścia oraz operacje na plikach dokonywane są przez system operacyjny, po którym działa program. W MODULI istniał system zarządzający procesami, natomiast MODULA-2 nie potrzebuje żadnej z tych możliwości. Ponieważ powyższe operacje są

w MODULI-2 programowane jako moduły biblioteczne, nie ma dodatkowego sprzęgu między językiem a sprzętem.

Typowymi operacjami zawartymi w bibliotece MODULI-2 są: operacje konsoli wejściowo—wyjściowej, konwersje tekstów i reprezentacji dwójkowych, odczytywanie, zapisywanie i wyszukiwanie plików, operacje na katalogach plików, zarządzanie pamięcią, wykonywanie programów, zarządzanie procesami, funkcje matematyczne, operacje napisowe.

Większość tych operacji jest na ogół domeną systemu operacyjnego, jednak w dążeniu do stworzenia jednolitego i wszechstronnego środowiska obarczono te systemy taką nadmiarowością, że zajmują one często bardzo dużą część pamięci i czasu maszyny. Moduły biblioteczne zawierają natomiast tylko te funkcje, które są niezbędne w konkretnej aplikacji. Takie podejście pozwala na blokowy rozwój oprogramowania: moduły biblioteczne wysokiego poziomu są budowane z modułów niższego poziomu, duże programy — z małych. Możliwe jest dzięki temu tworzenie oprogramowania przenośnego, które może być przenoszone bez zmian, lub z niewielkimi zmianami, pomiędzy różnymi systemami komputerowymi. Zbiór standardowych modułów definicyjnych tworzy dla modułu klienta otoczenie operacyjne niezależne od użytego systemu operacyjnego. Przeniesienie programu pod inny system wymaga jedynie napisania zbioru modułów implementujących standardowe moduły definicyjne.

MODULA-2 umożliwia również włączenie istniejącego oprogramowania do swojego otoczenia. Programy pisane w innych językach mogą być wywoływane jako podprogramy, stając się częścią programu w MODULI-2. Specjalne moduły definicyjne umożliwiają dostęp do bibliotek spoza systemu MODULA-2, do funkcji systemu operacyjnego, a nawet do rozkazów maszynowych.

### Otoczenie programu

Systemy oparte na MODULI-2 wymagają innego otoczenia niż powszechnie stosowane języki. Kompilator, program ładujący i biblioteka — tworzą zintegrowany system z programem w trakcie jego wykonywania. Dodanie modułu do biblioteki (lub aktualizacja istniejącego) wymaga jedynie jego kompilacji. Kompilator zapisuje postać wynikową do biblioteki, tak że gdy moduł (lub jego klient) jest wykonywany, program ładujący tylko go pobiera.

Pozwala to na uniknięcie redagowania (łączenia) programów i wynikających z tego powodu problemów. Pojedyncza kopia nowego zbioru może być używana przez wiele programów. Ponieważ ta metoda jest wolniejsza niż wykonywanie programów połączonych, istnieje opcjonalna możliwość uzyskiwania połączonej wersji programu, szczególnie przydatna w przypadku tworzenia oprogramowania na większej maszynie dla mniejszego komputera docelowego. Dodatkowo — program tak uzyskany może pracować na komputerze docelowym bez żadnego systemu operacyjnego, zawierając w sobie jego niezbędne funkcje.

### Procesy i wieloprogramowanie

Przez proces rozumie się ciąg obliczeń. Możliwość wieloprogramowania oznacza, że może być realizowanych równocześnie kilka procesów. Komunikacja między procesami jest realizowana przez zmienne współdzielone i sygnały. Na sygnałach mogą być dokonywane tylko dwie operacje: wystawienie sygnału i oczekiwanie na sygnał. Wystawienie sygnału oznacza, że określony (przez programistę) warunek jest spełniony. Proces oczekujący na sygnał kontynuuje działanie, gdy warunek związany z sygnałem zostanie spełniony.

Do jednej zmiennej w jednym czasie nie należy odwoływać się z różnych modułów. Dlatego zmienne współdzielone umieszcza się w jednym module, który gwarantuje wyłączny dostęp. Moduł taki nazywa się monitorem. Eksportuje on — na przykład — dwie procedury DEPOSIT i FETCH, za pomocą których można umieszczać i pobierać zmienne z monitora.

W jednoprocesorowym systemie całe wieloprogramowanie oparte jest na koncepcji współprogramu. Współprogram jest odpowiednikiem procesu, ale współprogramy są zawsze quasi-równoległe (tzn. nie mają własnych procesorów) i przekazywanie sterowania między nimi odbywa się w sposób jawny za pomocą procedury TRANSFER. Współprogram otrzymujący sterowanie podejmuje pracę w miejscu ostatnio przerwanym przez TRANSFER.

Większość nowoczesnych języków programowania zawiera pewne funkcje umożliwiające wieloprogramowanie. W MODULI są to procesy i semafory, w ADZIE zadania i spotkania, a CONCURRENT PASCAL zawiera monitory. MODULA-2 proponuje prostsze i realizowane na niższym poziomie — współprogramy.

Obsługa współprogramu jest zapewniona w module SYSTEM przez procedury NEWPROCESS, TRANSFER i IOTRANSFER oraz typ PROCESS. Procedura NEWPROCESS tworzy nowy współprogram, a jej parametrami są: nazwa procedury, adres i obszar pamięci, w której ma być wykonywana. Identyfikator nowego współprogramu podawany jest przez procedurę i ma typ PROCESS. Wywołanie współprogramu dokonywane jest przez procedurę TRANSFER, której parametrami są identyfikatory programów — zawieszanego i uaktywnianego. TRANSFER zawiesza działanie aktywnego współprogramu i uaktywnia nowy. Procedura IOTRANSFER pozwala używać współprogramów jako programów obsługi przerwań. Dodatkowym parametrem jest tu adres wektora przerwań. Procedura przekazuje normalnie sterowanie nowemu procesowi, jednak w chwili przerwania sterowanie jest automatycznie przekazywane do współprogramu jego obsługi.

Program ilustrujący użycie współprogramów przedstawiono w przykładzie 3. Moduł standardowy SYSTEM w rzeczywistości jest pseudomodułem, tzn. że identyfikatory, które eksportuje, są implementowane na poziomie języka maszynowego, a maszynowo zależne deklaracje znane są kompilatorowi jako predefiniowane.

```
(* Przykład 3 — Zastosowanie współprogramów do komunikacji
z urządzeniem zewnętrznym. Główny program tworzy dwa współ-
programy Hej i Ho, i przekazuje sterowanie do Hej zawieszając
się. Hej wykonuje IOTRANSFER, przekazując sterowanie do Ho.
Ho wypisuje na ekranie "ho", jednak kiedy następuje przerwanie
zewnętrzne do wektora o adresie 64 (konsola operatorska) stero-
wanie jest automatycznie przekazywane do Hej. Hej pisze na
ekranie "hej" i zwraca sterowanie do Ho. *)

MODULE HejHo;
  FROM SYSTEM IMPORT
    WORD, ADR, SIZE, PROCESS, NEWPROCESS,
    TRANSFER, IOTRANSFER;
  FROM Terminal IMPOT WriteString, Writeln;
  CONST maxHejHo = 17;
  VAR i: CARDINAL;
    Hej, Ho, Main: PROCESS;
    A, B: ARRAY [1..200] OF WORD;
  PROCEDURE PiszHej;
  BEGIN
    zLOOP IOTRANSFER (Hej,Ho,64);
    WriteString("hej");
    END;
  END PiszHej;
  PROCEDURE PiszHo;
  BEGIN
    LOOP WriteString("ho");
    INC(i);
    IF i > maxHejHo THEN
    Writeln; i := 0;
    END;
    END;
  END PiszHo;
BEGIN i := 0;
  NEWPROCESS (PiszHej, ADR(A), SIZE(A), Hej);
  NEWPROCESS(PiszHo, ADR(B), SIZE(B), Ho);
  TRANSFER(Main, Hej);
END HejHo.
```

Współprogramy wystarczają do wielu zastosowań wieloprogramowości jednakże jeśli wymagane są bardziej wyrafinowane metody sterowania procesami (semafory, przekazywanie komunikatów), to mogą być także zaimplementowane przy użyciu współprogramów i dołączone do innych modułów bibliotecznych. MODULA-2 umożliwia więc użycie różnych metod synchronizacji i komunikacji procesów, sama nie zawierając żadnej.

### Dostęp do sprzętu

Siłą języków wysokiego poziomu jest możliwość wykrycia w trakcie kompilacji odstępstw od reguł języka. Z drugiej strony — koncepcja dokładnie zdefiniowanych struktur danych może prowadzić do istotnych ograniczeń.

W języku MODULA-2 predefiniowane są dwa różne typy liczb całkowitych, których zakresy wartości przecinają się: typ INTEGER i CARDINAL (bez znaku). Istnieje również typ BITSET, który w zasadzie podaje, czy określony bit jest ustawiony, czy nie. Wszystkie trzy typy charakteryzują się tym, że ich definicja jest zależna od maszyny, a nie od języka Każda zmienna tych trzech typów wymaga dokładnie jednego słowa pamięci. W zależności od komputera słowo ma 8, 16, 32 lub więcej bitów. Dlatego na różnych komputerach funkcja konwersji typów ma różne znaczenie. Dla słów 8-bitowych funkcja CARDINAL (—1) ma wartość 255, dla 16 bitów 65535. Analogicznie BITSET (—1) ma wartość odpowiednio [0,..., 7] oraz [0,...,15]. Funkcje konwersji typów umożliwiają ominięcie warunku zgodności typów i są ograniczone do typów zajmujących taki sam obszar w pamięci.

MODULA-2 umożliwia dostęp z języka wysokiego poziomu do praktycznie wszystkich elementów sprzętu. Udogodnienia takie, zawarte w modułach bibliotecznych, są z natury rzeczy maszynowo zależne, a więc różne w różnych implementacjach MODULI-2. Moduł SYSTEM eksportuje systemowo zależne identyfikatory WORD, ADDRESS, SIZE, TSIZE i ADR. Użycie typu WORD jako typu parametru zapewnia zgodność z dowolnym obiektem zajmującym słowo maszynowe. WORD użyty jako typ bazowy tablicy dynamicznej (ARRAY OF WORD) czyni ją zgodną z typem dowolnego rozmiaru, łącznie z rekordami i plikami. Umożliwia to pisanie procedur uniwersalnych, działających na zmiennych dowolnego typu (przykład 4).

```
(* Przykład 4 — Czytanie danych w dowolnym formacie z pliku
dyskowego. Moduł importuje procedury z modułu SystemPlikow
oraz standardowego modułu InOut. Pierwsze słowo pliku określa
jego długość w słowach. Dane wprowadzane z dysku pisane są
na ekranie w postaci liczb typu INTEGER *)

MODULE Kopiuj;
FROM SystemPlikow IMPORT CzytajSlowo, FILE, Otwórz,
                         Zamknij;
FROM InOut       IMPORT WriteInt;
VAR dlug : CARDINAL;
    i    : INTEGER;
    ZbiorWe : FILE;
BEGIN
Otworz(ZbiorWe, "ZBIOR1");
CzytajSlowo(ZbiorWe,dlug); dlug : = dlug—1;
WHILE dlug > 0 DO
  CzytajSlowo(ZbiorWe,i);dlug : = dlug—1;WriteInt(i)
  END;
Zamknij(ZbiorWe);
END Kopiuj.
```

Typ ADDRESS jest zgodny z typem CARDINAL (liczba całkowita bez znaku) oraz wszystkimi typami wskaźnikowymi. Na zmiennych typu ADDRESS dopuszczalne są wszystkie operacje arytmetyczne. Umożliwia to tworzenie programów bezpośrednio zarządzających pamięcią, zwłaszcza że MODULA-2 tłumaczy standardowe procedury NEW i DISPOSE na odwołania do wykonywanych na niskim poziomie procedur ALLOCATE i DEALLOCATE. Te procedury są importowane ze standardowego modułu zarządzania pamięcią, ale mogą być również definiowane w inny sposób, umożliwiając różne rodzaje gospodarki pamięcią.

Funkcje SIZE (x) i TSIZE (t) modułu SYSTEM podają rozmiar zmiennych i typów w słowach maszynowych, natomiast funkcja ADR (x) podaje adres zmiennej w pamięci. Możliwe jest deklarowanie zmiennych w określonych miejscach pamięci, co stanowi prostą metodę dostępu do rejestrów urządzeń zewnętrznych w maszynach, w których rejestrom tym odpowiada określony adres.

### Typ procedurowy

Modularna struktura języka umożliwia tworzenie fragmentów oprogramowania wielokrotnego użytku. Problem, który w związku z tym powstaje polega na tym, że różne moduły klientów stawiają różne wymagania przed modułami bibliotecznymi. Przykładowo — jednemu klientowi wystarcza obsługa błędów zawarta w module bibliotecznym, natomiast drugi woli przeprowadzić ją na poziomie swojego modułu. MODULA-2 rozwiązuje ten problem przez wprowadzenie typu procedurowego — uogólnienie pascalowych parametrów procedury. Typ procedurowy umożliwia modułowi klienta włączenie swoich własnych procedur do modułu bibliotecznego. W przykładzie 5 identyfikator będący nazwą procedury sam jest parametrem procedury.

```
(* Przykład 5 — Identyfikator funkcji przekazywany jako pa-
rametr *)

MODULE TypProcedury;
FROM MathLib IMPORT sin, cos;
FROM RealInOut IMPORT WriteReal;
TYPE Funkc = PROCEDURE(REAL): REAL;
PROCEDURE PiszWynik (p: Funkc);
  BEGIN WriteReal (p(0.65)); WriteReal (p(1.35));
END PiszWynik;
BEGIN
PiszWynik (sin); PiszWynik(cos);
END TypProcedury.
```

### MODULA-2 a PASCAL

MODULA-2 jest nie tylko rozszerzeniem PASCALA o strukturę modułową, dostęp do maszyny oraz wieloprogramowość, ale rozwiązuje też szereg innych problemów, które czyniły programowanie w PASCALU niewygodnym i niedostatecznie elastycznym.

Tablice w PASCALU mają stałą wielkość, wskutek czego programowanie operacji napisowych jest niewygodne, a tworzenie np. uniwersalnych bibliotek matematycznych — praktycznie niemożliwe. MODULA-2 zawiera otwarte parametry tablic, co pozwala na przekazywanie tablic o różnych rozmiarach. Dotyczy to również napisów deklarowanych jako ARRAY OF CHAR.

Widoczność globalnych zmiennych w PASCALU jest na ogół zbyt szeroka, co prowadzi do powstania efektów ubocznych nieprzewidzianych przez programistę. Modularna struktura MODULI-2 umożliwia ukrycie zmiennych lokalnych — tak, że stają się niewidoczne w pozostałej części programu.

Brak rozłącznej kompilacji w PASCALU utrudnia pisanie dużych programów oraz korzystanie z bibliotek. Rozłączna kompilacja w MODULI-2, wraz z pełną kontrolą zgodności typów w czasie kompilacji, rozwiązuje ten problem.

Wymagania dotyczące porządku deklaracji w PASCALU utrudniają grupowanie odpowiadających sobie deklaracji w jednym miejscu programu. MODULA-2 narzuca mniejsze ograniczenia na kolejność deklarowania typów, zmiennych, procedur i modułów, umożliwiając czytelniejszy opis programu.

W przeciwieństwie do PASCALA wyrażenia logiczne MODULI-2 obliczane są do momentu, w którym następuje jednoznaczne określenie ich wartości i dalsza część wyrażenia jest pomijana. W PASCALU występuje czasem konieczność użycia instrukcji pustej w wyrażeniach warunkowych. W MODULI-2 ograniczenie takie nie występuje.

W standardowym PASCALU brak klauzuli OTHERWISE w instrukcji CASE. MODULA-2 ma opcjonalną klauzulę ELSE, dzięki czemu można nie wyliczać wszystkich etykiet wyboru.

Możliwości PASCALA w zakresie operacji wejścia—wyjścia są bardzo ograniczone. MODULA-2 nie ma żadnych instrukcji wejścia—wyjścia, a operacje te są wykonywane przez moduły biblioteczne i mogą być w nieograniczony sposób rozszerzane lub zmieniane.

Ścisła kontrola typów w PASCALU uniemożliwia korzystanie z programowania na poziomie języka maszynowego. MODULA-2 pozwala na ominięcie tego ograniczenia, pod kontrolą języka wysokiego poziomu.

Każda instrukcja strukturalna w MODULI-2 ma swój terminator (UNTIL dla instrukcji REPEAT i END dla

pozostałych). Eliminuje to konieczność używania pascalowej konstrukcji BEGIN...END.

Wszędzie tam, gdzie PASCAL dopuszcza użycie stałych, MODULA-2 pozwala używać wyrażenia stałego (np. TYPE a = ARRAY [0..n—1] OF CHAR).

Rekordy z nawiasami używają takiej samej składni jak instrukcja CASE.

Wynik funkcji może być dowolnego typu, w tym również rekordowego i tablicowego.

Klauzula EXIT umożliwia stworzenie konstrukcji LOOP/EXIT, czyli przejście do instrukcji następnej za terminatorem pętli.

Instrukcja FOR ma opcjonalną klauzulę BY do specyfikacji kroku różnego od 1.

Wszystkie znaki identyfikatora w MODULI-2 są znaczące, co umożliwia tworzenie identyfikatorów o dowolnej długości. Duże i małe litery są rozróżniane przez kompilator.

* * *

MODULA-2 jest kolejnym krokiem w ewolucji języków programowania. W porównaniu z modnymi obecnie językami, takimi jak ADA, C, czy — na drugim biegunie — FORTH, ma szereg zalet. Jest na pewno dużo bardziej czytelna niż C i FORTH, a jej prostota czyni ją w niektórych zastosowaniach wygodniejszą od ADY. Jednocześnie ma tyle wspólnego z PASCALEM, że przestawienie się na programowanie w MODULI-2 jest bardzo łatwe, choć pełne wykorzystanie możliwości zawartych w modularnej strukturze może wymagać pewnego doświadczenia. Przypuszcza się, że tak jak powstanie PASCALA wyeliminowało całkowicie z użycia ALGOL, tak też MODULA-2 zajmie miejsce PASCALA, szczególnie w coraz powszechniejszych zastosowaniach mikrokomputerowych.

Druga część artykułu będzie zawierać systematyczny opis składni języka, natomiast trzecia zostanie poświęcona implementacji kompletnego systemu MODULA-2.

LITERATURA


[1] Amman U.: Modula-2: Eine maschinennahe, modulare Programmiersprache. Elektronik Nr. 9, 1983

[2] Kernighan B.: Why Pascal Is Not My Favorite Programming Language. Computing Science Technical Report No. 100, Bell Laboratories, Murray Hill (NJ), 1981 July 18

[3] McCormack J., Gleaves R.: Modula-2 — A Worthy Succesor to Pascal. BYTE, April 1983

[4] Summer R. T., Gleaves R. E.: Modula-2 — A Solution to Pascal's Problems. ACM SIGPLAN Notices, September 1982

[5] Wirth N.: Modula: a Language for Modular Multiprogramming. Software Practice and Experience, No. 7, 1977

[6] Wirth N.: Programming in Modula-2. Springer—Verlag, New York, 1982.



**JACEK T. MAŁUSZYŃSKI**
**Ośrodek Informatyki**
**Politechnika Poznańska**


# Odtwarzanie bazy danych techniką dzienników częściowych

Terminem odtwarzanie bazy danych określa się doprowadzenie uszkodzonej bazy danych do stanu używalności. Można wyróżnić trzy typy uszkodzeń bazy danych:

- **Uszkodzenie fizyczne** w wyniku zniszczenia fragmentów lub całości magnetycznego nośnika informacji zawierającego bazę danych. Uszkodzenie takie wymaga odtworzenia zniszczonych fragmentów bazy danych, celem uzyskania stanu bazy logicznie identycznego ze stanem istniejącym bezpośrednio przed uszkodzeniem nośnika.

- **Uszkodzenie logiczne w wyniku błędnej aktualizacji,** spowodowanej np. błędnymi danymi wejściowymi transakcji, wadami oprogramowania podstawowego lub użytkowego oraz błędami operatora lub sprzętu. Uszkodzenie takie wymaga najpierw doprowadzenia bazy danych do stanu poprzedzającego wystąpienie błędu, a następnie powtórne, tym razem poprawne przeprowadzenie transakcji, która spowodowała błąd, a także wszystkich innych transakcji, które po niej nastąpiły. Alternatywą może być przeprowadzenie dodatkowych transakcji niwelujących powstałe błędy, co jednak jest zwykle dużo trudniejsze.

Dr inż. JACEK T. MAŁUSZYŃSKI ukończył w 1972 r. studia na Wydziale Elektrycznym Politechniki Poznańskiej, a w 1980 uzyskał tytuł doktora nauk technicznych.
Od 1972 r. pracuje w Ośrodku Informatyki Politechniki Poznańskiej. W latach 1980—1981 przebywał na stażu naukowym w Centrum Obliczeniowym Uniwersytetu w Pittsburgu (USA). Obecnie zajmuje się sterowaniem współbieżną aktualizacją i odtwarzaniem bazy danych.

- **Uszkodzenie logiczne w wyniku niekompletnej aktualizacji,** tzn. przerwania jednej lub wielu transakcji, spowodowane np. błędami wymienionymi w punkcie poprzednim lub awarią zasilania. Najczęstszą przyczyną jest tu blokada (ang. deadlock). Uszkodzenie takie wymaga wycofania z bazy danych skutków działania niedokończonych transakcji aktualizacyjnych.

Do odtworzenia bazy danych we wszystkich trzech typach uszkodzeń konieczna jest jedynie kopia stanu początkowego bazy danych oraz notowanie w tzw. dzienniku transakcji, wszystkich poleceń przeprowadzenia transakcji aktualizacyjnych. Uniwersalna technika oparta na tych dwóch podstawowych mechanizmach wywodzi się z techniki stosowanej przy wsadowej aktualizacji plików sekwencyjnych, zapisanych na taśmie magnetycznej. W przypadku zniszczenia aktualnej generacji pliku, lub też błędnej czy przerwanej aktualizacji, prawidłowy stan pliku można odtworzyć przez powtórną aktualizację jego poprzedniej generacji. Dla współbieżnie aktualizowanych dużych baz danych zapisanych w pamięci dyskowej ta prosta metoda jest jednak mało efektywna i dlatego konieczne jest stosowanie szybszych algorytmów, wykorzystujących takie techniki, jak:

- **Dziennik nowych wersji.** Zawiera nowe wersje aktualizowanych danych, co pozwala uniknąć powtórnego przetwarzania wszystkich transakcji w przypadku fizycznego zniszczenia bazy danych (zapamiętanymi nowymi wersjami danych można uaktualnić starą kopię bazy danych). W celu przyspieszenia przetwarzania dziennika nowych wersji stosuje się dodatkowo pliki z indeksami dotyczącymi dziennika nowych wersji (ang. audit trail tag file) [3].

- **Dziennik starych wersji.** Zawiera stare wersje zaktualizowanych danych zapisane przed dokonaniem aktualizacji w bazie danych. Pozwala to wycofać z bazy skutki dzia-

łania pewnej liczby ostatnich transakcji aktualizacyjnych. W przypadku obu typów uszkodzeń logicznych pozwala to szybciej doprowadzić bazę danych do stanu używalności, w porównaniu do ładowania i uaktualniania jej starej kopii, jak to odbywa się w przypadku uszkodzeń fizycznych.

• **Tzw. technika opóźnionej aktualizacji** wykorzystująca chwilowe zbiory dyskowe zawierające nowe wersje danych aktualizowanych przez pojedynczą transkację. Pliki te umożliwiają niemalże bezkarne przerywanie transakcji [1, 3].

Poza wymienionymi technikami, które ogólnie można nazwać „techniką dzienników", istnieje szereg innych, które jednak nie są tak uniwersalne. Są to np.: **technika ostrożnej aktualizacji** (ang. careful replacement), wymagająca szczególnej struktury bazy danych; **technika plików różnicowych** (ang. differential files), wymagająca przerywania normalnej pracy systemu dla dokonania okresowej aktualizacji; **technika składowania przyrostowego** (ang. incremental dumping), nie gwarantująca jednak odtworzenia uaktualnień dokonanych po ostatnim składowaniu. Przegląd tych technik można znaleźć w publikacji [9]. W niniejszym artykule zostanie rozważony dokładniej pewien szczególny rodzaj techniki dzienników, a mianowicie — **technika dzienników częściowych** (ang. logical logging) [5].

## TECHNIKA DZIENNIKÓW CZĘŚCIOWYCH

Podstawowym mechanizmem techniki dzienników jest notowanie w dzienniku starych i nowych wersji aktualizowanych danych. Notowaniu podlegają zmiany wszystkich aktualizowanych danych, a więc zarówno danych wprowadzonych do bazy przez jej użytkowników (nazwijmy je danymi podstawowymi), jak i zmiany w wewnętrznych danych organizacyjnych bazy, utworzonych przez system zarządzania bazą danych — SZBD (np. wskaźniki tworzące kartoteki odwrócone). Technika dzienników częściowych polega na notowaniu jedynie starych i nowych wersji danych podstawowych oraz ignorowaniu zmian w danych organizacyjnych. Zauważmy, że pomimo nienotowania niektórych zmian baza danych może zostać odtworzona, gdyż SZBD na podstawie danych podstawowych potrafi odbudować uszkodzone wewnętrzne dane organizacyjne.

Technika dzienników częściowych ma następujące istotne zalety:

• Zmniejszenie liczby zapisów w dzienniku.

• Możliwość dobudowania do systemu informatycznego mechanizmu odtwarzania bazy danych, bez konieczności wprowadzania zmian w wykorzystywanym SZBD. Wymaga to przechwytywania wszystkich wywołań SZBD pochodzących od programów użytkowych i odnotowywania odpowiednich informacji w dzienniku częściowym przed aktualizacją bazy danych.

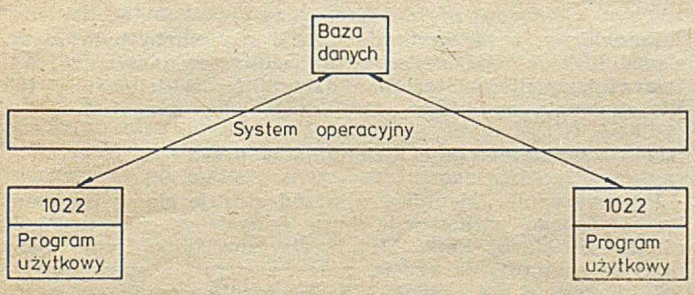
Technika dzienników częściowych ma również następujące wady:

• Odtwarzanie bazy danych jest wolniejsze od odtwarzania w przypadku „klasycznej" techniki dzienników, ponieważ wymaga korzystania z SZBD.

• Algorytm odtwarzania jest bardziej skomplikowany od algorytmów „klasycznej" techniki dzienników, ponieważ wymaga generowania poleceń dla SZBD.

## PRZYKŁAD SYSTEMU ODTWARZANIA BAZY DANYCH

Dla zobrazowania wymienionych zalet techniki dzienników częściowych może posłużyć system CRASH [7]. Jest to system odtwarzania bazy danych współdziałający z SZBD o nazwie 1022 [8], opracowanym dla komputera DEC 10.




**Rys. 1. Sposób wykorzystywania SZBD 1022**

SZBD 1022 nie jest programem wieloprocesowym, lecz pakietem podprogramów, zapewniającym podstawowe operacje zakładania i wykorzystywania relacyjnej bazy danych. Pakiet ten, konsolidowany z programem użytkowym, konkuruje z innymi podobnymi programami o dostęp do plików bazy danych zarządzanych przez monitor (rys. 1).

Główne wady SZBD 1022, które spowodowały budowę systemu CRASH, są następujące:

• dziennik starych wersji nie uwzględnia koncepcji transakcji [2] aktualizującej więcej niż jeden plik

• brak dziennika nowych wersji

• brak blokowania dla odczytu (ang. shared locking) [4]

• problem blokady jest rozwiązany w ten sposób, że po określonej liczbie nieudanych prób uzyskania wyłącznego dostępu do pliku, program użytkowy jest usuwany z systemu

• prowadzenie dziennika nowych wersji jest bardzo kosztowne ze względu na znaczną liczbę transmisji dyskowych, ponieważ o dostęp do pliku zawierającego dziennik poszczególne programy ubiegają się jak o dostęp do innych plików (za każdym razem plik jest zamykany i otwierany na nowo przez system operacyjny)

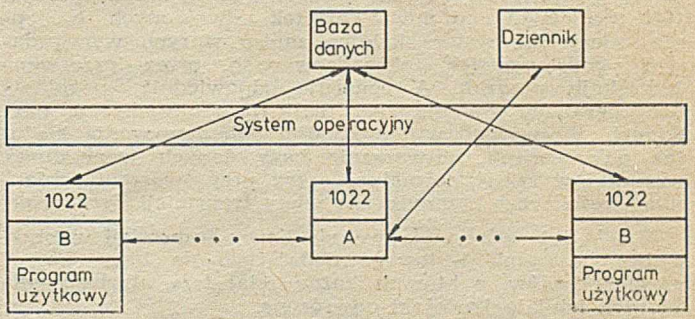
• brak mechanizmów przydatnych dla administratora bazy danych, takich jak:

— dostarczanie informacji o aktualnie działających programach użytkowych, wykorzystujących bazę danych

— dostarczanie informacji o tym, które programy użytkowe ostatnio działały i czy zakończyły się one poprawnie, czy też uległy awarii, powodując automatyczne odtwarzanie bazy danych

— informowanie o czasie trwania najdłuższych transakcji, co pozwala wykryć błędnie napisane programy użytkowe

— łagodne (po zakończeniu aktualnie działających programów) lub gwałtowne wyłączanie bazy danych z użytku.




**Rys. 2. Organizacja systemu CRASH**

Planowane zastosowanie SZBD 1022 nie wymagało szczególnie szybkiego odtwarzania bazy danych, można było więc zastosować technikę dzienników częściowych. Zbudowanie systemu CRASH pozwoliło usunąć wszystkie wymienione wady systemu 1022 bez dokonywania w nim jakichkolwiek zmian. Spośród wielu możliwych rozwiązań systemowych [6] wybrano rozwiązanie pokazane na rysunku 2. System CRASH składa się z dwóch części: **A** i **B**. Wywołania SZBD 1022 są przechwytywane przez podprogramy stanowiące część **B**. Jest to zrealizowane poprzez narzucenie programistom standardu programowania, polegającego na używaniu w programach użytkowych innych nazw dla wywołania funkcji 1022 (pierwsze litery wywołań zmieniono z pierwotnych **DB** na **DI**). Po edycji źródłowych wersji istniejących programów użytkowych, polegającej na zamianie liter **DB** na **DI** oraz rekomplikacji tych programów, współdziałają one z systemem CRASH.

System 1022 uzupełniony o system CRASH jest więc kompatybilny z oryginalnym systemem 1022. Niemniej nowe programy, pisane już z myślą o systemie CRASH, są lepsze, jeśli używa się dodatkowych podprogramów części **B**, nie mających odpowiedników w podprogramach 1022 (np. podprogramów realizujących blokowanie dla odczytu). Podprogramy części **B** generują rekordy zawierające stare i nowe wersje danych, o aktualizację których program użytkowy zwraca się do systemu 1022. Następnie przysyłają te rekordy za pośrednictwem istniejącego w monito-

rze mechanizmu komunikatów międzyprocesowych, do części **A** systemu CRASH. Po wprowadzeniu tych rekordów przez część **A** do dziennika, podprogramy **DI** wywołują odpowiednie podprogramy **DE** systemu 1022. Po przechwyceniu pierwszego wywołania dotyczącego SZBD, część **B** żąda od systemu operacyjnego utworzenia i zamknięcia semafora **S** o unikalnym identyfikatorze (identyfikatora tego używa się również do przesyłania komunikatów międzyprocesowych pomiędzy częściami **A** i **B**). Następnie część **B** przesyła ten identyfikator do części **A**, która dopisuje się do kolejki związanej z **S**.

Jeżeli program użytkowy korzystający z części **B** ulegnie awarii i zostanie usunięty z systemu, wtedy zwalniają się wszystkie zasoby blokowane przez ten program, a więc również omawiany semafor. Powoduje to z kolei przerwanie programowe sygnalizujące programowi **A**, że semafor, na który oczekiwał, został otwarty, co w tym przypadku oznacza awaryjne przerwanie transakcji. Przerwanie takie to, oczywiście, konieczność wycofania z bazy danych efektów działania przerwanej transakcji, co program **A** natychmiast czyni, posługując się starymi wersjami danych zawartymi w dzienniku. Wszystkie programy użytkowe, a ściślej odpowiednie podprogramy części **B** informują program **A** o każdym uzyskaniu od systemu operacyjnego przydziału (zablokowania) pliku, po czym oczekują na wyrażenie przez program **A** zgody na wykorzystanie tego pliku. Zapewnia to programowi **A** możliwość odtworzenia plików uszkodzonych przez przerwaną transakcję, zanim inne programy użytkowe zaczną na nich działać.

Odtwarzanie bazy danych za pomocą dziennika starych wersji, przeprowadzane przez **A**, odbywa się w ten sposób, że najpierw program ten zwraca się do systemu 1022 z zapytaniem o liczbę rekordów w danym pliku oraz liczbę rekordów wskazywanych przez kartoteki odwrócone, związane z kolejnymi wartościami danego atrybutu (informacje o kolejnych wartościach atrybutu program **A** również uzyskuje przez zapytanie kierowane do systemu 1022). Jeżeli sumaryczna liczba rekordów wskazywanych przez kartoteki odwrócone nie zgadza się z liczbą wszystkich rekordów, to oznacza, że wewnętrzne struktury danych uległy uszkodzeniu i wtedy program **A** żąda od systemu 1022 zbudowania od nowa kartotek odwróconych. Następnie, odczytując wstecz kolejne zapisy starych wersji danych podstawowych, aktualizowanych przez przerwaną transakcję, program **A** generuje odpowiednie wywołania podprogramów systemu 1022 realizujące aktualizację bazy danych. W przypadku konieczności odbudowywania kartotek odwróconych, odtwarzanie bazy danych trwa długo, niemniej w czasie normalnej pracy przy tworzeniu dziennika częściowego oszczędza się wielu transmisji dyskowych.

W tabeli 1 podano porównanie liczby transmisji w przypadku programu testowego dodającego do bazy danych sześć rekordów, z których każdy miał trzy atrybuty, dla których istniały kartoteki odwrócone.

**Tabela 1. Porównanie techniki dzienników zupełnych z techniką dzienników częściowych**

| | Czas jednostki centralnej [S] | Liczba transmisji dyskowych | |
|---|---|---|---|
| | | zapis | odczyt |
| Dziennik zupełny (SZBD 1022) | 0.46 | 48 | 37 |
| Dziennik częściowy (system CRASH) | 0.46 | 22 | 24 |

## PRZYKŁAD ALGORYTMU ODTWARZANIA BAZY DANYCH

Przykładem jest algorytm zbudowany dla systemu CRASH. Algorytm ten wykorzystuje zapisywany w pamięci dyskowej dziennik, zawierający zarówno stare jak i nowe wersje danych podstawowych.

Przyjmijmy występowanie następujących trzech rodzajów aktualizacji bazy danych:

- zmiana wartości danych w istniejącym rekordzie
- dodawanie nowego rekordu
- wymazanie rekordu

Odpowiadają im następujące trzy typy rekordów wprowadzanych do dziennika:

- CHANGE (zmień)
- ADD (dodaj)
- DELETE (usuń).

Jeśli ilość informacji konieczna do opisania aktualizacji jednego rekordu z bazy danych przekracza maksymalny rozmiar rekordów, jakie mogą być wprowadzane do dziennika, stosuje się, następujące dodatkowe typy kontynuacyjne:

- CONT.CHANGE
- CONT.ADD
- CONT.DELTE.

Rekordy typu ADD zawierają nowe dane, typu DELETE — stare wersje danych, zaś CHANGE — zarówno stare, jak i nowe wersje.

Ponadto każdy rekord dziennika ma nagłówek zawierający:

- identyfikator transakcji
- identyfikator pliku
- identyfikator rekordu w ramach pliku — jest to unikalny identyfikator, który nigdy nie zostaje przypisany innemu rekordowi, nawet jeśli rekord oznaczony tym identyfikatorem zostanie usunięty z bazy danych
- dodatkowe informacje, np. czas i data zapisywania rekordu.

Poza już opisanymi, w dzienniku stosuje się jeszcze dwa następujące typy rekordów:

- EOT (koniec transakcji), wprowadzany do dziennika po zakończeniu każdej transakcji celem zmniejszenia liczby transmisji dyskowych, fakt rozpoczęcia transakcji odnotowuje się w pierwszym rekordzie dotyczącym danej transakcji.
- DB OK (baza danych w stanie spójności). Rekord ten umieszcza się w dzienniku nie częściej niż co 5 minut, w momencie gdy żadna transakcja nie jest przetwarzana. Pozwala to unikać czytania zbyt długiego odcinka dziennika przy odtwarzaniu bazy danych po awarii systemu.

W algorytmie zastosowanym w systemie CRASH można wyróżnić dwie części, które nazwano algorytmem przetwarzania dziennika w przód oraz algorytmem przetwarzania dziennika wstecz. Oba algorytmy zbudowane są w ten sposób, że dopuszczalne jest przerwanie w dowolnym miejscu i dowolną liczbę razy przetwarzania zarówno w przód, jak i wstecz. Dopuszczalne jest również dowolne przeplatanie przetwarzania w przód i wstecz, niezależnie od ewentualnego ich przerwania. Algorytm jest więc w pełni odporny na awarie systemu oraz błędy operatora.

Ponadto, celem zmniejszenia objętości dziennika oraz przyspieszenia jego przetwarzania, omawiany algorytm nie uwzględnia zasady odnotowywania w dzienniku wszystkich zmian, jakie wprowadza się do bazy danych, a więc również zmian wprowadzanych przy wycofywaniu skutków działania przerwanej transakcji. Zamiast tego, w dzienniku zapisanym w pamięci dyskowej, zaznacza się rekordy dotyczące wycofywanej transakcji jako nieważne. Jeżeli w czasie takiego unieważniania nastąpi awaria systemu, poprawiany rekord dziennika może ulec zniszczeniu. Omawiany algorytm gwarantuje poprawne odtwarzanie bazy danych nawet w takim przypadku. Jest to możliwe dzięki odpowiedniej kolejności działań (patrz algorytm przetwarzania wstecz) oraz dzięki temu, że każdy rekord umieszcza się w osobnym bloku, a więc awaria przerywająca zapisywanie bloku niszczy tylko jeden rekord dziennika.

Zauważmy, że w systemie CRAH nie warto było zastosować techniki opóźnionej aktualizacji przyspieszającej odtwarzanie, ponieważ technika dzienników częściowych i tak nie zapewnia szybkiego odtwarzania bazy danych.

### Algorytm przetwarzania dziennika w przód

Przetwarzanie dziennika w przód stosuje się w przypadku fizycznego uszkodzenia bazy danych w celu uaktualnienia jej starej kopii. Jeżeli fizyczne uszkodzenie bazy danych nastąpiło w czasie pracy systemu i spowodowało jej gwałtowne przerwanie, to po przetwarzaniu w przód nastąpić powinno przetwarzanie dziennika wstecz.

Przetwarzanie w przód polega na czytaniu kolejnych rekordów dziennika i wywoływaniu odpowiednich akcji SZBD 1022 w sposób przedstawiony w tabeli 2.

Tabela 2. Podstawowe akcje podejmowane przy przetwarzaniu dziennika w przód

| Typ rekordu w dzienniku | Działanie, którego żąda się od SZBD 1022 |
|---|---|
| CHANGE | zmień |
| CONT. CHANGE | zmień |
| ADD | dodaj |
| CONT. ADD | zmień |
| DELETE | wymaż |
| CONT. DELETE | — |
| EOT | — |
| DB OK | — |

Przy czytaniu dziennika w przód ignoruje się rekordy zaznaczone jako nieważne (rekord może zostać zaznaczony jako nieważny przy przetwarzaniu dziennika wstecz). Nie przeprowadza się również działań nakazanych w tabeli 2, jeżeli rekord bazy danych, na który wskazuje rekord w dzienniku, nie istnieje. Oznacza to bowiem, że aktualizowana obecnie kopia bazy danych była już poddana przetwarzaniu w przód, które spowodowało usunięcie poszukiwanego rekordu z bazy danych (aktualne przetwarzanie może być prowadzone błędnie, może też nastąpić awaryjne przerwanie poprzedniego przebiegu przetwarzania w przód).

Jeżeli na podstawie dziennika należy dodać nowy rekord do bazy danych, to dokonuje się tego tylko wtedy, gdy identyfikator rekordu, jaki ma być utworzony, jest większy od największego identyfikatora istniejącego w bazie danych. W przeciwnym przypadku nie podejmuje się żadnej akcji, ponieważ widocznie poprzednio przeprowadzane przetwarzanie w przód, zostało doprowadzone dalej i omawiany rekord został już do bazy danych wprowadzony. Zauważmy, że w tej chwili może go już nie być w bazie danych, ponieważ mógł on zostać wymazany w czasie poprzedniego przebiegu.

W przypadku, gdy na podstawie dziennika przy przetwarzaniu w przód trzeba dodać do bazy danych nowy rekord o identyfikatorze **n**, może się okazać, iż następny wolny identyfikator jest mniejszy od **n**. Należy wtedy dodać i usunąć odpowiednią liczbę rekordów pustych, ażeby rekord, który ma być wprowadzony do bazy danych na podstawie dziennika, otrzymał od SZBD odpowiedni identyfikator. Opisana sytuacja może nastąpić, jeżeli SZBD działa tak, że najpierw odnotowuje zwiększoną wartość następnego wolnego identyfikatora, a potem wprowadza nowy rekord do bazy danych.

Jeżeli awaria systemu nastąpi tuż przed wprowadzeniem rekordu do bazy danych, to po awarii algorytm przetwarzania wstecz, nie znalazłszy w bazie danych tego (najnowszego) rekordu, unieważni odpowiedni rekord dziennika. W rezultacie dziennik nie odzwierciedla faktu, że SZBD zwiększył o 1 wartość następnego wolnego identyfikatora; co w przypadku przetwarzania takiego dziennika w przód prowadzi do wystąpienia opisanej anomalii.

**Algorytm przetwarzania dziennika wstecz**

Przetwarzanie dziennika wstecz stosuje się w następujących przypadkach:

- **Awarii pojedynczego programu.** W tym przypadku sprawdza się i ewentualnie żąda odbudowania wewnętrznych struktur danych związanych z plikiem, który ostatnio był uaktualniany przez przerwaną transakcję. Następnie na podstawie rekordów dziennika, dotyczących przerwanej transakcji, wywołuje się odpowiednie akcje SZBD według reguł zebranych w tabeli 3. Czytanie dziennika wstecz przerywa się po znalezieniu rekordu oznaczającego początek transakcji.

- **Awarii systemu.** Ten przypadek jest zwielokrotnieniem przypadku pierwszego. Postępuje się tu analogicznie w stosunku do wszystkich transakcji przerwanych awarią systemu. W poszukiwaniu przerwanych transakcji dziennik odczytuje się aż do napotkania rekordu typu DB OK. Przy przetwarzaniu pomija się wszystkie rekordy dotyczące transakcji zakończonych, a więc takich, dla których w dzienniku znaleziono odpowiedni rekord typu EOT.

- **Cofania bazy danych do żądanego punktu kontrolnego** (rekordu typu DB OK). Cofać należy wszystkie zakończone i niezakończone transakcje aż do osiągnięcia rekordu DB OK (tego przypadku nie zrealizowano w omawianej wersji systemu CRASH).

Tabela 3. Główne akcje podejmowane w czasie przetwarzania dziennika wstecz

| Typ napotkanego w dzienniku rekordu oraz istnienie w BD rekordu wskazywanego przez ten rekord | Typ nowego rekordu tworzonego i zapisywanego do dziennika | Wymagana od SZBD akcja | Unieważnienie przetworzonego rekordu dziennika |
|---|---|---|---|
| 1. DELETE i CONT. DELETE | | | |
| — istnieje | ADD | dodaj | tak |
| — nie istnieje | CONT. ADD* | zmień | nie |
| — istnieje kopia | | odwołaj | nie |
| — usunięcie może być odwołane | | usunięcie | tak |
| 2. CHANGE i CONT. CHANGE | | | |
| — istnieje | CONT. ADD* | zmień | tak |
| — nie istnieje (musi wtedy istnieć kopia) | | zmień | nie |
| 3. ADD | | | |
| — istnieje | | usuń | tak |
| — nie istnieje | | | tak |
| 4. CONT. ADD | | | tak |

* wskazuje na identyfikator kopii

Akcje podane w tabeli 3 wykonywane są w następującym porządku:

1. Utworzenie nowego rekordu i zapisanie go na końcu dziennika.
2. Zażądanie właściwej akcji od SZBD.
3. Zaznaczenie unieważnienia przetworzonego rekordu dziennika.

Tworzenie nowego rekordu jest konieczne tylko wtedy, gdy nie da się odwołać usunięcia rekordu z bazy danych. Odwołanie takie jest możliwe w SZBD 1022 ponieważ system ten zaznacza jedynie, że dany rekord został usunięty, chociaż rekord ten fizycznie pozostaje w bazie danych. Odwołanie usunięcia rekordu staje się jednak niemożliwe po przeprowadzeniu reorganizacji bazy danych, podczas której usunięty rekord zostaje rzeczywiście wymazany z pamięci dyskowej. W takim przypadku przy przetwarzaniu wstecz trzeba w bazie danych utworzyć kopię usuniętego rekordu. Kopia taka ma jednak inny identyfikator niż rekord oryginalny. Jeżeli usunięty z bazy danych rekord (którego kopia została do niej wprowadzona) przed usunięciem podlegał aktualizacji, wtedy przy przetwarzaniu wstecz napotka się w dzienniku rekordy opisujące te aktualizacje. Rekordy te wskazują jednak na identyfikator, który obecnie nie istnieje już w bazie danych. Należy wtedy odnaleźć i odpowiednio zmodyfikować kopię nieistniejącego rekordu. Ażeby umożliwić odszukanie tej kopii, w systemie CRASH zastosowano następujące rozwiązanie: przy tworzeniu kopii, w nagłówku rekordu zapisywanego do dziennika, notuje się identyfikator rekordu, którego kopię poleca się wprowadzić do bazy danych.

W ten sposób system CRASH jest w stanie na podstawie dziennika starych wersji cofnąć bazę danych do dowolnego punktu, nawet jeżeli została ona zreorganizowana i wymazano z niej fizycznie wszystkie usunięte rekordy.

Z braku miejsca nie podano rozważań dotyczących poprawności przedstawionego algorytmu. Najłatwiej jest przeprowadzić nieformalny dowód poprawności, rozważając kolejno wszystkie możliwe przypadki, których wbrew pozorom jest tylko kilka.

* * *

Głównym celem artykułu było zwrócenie uwagi na możliwości techniki dzienników częściowych. Jak już stwierdzono — wadą tej techniki, oprócz wolniejszego odtwarzania, jest bardziej skomplikowany algorytm niż w przypadku techniki dzienników zupełnych. Natomiast główną zaletą jest możliwość odbudowania systemu odtwarzania bazy danych do SZBD bez konieczności zmieniania tego ostatniego. System CRASH jest przykładem takiego rozwiązania dla SZBD 1022 i jest wielce prawdopodobne, że dla wielu innych systemów zarządzania bazą danych można zastosować podobne rozwiązanie.

LITERATURA


[1] Bernstein P. A., Nathan Goodman N., Hadzilacos V.: Recovery Algorithms for Database Systems. Report Aiken Computation Laboratory, Harvard University, Ma, 1982

[2] Eswarn K. P., Gray J. N., Lorie R. A., Traiger I. L.: The Notions of Consistency and Predicate Locks in Database Systems. Comm. ACM, vol. 19, No. 11, str. 624—633, November 1976

[3] Gibbons T.: Integrity and Recovery in Computer Systems. NCC Publ., Manchester, Uk, 1976

[4] Gray J. N., Lorie R. A., Putzolu G. R., Traiger I. L.: Granularity of Locks and Degrees of Consistency in a Shared Data Base. Modeling in Data Base Management Systems, Ed. by G. M. Nijssen, North—Holland Publ. Co. 1976, pp. 365—394

[5] Lindsay B. G., et al.: Notes on Distributed Databases IBM Research Laboratory, San Jose, RJ 2571 33471 7/14/79

[6] Małuszyński J. T.: Analysis of Crash Recovery for EDP Systems Using 1022 DBMS. University of Pittsburgh Computer Center, 1980

[7] Małuszyński J. T.: CRASH Crash Recovery System for 1022 DBMS. Univ. of Pittsburgh Computer Center, 1981

[8] System 1022, User's Reference Manual, Software House Cambridge, MA, 1980

[9] Verhofsted J. S. M.: Recovery Techniques for Database Systems. Computing Surveys, vol. 10, str. 167195, No. 2, June 1978.



MIROSŁAW PAPRZYCKI
KRZYSZTOF URBANIEC
Warszawa


# Ujednolicenie komputerowego wspomagania prac inżynierskich


**Zastosowania komputerów w pracach inżynierskich są ostatnio w krajach wysoko uprzemysłowionych jedną z najszybciej rozwijających się dziedzin zastosowań informatyki. Początkowo, tj. w latach pięćdziesiątych, zastosowania te sprowadzały się do wykonywania skomplikowanych obliczeń technicznych. W latach sześćdziesiątych rozwinęły się komputerowe metody wspomagania projektowania (CAD). Od lat siedemdziesiątych powstają i są wdrażane do praktyki systemy obejmujące także wspomaganie produkcji i elementy zarządzania produkcją, a więc m.in. planowanie i kontrolę wytwarzania oraz informowanie kierownictwa przedsiębiorstw (CAD/CAM). Dąży się do tego, by wspomagać informatyką zintegrowane systemy produkcyjne — od projektowania konstrukcji wyrobów i procesów ich wytwarzania, po sterowanie produkcją i kontrolę produkcji.**


Elementem łączącym różne dziedziny jest wspólna baza danych, która — raz wprowadzona — jest wykorzystywana wielokrotnie w różnych fazach projektowania i realizacji produkcji. Stwarza to możliwości skracania czasu przygotowania produkcji oraz eliminacji błędów powstających na styku przygotowania i realizacji (np. błędów kreślarskich w rysunkach, błędów trasowania). Przy integracji projektowania z wytwarzaniem, kompleksowy system informatyczny i kompletna baza danych tworzą podstawę do projektowania z wariantowaniem i optymalizacją rozwiązań, także z punktu widzenia pracochłonności i kosztów wytwarzania.

## NA ŚWIECIE I W POLSCE

Najwyżej rozwinięto komputeryzację prac inżynierskich w Japonii i USA. W USA systemy CAD są stosowane w ok. 5% przedsiębiorstw, a prognozy wskazują, że liczba ta wzrośnie w roku 1985 do 20% [10]. Wiele innych krajów (w tym również ZSRR, NRD, CSRS i WRL) szybko rozwija systemy wspomagania prac inżynierskich [2, 3, 12]. Interesującym trendem jest to, że ostatnio największym użytkownikiem systemów CAD staje się przemysł elektromaszynowy, gdy w latach siedemdziesiątych był nim przemysł elektroniczny.

Doświadczenia wielu krajów wykazują niezbicie celowość nowoczesnego podejścia do systemów CAD. Niektóre źródła podają dane o trzykrotnym wzroście rentowności nakładów na projektowanie, zmniejszaniu liczby zatrudnionych kreślarzy nawet do 75% [10], kilkakrotnym skróceniu czasu projektowania oraz zwiększeniu dokładności i jakości opracowań projektowych. Dalsze efekty stosowania systemów CAD, ujawniające się w sferze produkcji i eksploatacji wyrobów, to m.in.: oszczędności materiałowe i zmniejszenie energochłonności dzięki optymalizacji projektów czy zmniejszenie kosztów uruchamiania produkcji.


Mgr MIROSŁAW PAPRZYCKI ukończył w 1969 r. Wydział Matematyki Uniwersytetu Warszawskiego. Pracę zawodową rozpoczął w Centrum Obliczeniowym PAN w 1968 r. prowadząc m.in. prace z zakresu taksonomii numerycznej oraz programowania liniowego. W 1973 r. podjął pracę w Ośrodku Badawczo-Rozwojowym Podstaw Technologii i Konstrukcji Maszyn, gdzie pełni funkcję Kierownika Zakładu Automatyzacji Prac Inżynierskich. W latach 1974—1980 był konsultantem ds. API w OBR Narzędzi. Był m.in. współautorem systemów FORTRAN-305 i FORTRAN-306 na minikomputery serii MERA 300 oraz systemu informacji patentowej PATENT.

Doc. dr hab. inż. KRZYSZTOF URBANIEC ukończył w 1964 r. Politechnikę Warszawską, gdzie również uzyskał stopień doktora (1970) i doktora habilitowanego (1978). W latach 1975—1982 w Przedsiębiorstwie CHEMADEX—WARSZAWA kierował pracownią projektową, specjalizującą się w zastosowaniach techniki obliczeniowej. Od 1982 r. jest Kierownikiem Zakładu i Pełnomocnikiem Dyrektora ds. Informatyki w Instytucie Organizacji Przemysłu Maszynowego. Przedstawiciel PRL w Radzie ds. Zastosowań Środków Techniki Obliczeniowej KM ds. ETO. Autor kilkudziesięciu publikacji, w tym książki i kilkunastu artykułów z zakresu inżynierskich zastosowań metod matematycznych i informatyki.

Z tendencjami światowymi jaskrawo kontrastuje obecna sytuacja w Polsce. Można oceniać, że kontakt z rozwojem światowym był utrzymywany jeszcze w połowie lat siedemdziesiątych, kiedy to w niektórych ośrodkach zrealizowano stosunkowo nowoczesne wspomaganie prac inżynierskich przy użyciu środków technicznych i programowych importowanych z krajów zachodnich [11]. Nowsze opracowania nie dotarły już do Polski wskutek zahamowania importu, a przy tym nie rozwinięto szerzej rodzimych prac, ani nie zorganizowano dostatecznej współpracy z krajami RWPG [8]. W efekcie polscy inżynierowie nie mają dziś dostępu do systemów komputerowego wspomagania, jakimi posługują się nasi konkurenci i na Zachodzie, i na Wschodzie. Dodać trzeba, że w placówkach naukowych i projektowych w Polsce śledzi się rozwój światowy, powstałe opóźnienia mogłyby więc być nadrobione przy skierowaniu na ten cel odpowiednich środków [1].

## BADANIA W POLSCE

W 1982 roku OBR TEKOMA przeprowadził metodę ankietową badania stanu zastosowań komputerów w projektowaniu inżynierskim w przemyśle elektromaszynowym [6]. Dodatkowo, IOPM przeprowadził analizę podobnych zastosowań komputerów w przemyśle budowy maszyn ciężkich i aparatury procesowej oraz w hutnictwie [9]. Analiza zebranych materiałów wykazuje, że najwięcej zastosowań dotyczy projektowania konstrukcyjnego. Różnorodność zastosowań jest bardzo duża — od stosunkowo prostych obliczeń części maszyn, do złożonych uniwersalnych systemów analiz wytrzymałościowych. We wszystkich badanych przypadkach komputer jest stosowany jedynie na niektórych etapach projektowania — żadne z zastosowań nie ma cech systemu kompleksowego wspomagania pracy inżyniera. Użytkowane oprogramowanie ma najczęściej charakter opracowania indywidualnego (na potrzeby danego przedsiębiorstwa) i stanowi komputerową realizację powszechnie stosowanych procedur obliczeniowych. Praktycznie nie korzysta się z jednolitych baz danych; z reguły dane są wprowadzane oddzielnie dla różnych programów lub systemów, stosowanych w różnych fazach projektowania.

Zebrane materiały dają także obraz braku wymiany oprogramowania między różnymi jednostkami — te same problemy techniczne są oprogramowane wiele razy. Częściowo wynika to z dużej różnorodności stosowanego sprzętu, z reguły zresztą nie dostosowanego do potrzeb inżynierskich (brak plotterów i grafoskopów). Dokumentacja oprogramowania oraz stosowane rozwiązania wprowadzania danych i wyprowadzania wyników mają z reguły charakter unikalny, co zmniejsza możliwości wykorzystania programów przez inne jednostki. Oprogramowanie jest wytwarzane metodami rzemieślniczymi (jeśli nie wręcz amatorskimi), co powoduje, że jest ono trudne do modyfikowania i kłopotliwe w konserwacji. I wreszcie — jednym z najbardziej pesymistycznych ustaleń są występujące w wielu ośrodkach trudności związane z brakiem zaufania i niechęcią do korzystania z wyników komputerowego przetwarzania wśród inżynierów.

## PODSTAWOWE BRAKI UJEDNOLICEŃ

Przedstawiony zarys stanu zastosowań komputerów w projektowaniu inżynierskim wykazuje istnienie szeregu problemów, których wspólnym elementem jest nadmierna różnorodność i dowolność stosowanych rozwiązań. Trudności zaczynają się już w fazie planowania zastosowań komputera. Wiedza o komputerowym wspomaganiu projektowania jest w Polsce słabo spopularyzowana. Brakuje kształcenia, w którym problematyka ta byłaby ujmowana kompleksowo — od określenia podstawowych pojęć, przez metody przygotowania zastosowań komputera (metodologia i zasady ogólne, budowa modeli matematycznych, projektowanie oprogramowania itp.), do praktycznego wykorzystania metod komputerowych.

Następna sprawa — brak warunków i stymulacji dla wymiany oprogramowania. Brak ochrony prawnej i powszednia niechęć do traktowania programów jako towaru odstręczają twórców od wytwarzania oprogramowania powielarnego. Właśnie dlatego chętnie projektuje się i opracowuje programy od początku, choć są one nieraz kopiami wdrożonych już opracowań.

Kwestią braku zaufania do wyników otrzymywanych z komputera można także sprowadzić do braku uporządkowania zasad wykorzystania programów. Projektant, odpowiedzialny wobec prawa za wynik swojej pracy, nie może ryzykować wykorzystania obliczeń których prawidłowości nie jest w stanie sprawdzić. Dotyczy to szczególnie tych dziedzin techniki, w których projektowanie musi uwzględniać przepisy bezpieczeństwa. W pewnych dziedzinach istnieją już rozwiązania tego problemu; np. w budowie statków — towarzystwa ubezpieczeniowe żądają wykonywania obliczeń wytrzymałościowych atestowanymi pakietami programowymi.

## WARUNKI WPROWADZANIA UJEDNOLICEŃ

Skuteczną drogą do likwidacji nieuzasadnionych różnorodności jest normalizacja. Jeśli jednak rozważac normalizację w obszarze komputerowego wspomagania prac inżynierskich, to już w fazie definiowania problematyki pojawiają się co najmniej dwie poważne przeszkody, świadczące o tym, że nie tylko problematyka jest trudna, lecz również warunki dla jej podejmowania są niesprzyjające.

Stan normalizacji w informatyce — poza sprawami sprzętowymi — jest bardzo skromny. Istnieje tylko kilka norm PN na podstawowe nazwy i symbole graficzne — na potrzeby dokumentowania prac. Istnieje ponadto kilkanaście norm BN, opracowanych w ślad za normami RWPG — na potrzeby dokuemntowania oprogramowania JS i SM EMC. Były także wprowadzane pewne normy zakładowe w ZETO Katowice. Jak z tego wynika, podstawy do podejmowania prac nad normalizacją systemów inżynierskich są bardzo niekompletne. Pewne nadzieej można wiązać z pracami Rady ds. Zastosowań Srodków Techniki Obliczeniowej, gdzie prowadzony jest rozpoznawczy temat dotyczący normalizacji systemów inżynierskich. Wydaje się również, że trzeba bardziej zdecydowanie sięgać po opracowania ISO, która to organizacja wydała już kilkadziesiąt norm z zakresu informatyki.

Drugi problem wynika ze stanu — jak można by to określić — infrastruktury prawno-normatywnej techniki w Polsce. Otóż kwestia zaufania do wyników uzyskiwanych z systemów inżynierskich oraz prawnego sankcjonowania tych systemów — mają ścisły związek z nadzorem nad bezpieczeństwem techniki w Polsce. Powołanych do tego wiele instytucji nadzorczych, z których najważniejszymi są: Urząd Dozoru Technicznego, GIGE, Państwowa Inspekcja Sanitarna, Komenda Główna Straży Pożarnych oraz Państwowa Inspekcja Pracy. Działają one niezależnie od siebie, a w sensie skutków społecznych — działalność ta nie jest uporządkowana, nie stanowi spójnego systemu [4]. Różnie rozumiana jest kompleksowość działań, rodzaje zagrożeń technicznych, rodzaje urządzeń będących przedmiotem nadzoru itp. Występują liczne zazębienia obszarów działalności, brakuje jednoznacznych kryteriów ustanawiania nadzoru. Stan ten nie pozwala na precyzyjne definiowanie i jednolite traktowanie zasad kontroli i sankcjonowania tych zastosowań systemów inżynierskich, w których pojawiają się problemy bezpieczeństwa techniki.

* * *

Pierwszoplanowym zadaniem w zakresie normalizacji systemów inżynierskich wydaje się obecnie opracowanie norm dotyczących słownictwa oraz warunków technicznych wykonania i odbioru oprogramowania wspomagającego projektowanie. Równocześnie potrzebne jest prowadzenie prac nad ujednoliceniem metod badań i oceny jakości, a także metod badań i oceny powielarności systemów informatycznych. Dla wykonawców systemów inżynierskich będą też potrzebne ujednolicenia metod projektowania i metod testowania oprogramowania. Ukoronowaniem tych prac powinno być przygotowanie zasad atestacji systemów inżynierskich. Powstałyby ponadto podstawy do szerszego stosowania komputerowego wspomagania przy tworzeniu oprogramowania [6, 7].

Podkreślić należy, że powyższe postulaty ogarniają tylko część zagadnień. Potrzebne są bowiem także działania w sferze organizacji, metod pracy i form dokumentacji, stosowanych w technicznym przygotowaniu produkcji. Działania te stworzyłyby pole dla zastosowań systemów inżynierskich i pozwoliłyby na sprawniejsze ich wdrażanie.

LITERATURA

[1] Bonkowicz-Sittauer S.: Problemy oprogramowania projektującego. Biuletyn Informacyjny — Obiektowe Systemy Komputerowe, nr 2/1981
[2] CAD — Computer Aided Design. Zeitschr., für Organisation, nr 4/182
[3] Fomin B. I., Zubkow J. S.: Organizacja rabot po awtomatizacii projektirovanija w elektromaszinostroitelnom objedinenii. Elektrotechnika, nr 12/1980
[4] Nizielski M.: Raport nt. stanu nadzoru nad bezpieczeństwem techniki w Polsce. SIMP, Warszawa, 1981
[5] Paprzycki M.: Zasady atestacji informatycznych systemów automatyzacji prac inżynierskich. Materiały V Konferencji nt. Komputerowego Wspomagania Projektowania, Radzyna, 1983
[6] Rzevski G.: On the Design of Engineering Software. Prac. Int. Conf. Engineering Software, Southampton, 1979
[7] Rzevski G., Wells M.: Computer-Aided Design of Engineering Software. Adv. Engineering Software, nr 1/1982
[8] Sikora J.: Działalność SARP w ramach Rady ds. Zastosowań Środków Techniki Obliczeniowej (mat. niepublikowany, 1983)
[9] Stan zastosowań informatyki w techniczno-organizacyjnym przygotowaniu produkcji w przemyśle budowy maszyn i hutnictwie. Opracowanie koordynowane przez IOPM: Cz. I — OBR TEKOMA, Cz. II — CHEMADEX, Cz. III — CIBEH (niepublikowane, 1982)
[10] The Impact of Electronics on the International Economic Setting — the Case of Computer-Aided-Design. UNIDO Working Paper IS.297, 1982
11 Urbaniec K.: Oprogramowanie systemów automatyzacji projektowania. Informatyka, nr 6/1975
[12] Vančura J., Novotny K.: System racionalnich metod konstruovani. Progres, nr 3/1981.

Przegląd języków wysokiego poziomu (3)

# Języki wysokiego poziomu a inżynieria i metodologia programowania

**W literaturze można znaleźć wiele materiałów poświęconych inżynierii i metodologii programowania. Większość z nich wiąże się z językami wysokiego poziomu.**

**Inżynieria programowania jest — według definicji autorki — zbiorem narzędzi i metod potrzebnych do wytwarzania oprogramowania skutecznego, poprawnego i niezawodnego, charakteryzującego się aktualnością i efektywnością we wszystkich fazach „cyklu życia". Spośród wielu aspektów autorka zajmuje się jedynie techniczną stroną zagadnienia, która bezpośrednio wiąże się z językiem programowania. W chwili obecnej istnieje mnóstwo definicji „cyklu życia programu", ale większość z nich wymienia następujące główne etapy takiego cyklu: definicja wymagań, określenie specyfikacji, projektowanie, kodowanie, testowanie, tworzenie dokumentacji, weryfikacja programu i konserwacja. Charakteryzując te etapy można stwierdzić, że dwa pierwsze — to problem definicji, następne trzy — to sprawa rozwiązania, zaś trzy ostatnie — to zapewnienie pełnej aktualności oprogramowania.**

*

**Określenie wymagań** związanych z rozwiązaniem jakiegoś problemu jest częściej zagadnieniem psychologicznym niż technicznym. Ciągle jeszcze muszą być prowadzone poszukiwania mające na celu rozwijanie sposobów definiowania wymagań. Najprościej byłoby polecić komputerowi: „sporządź listę płac" lub „znajdź najlepsze połączenie lotnicze z Nowego Jorku do Los Angeles". Jednak przekazanie tego rodzaju informacji nawet najbardziej „inteligentnym" komputerom nie spowodowałoby automatycznego znalezienia rozwiązania, z tej prostej przyczyny, że informacje te nie są wystarczające nawet dla człowieka.

W przypadku listy płac należałoby dodatkowo podać dane o każdym pracowniku, a także informacje, czy lista płac będzie sporządzana co tydzień, co pół miesiąca, czy co miesiąc, czy pracownicy otrzymują gotówkę czy też mają konta bankowe, jakie należy wykonać zestawienia itp.

Natomiast w przypadku połączenia lotniczego określenie „najlepsze" jest niejednoznaczne. Może ono bowiem odnosić się do połączenia najszybszego, najwygodniejszego dla pasażerów, najbardziej ekonomicznego z punktu widzenia zużycia paliwa itp. Są to wszystko szczegóły, które określają wymagania.

Można wyobrazić sobie idealną sytuację, w której dysponujemy tak mądrym systemem komputerowym, który po otrzymaniu wymagań w postaci dosyć ogólnej, postawi odpowiednie pytania zmierzające do zdobycia na ten temat dodatkowych informacji. Oczywiście jesteśmy jeszcze daleko od takiego ideału i nawet, jeśli otrzymamy odpowiedzi, które lepiej określą nam problem, to jednak ciągle jeszcze daleko nam do jego rozwiązania.

Przykładami języków umożliwiających opis wymagań są:

PSL/PSA, zaliczany do języków służących również do opisu specyfikacji (opisany w 1977 roku przez Teichroew'a i Hershey'a)
RSL (Requirements Specification Language) — Davis'a i Rauscher'a (1979)
AFFIRM — Musser'a (1979)
RDL — Heacox'a (1979).

*

Toczy się bardzo wiele dyskusji na temat różnic między **opisem wymagań i specyfikacjami.** Niektórzy, podając definicje „cyklu życia", łączą oba te zagadnienia. Autorka uważa, że różnica między nimi polega na tym, że specyfikacje powinny być bardziej szczegółowe niż opis wymagań, nawet jeśli te ostatnie są podane dosyć dokładnie. W przypadku wspomnianej listy płac wymagania można określić odpowiadając tylko na wszystkie postawione wyżej pytania, natomiast specyfikacje powinny zawierać znacznie więcej szczegółów i konkretnych ustaleń (np. opis formatu dnia i godziny dla danych wejściowych i wyjściowych, maksymalną liczbę pracowników, powiązania między danymi różnych pracowników, sposób ochrony danych).

W niektórych przypadkach granica między opisem wymagań i specyfikacjami nie jest zbyt oczywista. W publikacji Biggerstaff (1979) znajduje się tabela zawierająca wykaz ośmiu języków (systemów), używanych do opisu specyfikacji, spośród których najczęściej używane są: PSL/PSA oraz SPECIAL (Robinson i Roubine, 1977).

Godnym uwagi jest związek języków do opisu specyfikacji z projektowaniem programów i ich weryfikacją. Specyfikacje określają c o należy zrobić, natomiast projekt powinien pokazywać j a k należy to zrobić (uwzględniając oczywiście wszystkie ograniczenia). Aby projekt mógł dobrze odwzorowywać specyfikacje, należy właściwie rozumieć wszystkie związki między językami przeznaczonymi do opisu specyfikacji oraz do opisu projektu.

ciąg dalszy na str. 21

*Zaczynamy! Pierwsze wydanie mikroKLANU przybrało wreszcie materialną postać i zeszło na papier. Jest więc pierwsza możliwość konfrontacji oczekiwań z realizacją. W odróżnieniu od szerzącej się obecnie praktyki zwiastowania fanfarami każdej nowej iluzji, nie będziemy nikomu wmawiać — pod groźbą banicji — że oto właśnie zaspokoiliśmy wszystkie potrzeby „mikrospołeczności". Wprost przeciwnie — sądzimy, że każda publikacja będzie wykazywać jak wiele problemów pozostaje nie rozwiązanych. Chcemy jednak przełamać poczucie bezsilności i pokazać, że jednak dla chcącego... W tej dziedzinie naprawdę warto próbować — mikroinformatyka to przecież nie tylko gry telewizyjne, to także szansa na inną jakość życia. Dlatego, nie czekając beznadziejnie na „zielone światła", programy operacyjne itp. proponujemy działać „na własną rękę". My zaś będziemy pośredniczyć w wymianie doświadczeń, publikując:*

*— ciekawe rozwiązania konstrukcyjne*
*— użyteczne procedury programowe*
*— uwagi o eksploatacji i dodatkowych możliwościach mikrokomputerów*
*— opisy szczególnych właściwości popularnych układów cyfrowych*
*— informacje o sytuacji „rynkowej" — co, gdzie, za ile*
*— wiadomości o inicjatywach i wydarzeniach w sferze mikroinformatyki.*

*Będziemy uwzględniać różny stopień zaawansowania czytelników począwszy od tych co noszą się z zamiarem, poprzez tych co znoszą podzespoły, a skończywszy na tych, którzy wynieśli już wiele doświadczeń i obmyślają bardziej wyrafinowane rozwiązania. Forma wkładki do pisma powinna ułatwić przechowanie materiałów — do chwili, gdy właśnie okażą się potrzebne.*

*W mikroKLANIE będziemy zamieszczać teksty zwięzłe (do 4 stron maszynopisu). Nadsyłane rysunki (czytelne!) muszą faktycznie ułatwiać wykorzystanie pomysłu przez innych.*

*Redaktorem wkładki jest mgr inż. Andrzej J. Piotrowski, pracownik Przemysłowego Instytutu Elektroniki w Warszawie.*

*Redakcja*

● Firmy polonijne są w zasadzie jedynym dostawcą sprzętu mikrokomputerowego na krajowy rynek. Żądają jednak horendalnych cen, które blokują żywiołowość rozwoju mikroinformatyki. Pobieżne porównanie wolnorynkowych cen podzespołów z ceną gotowego wyrobu zdaje się świadczyć o niesamowitym zysku firm.

Prawda wygląda jednak inaczej. Stosowane w mikrokomputerach podzespoły w większości trzeba albo sprowadzać z zagranicy, albo — co ma miejsce najczęściej — nabywać od prywatnych importerów. Ci jednak nie wydają rachunków, które firma mogłaby przedstawić Urzędowi Podatkowemu. Tak więc zakupy odbywają się z pieniędzy wpisywanych w rubrykę: zysk firmy. Kwoty te są obciążone ok. 80% podatkiem. Aby nie dokładać do interesu, firma musi żądać za układ wmontowany w sprzedawane urządenie... 5 razy więcej.

Tak więc „umiejętnie" prowadzona polityka podatkowa powoduje, że mikrokomputer kosztuje zamiast 30 — 40 tys. złotych (też jeszcze drogo) — ok. 150 tys. Ponieważ prywatnych odbiorców rzadko stać na taki wydatek, głównymi odbiorcami są... instytucje finansowane przez państwo (przeważnie uczelnie). Pieniądze krążą więc w kółko, dotacje na naukę są odpowiednio wysokie, a biurokraci mają co robić: jedni dzielą fundusze, inni obliczają podatki — wszystko oczywiście bez pomocy mikrokomputerów, które są... za drogie.

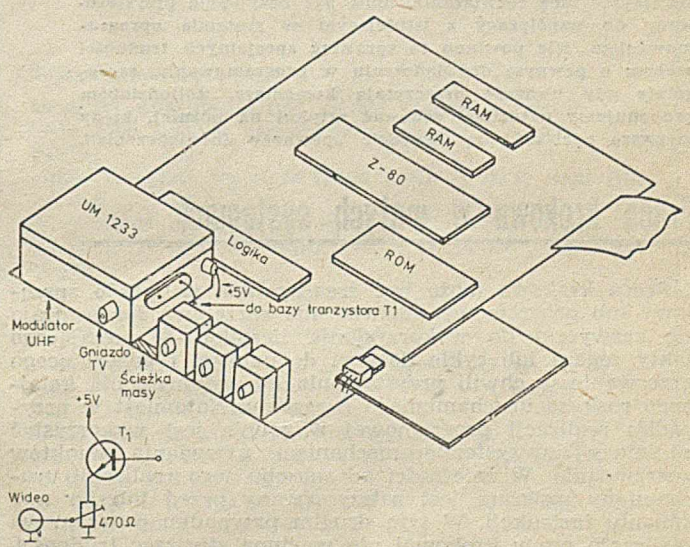
**Firma polonijna AMEPROD rozpętała rewolucję mikroinformatyczną w naszym kraju, oferując ZX81 za złotówki. Cena, może nazbyt wygórowana jak na komputer osobisty, nie odstrasza jednak instytucji. Odbiorców „za własne pieniądze" wspomaga prywatny import i tu cena jest już bardziej przystępna: ZX81 z pamięcią operacyjną o pojemności 16 KB jest sporo tańszy od... radzieckiego telewizora kolorowego (a o ileż bardziej pożyteczny). Tak więc krąg użytkowników ZX81 błyskawicznie się powiększa. Publikując kolejny tekst o tym mikrokomputerze (poprzednie w nr 5 i 9/83), tym razem opisujący nietypową możliwość zastosowania, być może przyczynimy się do „odkurzenia" istniejących sieci telewizji przewodowej.**

# ZX81

Dzięki niewielkiemu uzupełnieniu SINCLAIR ZX81 może współpracować z systemem telewizji przewodowej. Systemy takie są dość powszechne jako wyposażenie audio-wizualnych sal wykładowych. Możliwość podłączenia mikrokomputera do istniejącej sieci telewizji przewodowej znacznie rozszerza zakres zagadnień omawianych w trakcie zajęć dydaktycznych. Przykładowo — problemy optymalizacji czy też poszukiwania różnych rozwiązań są możliwe do obliczenia w czasie jednej jednostki lekcyjnej. Graficzne przedstawienie analizowanej funkcji przyspiesza przyswojenie wykładanego tematu. Możliwe jest również wykorzystanie ZX81 do przeprowadzania testów kontrolnych, sprawdzających wiedzę studentów.

Uzupełnienie układu mikrokomputera ma na celu wyprowadzenie niezmodulowanego, zespolonego sygnału wizyjnego. Sposób zrealizowania niezbędnej przeróbki przedstawiono na rysunku. Dodatkowo zamontowano tranzystor $T_1$, potencjometr montażowy $R_1$ oraz gniazdo koncentryczne $G_1$.

W trakcie użytkowania ZX81 w sali wyposażonej w wielkoekranowe monitory TV należy zwrócić uwagę na zakłócenia emitowane przez układy odchylające. Nie mają one wpływu na pracę samego mikrokomputera, mogą jednak uniemożliwić poprawne wczytanie programu z taśmy, zakłócając pracę magnetofonu. Najmniej wrażliwy na te zakłócenia spośród magnetofonów krajowych jest M 101.




**Szkic płytki drukowanej ZX81 z zaznaczonymi punktami podłączenia i układem dopasowującym**

Ma on także inne zalety predystynujące go do współpracy z ZX81:
— licznik taśmy, ułatwiający wyszukiwanie programów na kasecie
— ręczną regulację poziomu zapisu (można uniknąć przesterowania taśmy na początku nagrywanego programu)
— układ autostopu.

Informacja na taśmie magnetofonu jest kodowana za pomocą modulowanego sygnału o częstotliwości 2,5 kHz. Stan logiczny **0** jest reprezentowany przez cztery okresy przebiegu i 1,5 ms przerwy, a stan **1** przez osiem okresów i 1,5 ms przerwy. Szybkość zapisu informacji na taśmie wynosi 300 bitów/s.

Realizacja programów w języku BASIC-ZX81 jest stosunkowo wolna. W tabeli podano czasy wykonywania operacji w trybie FAST. SLOW jest przeciętnie czterokrotnie wolniejszy, ponieważ w trakcie wykonywania programu procesor obsługuje również monitor. Długi czas realizacji potęgowania skłania do korzystania z mnożenia, o ile tylko wykładnik potęgi jest liczbą całkowitą. Czas wykonywania niektórych funkcji i instrukcji, np. VAL i DIM, zależy od wartości argumentu. Czasy podane w tabeli są wielkościami przeciętnymi, uzyskanymi z serii pomiarów.

Pamięć operacyjna ZX81 jest wykorzystana w sposób bardzo racjonalny. Nie istnieje sztywny podział na pamięć programu, danych itp. Słowo kluczowe instrukcji zajmuje tylko jeden bajt pamięci. Nieco rozrzutnie zapisywane są stałe numeryczne w tekście programu. Zajmują one tyle bajtów, ile cyfr ma dana liczba, plus 6 bajtów, w których liczba ta zapisana jest w kodzie maszynowym. Dla zmniejszenia obszaru zajmowanego przez program wskazane jest wykorzystanie zmiennych o zadanej wartości. Przynosi to oszczędność pamięci już przy trzykrotnym występowaniu tej samej stałej w programie. Dodatkowa oszczędność pamięci programu jest możliwa poprzez wykorzystanie podprogramów napisanych w kodzie maszynowym mikroprocesora Z80.

| Operacja | Czas realizacji (ms) | Operacja | Czas realizacji (ms) |
|---|---|---|---|
| dodawanie | 1,25 | CODE | 1,25 |
| odejmowanie | 1,25 | CHR$ | 1,25 |
| mnożenie | 2 | PEEK | 1,25 |
| dzielenie | 3 | LET | 2 |
| potęgowanie | 109 | (zmienna numeryczna) | |
| pierwiastkowanie | 109 | | |
| sinus | 39 | GOTO | 1,5 |
| cosinus | 42 | IF/THEN | 2,8 |
| tangens | 84 | GOSUB/RETURN | 2,3 |
| arc sin | 174 | FOR/NEXT | 4,3 |
| arc cos | 174 | PRINT | 140 |
| arc tg | 59 | (zmienna numeryczna) | |
| LN | 66,5 | | |
| exp | 39 | PRINT | 1,4 |
| INT | 1,25 | (zmienna tekstowa) | |
| SGN | 0,5 | POKE | 2,3 |
| ABS | 0,5 | PLOT | 6,3 |
| RND | 11,5 | UNPLOT | 6,3 |
| VAL | 5 | DIM | 3,3 |
| STR$ | 112 | LET | 15 |
| LEN | 1,25 | (zmienna tekstowa) | |

**ANDRZEJ KAŁUŻA**
**SŁAWOMIR LITWIŃSKI**
**Politechnika Częstochowska**

---

**Napisanie i zasemblowanie programu to bynajmniej nie koniec pracy programisty. Trzeba jeszcze wygonić wszystkie parszywe pluskwy (ang. debug), które złośliwie namawiają Nasz Wspaniały Program do zupełnie innego funkcjonowania niż oczekujemy. Na pluskwy najlepsze jest DDT (ang. Designer Development Tool). Nie każdy jednak posiada ten środek pod ręką; natomiast znacznie łatwiej o pułapkę (nazywaną dyplomatycznie „punktem wstrzymania"). Aby sito, którym wyławiamy pluskwy było wystarczająco gęste, należałoby w zasadzie co krok zastawiać pułapkę. Stąd też zrodził się pomysł krokowej realizacji programu. Istota ułatwienia polega na automatycznym ustawianiu pułapki za każdą wykonywaną instrukcją, co oszczędza odcisków na palcach od ciągłego wprowadzania kolejnych punktów wstrzymania programu. Poniższy program, przedstawiający ideę rozwiązania, musi być oczywiście przystosowany do współpracy z istniejącym w systemie oprogramowaniem. Nie powinno to sprawiać specjalnych trudności osobom o pewnym doświadczeniu w programowaniu, szczególnie gdy uważnie przeczytają komentarz. Żółtodziobom proponujemy natomiast schować artykuł na później, kiedy pierwsze odciski będą dowodzić „pewnego doświadczenia".**

## Praca krokowa w małych systemach

Praca krokowa może być zrealizowana w sposób sprzętowy lub programowy. Rozwiązania sprzętowe sprowadzają się zazwyczaj do wykorzystania licznika — zliczającego takty zegara lub cykle dostępu do pamięci i generującego przerwanie w chwili przystąpienia do wykonywania kolejnego rozkazu uruchamianego programu. Natomiast w przypadku realizacji programowej wygodnie jest wykorzystać istniejący w systemie mechanizm ustawiania punktów wstrzymania. W zależności od sposobu jego realizacji uruchamiany program jest zatrzymywany przed lub po wykonaniu instrukcji. W tym drugim przypadku programowa realizacja pracy krokowej nie powinna stwarzać trudności.

Poniżej przedstawiono sposób realizacji pracy krokowej z wykorzystaniem punktów wstrzymania, zatrzymujących uruchamiany program przed wykonaniem wskazanej instrukcji i zabezpieczających zachowanie niezmienionego stanu środowiska programu. Zaprezentowany algorytm może zostać zrealizowany dla różnych mikroprocesorów. W przypadku 8080 uzyskano kod wynikowy o niewielkiej długości (ok. stu bajtów), co umożliwia wykorzystanie go w niedużych systemach mikroprocesorowych o ograniczonej pamięci stałej.

Prezentowany program można łatwo zmodyfikować dla uzyskania dodatkowych możliwości, takich jak: wykonanie określonej liczby instrukcji w jednym kroku, wykonanie programu z zachowanym śladem określonej liczby ostatnio wykonywanych instrukcji itp.

Wykorzystywane w programie punkty wstrzymania powinny być ustawione na adres instrukcji, do której może być przekazane sterowanie po wykonaniu kroku. Ponieważ instrukcje warunkowe umożliwiają rozejście na dwie drogi, wykorzystywane są dwa punkty wstrzymania. Jeżeli do dyspozycji jest tylko jeden punkt wstrzymania, to należy badać rodzaj warunku i sprawdzać jego spełnienie w celu wyboru właściwej drogi. Punkty wstrzymania muszą być chwilowe, tzn. powinny być usuwane po zatrzymaniu programu na jednym z nich. Zestaw instrukcji 8080 zawiera dziewięć różnych możliwości przekazywania sterowania po wykonaniu dyrektywy (tab).

| Typ instrukcji | Pierwszy możliwy adres następnej instrukcji | Drugi możliwy adres następnej instrukcji |
|---|---|---|
| Jednobajtowe (bez PCHL, RET, RST) | PC+1 | — |
| Dwubajtowe | PC+2 | — |
| Trzybajtowe (bez JMP i CALL) | PC+3 | — |
| JMP i CALL bezwarunkowe | (PC+1) | — |
| JMP i CALL warunkowe | (PC+1) | PC+3 |
| PCHL | HL | — |
| RET bezwarunkowe | (SP) | — |
| RET warunkowe | (SP) | PC+1 |
| RST | (PC) AND 38H | — |

Niektóre możliwości przekazywania sterowania, podane w tabeli, mogą być obsługiwane w identyczny sposób, bowiem ustawione dodatkowo i nie wykorzystane punkty wstrzymania po wykonaniu kroku zostaną usunięte. Pozwala to na połączenie kilku pozycji wymienionych w tabeli w jedną grupę instrukcji. Klasyfikację kodu instrukcji do jednej z tych grup można zrealizować w różny sposób. Metoda zastosowana w przedstawianym przykładzie polega na porównywaniu kodu rozkazu (po wyzerowaniu pól argumentów) z kodami przechowywanymi w tablicy. Długość tablicy można znacznie zmniejszyć łącząc kody różniące się jednym bitem we wspólną pozycję tablicy. Np. instrukcję IN o kodzie 0DBH i instrukcję OUT o kodzie 0D3H można badać łącznie uznając bit różniący te kody za bit argumentu. Z tablicy można także wyłączyć najbardziej liczną z grup, gdyż po zbadaniu, że kod nie należy do żadnej z pozostałych grup, można go bez dalszego badania zaklasyfikować do grupy nie objętej tablicą.

Każda pozycja tablicy zawiera maskę, przez którą instrukcja jest mnożona logicznie, oraz wzorzec, z którym jest porównywana. W rejestrze b ustawiany jest numer pozycji tablicy, umożliwiający wybór odpowiedniego sposobu ustawienia punktów wstrzymania. Adres punktu wstrzymania jest dostępny w parze rejestrów HL.

```
1: ;        REALIZACJA PRACY KROKOWEJ DLA MIKROPROCESORA 8080
2: ;
3: ;        ZMIENNE ZEWNETRZNE: PCBANK — ZACHOWANY PC PROGRAMU
4: ;                            SPBANK — ZACHOWANY SP PROGRAMU
5: ;                            HLBANK — ZACHOWANY HL PROGRAMU
6: ;        PUNKT WEJSCIA DO STARTU PROGRAMU — GO
7: ;
8: SSTEP:   LHLD PCBANK     ; POBRANIE KODU ROZKAZU DO C
9:          MOV  C, M
10:         MVI  B, 11      ; PORÓWNANIE Z TABELA KODÓW
11:         LXI  H, TABCOD
12: DECOD:  MOV  A, C       ; MNOŻENIE KODU PRZEZ MASKĘ
13:         ANA  M
14:         INX  H          ; PORÓWNANIE Z KODEM TABELI
15:         CMP  M
16:         JZ   RSTN       ; SKOK JEŻELI ROZPOZNANY KOD
17:         DCR  B
18:         INX  H
19:         JNZ  DECOD      ; SKOK JEŻELI NIE KONIEC TABELI
20: ;
21: ;       KOD INSTRUKCJI JEDNOBAJTOWEJ
22: ;
23:         LHLD PCBANK
24:         INX  H
25: ; ***   USTAW PUNKT WSTRZYMANIA (HL = PC+1)
26:         LHLD SPBANK
27:         MOV  E, M
28:         INX  H
29:         MOV  D, M
30:         XCHG
31:         JMP  SETBP      ; HL = (SP)
32: ;
33: ;       INSTRUKCJA RST
34: ;
35: RSTN:   DCR  B
36:         JNZ  PCHLO      ; SKOK JEŻELI NIE RST N ;
37:         MOV  A, C
38:         ANI  38H
39:         MVI  H, Ø
40:         MOV  L, A
41:         JMP  SETBP      ; HL = N*8
42: ;
43: ;       INSTRUKCJA PCHL
44: ;
45: PCHLO:  DCR  B
46:         JNZ  INSB2      ; SKOK JEŻELI NIE PCHL
47:         LHLD HLBANK
48:         JMP  SETBP      ; HL = HL
49: ;
50: ;       INSTRUKCJE DWUBAJTOWE
51: ;
52: INSB2:  LHLD PCBANK
53:         MOV  A, B
54:         SUI  4
55:         JNC  INSB3      ; SKOK JEŻELI NIE DWUBAJTOWA INSTR.
56:         INX  H
57:         INX  H
58:         JMP  SETBP      ; HL = PC+2
59: ;
60: ;       INSTRUKCJE TRZYBAJTOWE
61: ;
62: INSB3:  INX  H
63:         MOV  E, M
64:         INX  H
65:         MOV  D, M
66:         INX  H
67:         JZ   SETBP      ; HL = PC+3
68:         PUSH H
69:         XCHG
70: ; ***   USTAW PUNKT WSTRZYMANIA HL = (PC+1)
71:         POP  H
72: SETBP:  ; *** USTAW PUNKT WSTRZYMANIA
73:         JMP  GO         ; START URUCHAMIANEGO PROGRAMU
74: TABCOD: DB   ØFFH, ØC3H ; JMP, B = 11
75:         DB   ØFFH, ØCDH ; CALL, B = 1Ø
76:         DB   ØC7H, ØC4H ; C, B = 9
77:         DB   ØC7H, ØC2H ; J, B = 8
78:         DB   ØCFH, ØØ1H ; LXI, B = 7
79:         DB   ØE7H, Ø22H ; STA, LDA, LHLD, SHLD, B = 6
80:         DB   ØC7H, ØØ6H ; MVI, B = 5
81:         DB   ØC7H, ØC6H ; ADI, ACI, SUI, SBI,
82:                         ; ORI, CPI, ANI, XRI, B = 4
83:         DB   ØF7H, ØD3H ; IN, OUT, B = 3
84:         DB   ØFFH, ØE9H ; PCHL, B = 2
85:         DB   ØC7H, ØC7H ; RST, B = 1
86:         END
```

**IRENEUSZ MYZIK**
**Przemysłowy Instytut Elektroniki**
**Warszawa**

## *Ceny • Ceny • Ceny • Ceny • Ceny*

**listopad—grudzień 1983**

**Mikrokomputery:**

**RFN:** ZX SPECTRUM (16K) — 163 $; (48K) — 201 $
**USA:** ZX81 (1K) (do samodzielnego montażu) — 30 $
**Polska:**
**Firma UNITEX:** ZX SPECTRUM (16K) — 320 $; ZX81 (1K) — 146 $
Z ogłoszenia (ceny wywoławcze): ZX81 (16K) — 90 tys. zł; ZX SPECTRUM (16K) — 180 tys. zł; DRAGON — 500 „zielonych"

**Podzespoły na Perskim Jarmarku:**

zestaw8080, 8224, 8228 — 3,2 tys. zł; 8085 — 4,5 tys. zł
Z80A — 4 tys. zł; Z80B — 5 tys. zł; 6800 — 4 tys. zł
2708 — 1 tys. zł; 2716 — 2,5 tys. zł
8x4116 — 15 tys. zł; 8x4164 — 60 tys. zł
8251 — 1,2 tys. zł; 8255 — 2 tys. zł; 8279 — 4 tys. zł; 8257 — 4,5 tys. zł; 8755 — 4,5 tys. zł;
Z80 DMA — 3,5 tys. zł; Z80 PIO — 3,5 tys. zł; Z80 SIO/2 — 3,5 tys. zł.

**Żaden mikrokomputer nie może obejść się bez pamięci RAM (ang. Random Access Memory). Jest to jednak jeden z najdroższych składników mikrokomputera. Dlatego też, oprócz wymyślania coraz to doskonalszych mikrokomputerów, „jajogłowi" poświęcają wiele swego cennego czasu właśnie pamięciom.**

**Pamięci dynamiczne są rozwiązaniem pozwalającym stosunkowo małym kosztem upchnąć sporo informacji w jednej „kostce" (są już pamięci o pojemności 256 KB). Róża — jednak nie bez kolców. Zasadniczą wadą DRAM-ów (ang. Dynamic Random Access Memory) jest konieczność regularnego odświeżania zawartości pamięci. Niektóre firmy oferują specjalizowane układy „załatwiające" problem współpracy procesora z pamięciami dynamicznymi (np. INTEL 8202). Jednak cena 8202, przewyższająca kilkakrotnie cenę CPU, skutecznie eliminuje stosowanie tego układu w kraju. Układy innych firm też nie są tanie.**

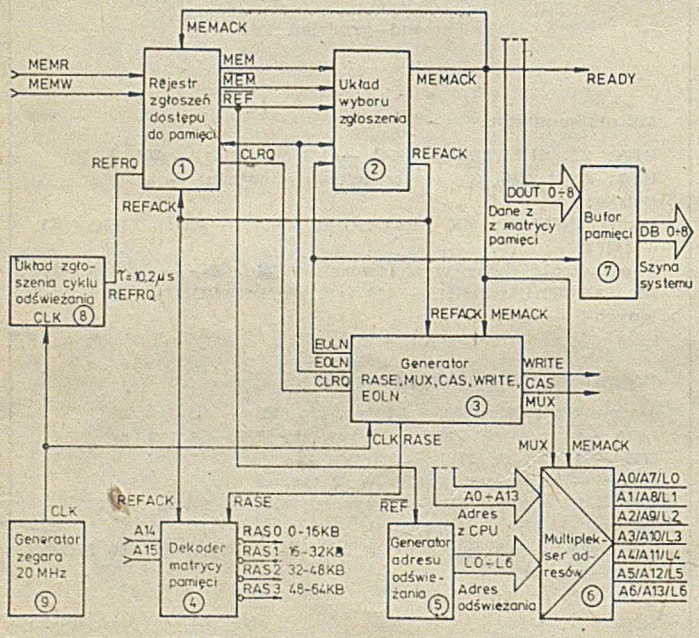
**Zamieszczony poniżej opis konstrukcji zakłada, że czytelnik posiada już pewien zasób informacji o DRAM-ach. Co będzie, jeśli okaże się, że zbyt wysoko ustawiliśmy poprzeczkę — trudno, będziemy wspólnie narzekać na brak literatury fachowej! Ponieważ CEMI zapowiedziało wprowadzenie do produkcji w 1983 roku pamięci dynamicznej 16 KB (którego to dzisiaj mamy...?), chyba niebawem każdy szanujący się mikrokomputer „dostanie" swoje 64 KB.**

**A zatem jak zrealizować:**

## Sterownik pamięci dynamicznych

Schemat blokowy przykładowego sterownika pamięci dynamicznej zbudowanego z układów TTL przedsta;wiono na rysunku 1. W skład sterownika wchodzą następujące bloki funkcjonalne:

1) rejestr zgłoszeń dostępu do pamięci
2) układ wyboru zgłoszenia
3) układ generatora strobów: RASE, MUX, CAS, WRITE, EOLN (ang. End of line)
4) dekoder bloków matrycy pamięci dynamicznej
5) generator adresu odświeżania
6) multiplekser adresów
7) bufor pamięci
8) układ zgłoszenia cyklu odświeżającego
9) generator zegara 20 MHz.




**Rys. 1. Schemat blokowy interfejsu pamięci dynamicznej**

Generator zegara został zbudowany przy użyciu układu 74S04 i dostarcza impulsy zegarowe o częstotliwości 20 MHz, które umożliwiają generowanie sygnałów RASE, MUX, CAS, WRITE, EOLN oraz sygnału zgłoszenia cyklu odświeżania REFRQ (ang. Refresh Request). Schemat ideowy generatora zegara przedstawiony jest na rysunku 3 (blok 9). Na tym samym rysunku przedstawiono układ zgłoszenia cyklu odświeżającego (blok 8). Proponowane rozwiązanie zapewnia odświeżanie jednego wiersza pamięci co 10,2 µs, czyli każdy bit pamięci odświeżany jest co 1,3 ms.

Układ zgłoszenia cyklu odświeżania składa się z dwóch liczników 74LS193. Impulsy zegarowe CLK o częstotliwości 20 MHz dzielone są wstępnie przez 16 (pierwszy licznik — U4). Wyjście TCU (ang. Terminal Count Up) z pierwszego licznika doprowadzone jest do wejścia zegarowego CLK drugiego licznika (U4). Na wyjściu TCU licznika U3 pojawia się impuls co 256 taktów zegara. Impuls ten wykorzystywany jest jako sygnał REFRQ.

Sterownik pamięci dynamicznej może znajdować się w jednym z trzech stanów.

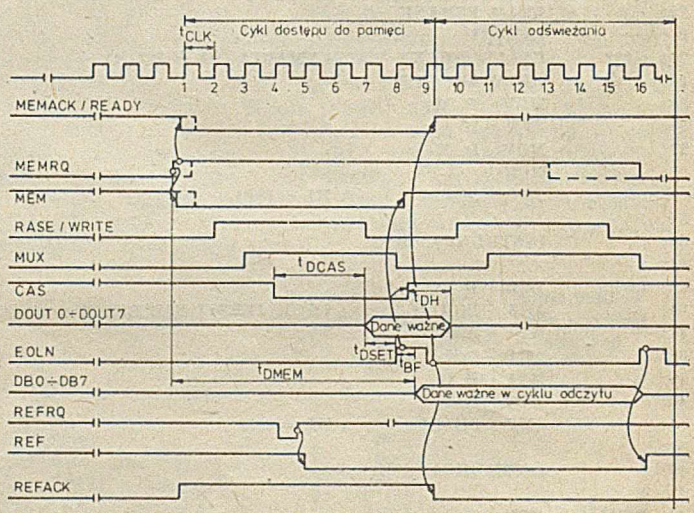
— cyklu zapisu/odczytu pamięci przez CPU
— cyklu odświeżania zawartości pamięci
— cyklu oczekiwania na zgłoszenie cyklu 1 lub 2.

Rejestr zgłoszenia dostępu (blok 1), przechowuje zgłoszenia MEMRQ (ang. Memory Request) oraz REFRQ (ang. Refresh Request). Układ wyboru zgłoszenia (blok 2), decyduje o kolejności obsługi żądań zapamiętanych w rejestrze zgłoszenia dostępu.

Żądania MEM (ang. Memory) i REF (ang. Refresh) obsługiwane są w kolejności zgłoszenia, zgodnie z zasadą FIFS (ng. First In First Served). W przypadku zgłoszenia któregokolwiek z żądań podczas obsługi poprzedniego, jego obsługa jest zawieszona do momentu zakończenia aktualnie wykonywanego cyklu.

Sygnał $\overline{\text{READY}}=0$ generowany jest tylko wtedy, gdy sterownik przejdzie do wykonywania cyklu zapisu/odczytu z pamięci tzn. $\overline{\text{MEMACK}}=0$ (ang. Memory Acknowledge). Każdy cykl zakończony jest anulowaniem odpowiedniego żądania w rejestrze zgłoszenia dostępu. W cyklu oczekiwania układ wyboru zgłoszenia generuje sygnały $\overline{\text{MEMACK}}=1$ $\overline{\text{REFACK}}=0$.

Sterownik pamięci dynamicznej rozpoczyna generowanie sygnałów strobujących do matrycy pamięci w przypadku, gdy sygnał CLRQ=1 (ang. Cycle Request). Kolejność po-




**Rys. 2. Przebiegi czasowe w układzie interfejsu pamięci DRAM w systemie µP**
**($t_{CLK}$ — 50ns; $t_{DSET}$ — min 20ns; $t_{DCAS}$ — max 180ns; $t_{DMEM}$ — min 350ns, max 403ns; $t_{BF}$ — max 28ns; $t_{DH}$ — min 0, max 40ns)**

jawiania się sygnałów strobujących oraz zależności czasowe między nimi obrazuje rysunek 2. Schemat ideowy generatora strobów (blok 3) przedstawia zaś rysunek 3.

Sygnał RASE umożliwia generowanie strobów RAS0÷÷RAS3 w sposób następujący:

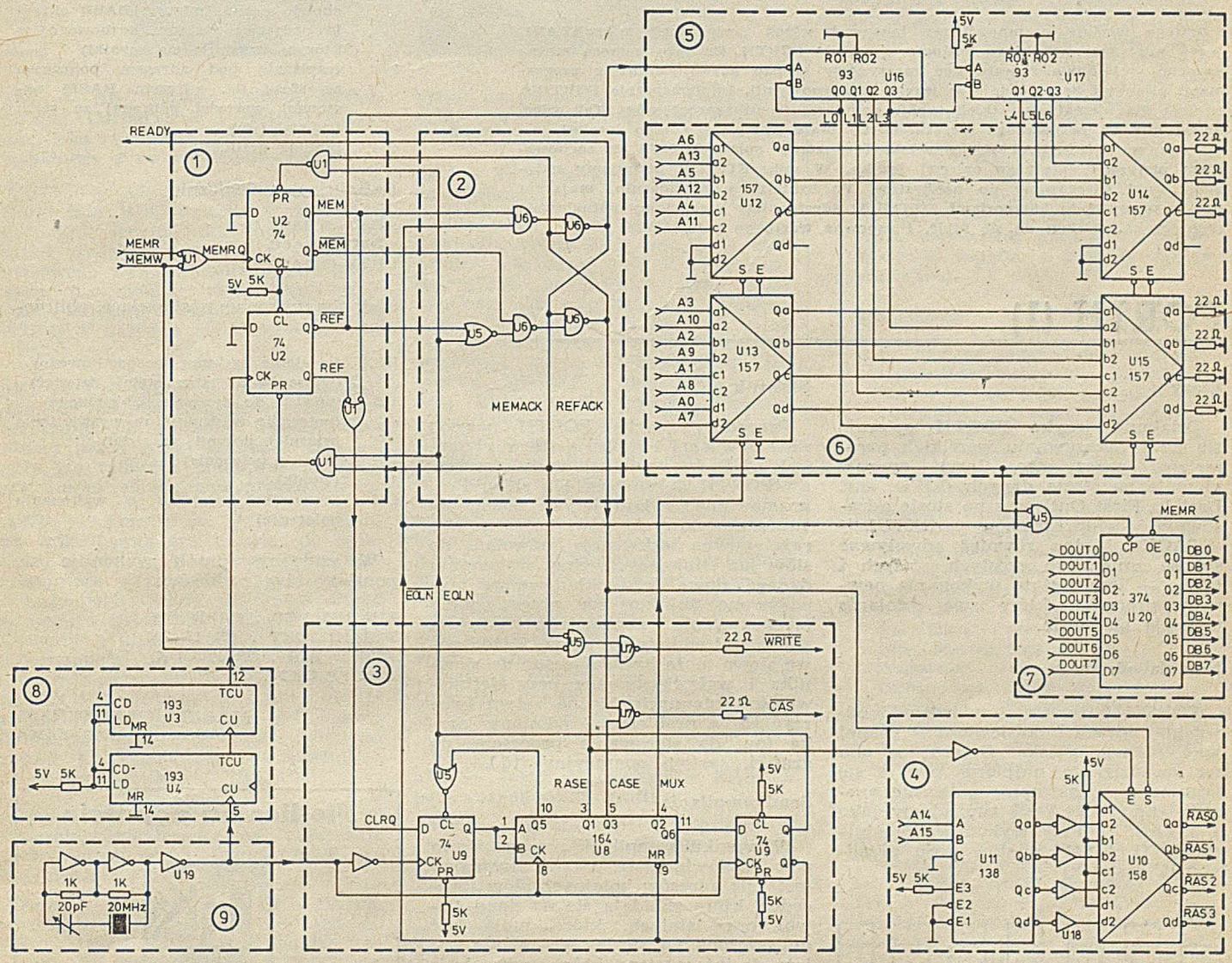
| RASE | REF/ ACK | A15 | A14 | RAS3 | RAS2 | RAS1 | RAS0 |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 0 | X | X | 0 | 0 | 0 | 0 |
| 0 | X | X | X | 1 | 1 | 1 | 1 |

Sygnał $\overline{CAS}$=0 (blok 3) generowany jest tylko w cyklu zapisu/odczytu pamięci przez procesor, jednocześnie dla całej matrycy. Opadające zbocze sygnału $\overline{CAS}$ powoduje:
— w cyklu zapisu — wpisanie informacji z szyny danych systemu do pamięci dynamicznej (jednocześnie sygnał $\overline{WRITE}$=0)
— w cyklu odczytu — podanie na szynę systemu danych z matrycy pamięci dynamicznej; aby dane zostały poprawnie zapisane do bufora musi to nastąpić przed upływem 180 ns od opadającego zbocza sygnału CAS.

Sygnał MUX (ang. Multiplexer) służy do przełączania adresu podawanego przez procesor, w sposób następujący:

| MUX | Wyjścia Q multiplekserów U12, U13 | | | | | | |
|---|---|---|---|---|---|---|---|
| | Q6 | Q5 | Q4 | Q3 | Q2 | Q1 | Q0 |
| 0 | A6 | A5 | A4 | A3 | A2 | A1 | A0 |
| 1 | A13 | A12 | A12 | A10 | A9 | A8 | A7 |

Natomiast w cyklu odświeżania — sygnał $\overline{MEMACK}$=1, co powoduje że na matrycę pamięci podawany jest — przez multipleksery U14 i U15 — adres z liczników U16 i U17. Liczniki U16, U17 generują adres wierszy matrycy. Liczba wierszy wynosi 128. Nowy stan liczników ustalany jest przez opadające zbocze sygnału $\overline{REF}$. Wyjście adresowe multiplekserów U14, U15 doprowadzone są do matrycy pamięci przez oporniki 33Ω w celu zmniejszenia przesłuchów.




Ryc. 3. Schemat ideowy
(7400 — U1, U6; 7402 — U5; 7404 — U18, U19; 1/2 7437 — U7; 7474 — U2, U9; 7493 — U16, U17; 74138 — U11; 74157 — U12, U13, U14, U15; 74158 — U10; 74164 — U8; 74193 — U3, U4; 74374 — U20)

Schemat ideowy układu dekodera matrycy pamięci, multipleksera adresów, generatora adresu odświeżania przedstawiono na rysunku 3.

Bufor pamięci służy do pamiętania danych z matrycy pamięci podczas cyklu odczytu. Wpis do bufora dokonywany jest, gdy EOLN=1 i MEMACK=0. Natomiast sygnał EOLN = 0 powoduje:
— skasowanie żądania aktualnie wykonywanego cyklu w rejestrze zgłoszenia dostępu
— wyzerowanie rejestru przesuwnego 74174 (U8) —
— wycofanie wszystkich sygnałów strobujących do pamięci dynamicznej
— ustawienie sygnału READY=1, gdy wykonywany był cykl dostępu do pamięci przez procesor.

Zastosowanie bufora pośredniczącego pomiędzy szyną danych systemu a szyną DOUT pamięci pozwala na utrzymywanie danych podczas odczytu przez dowolnie długi okres. Dzięki temu pamięć dynamiczna widziana jest przez procesor jak zwykła pamięć statyczna.

Schemat ideowy bufora pamięci przedstawia rysunek 3 (blok 7). Zaproponowane rozwiązanie można uprościć stosując zamiast układów 1, 2, 3 automat synchroniczny z pamięcią PROM.

Uwaga: Nie należy stosować układu 8238 jako sterownika systemowego, ponieważ generuje on wyprzedzony sygnał zapisu do pamięci, co może prowadzić do błędów podczas zapisu informacji do pamięci przez procesor.

**PIOTR SMÓLSKI**
Przemysłowy Instytut Elektroniki
Warszawa

---

**Programowanie w języku maszynowym jest żmudne. Trzeba nie lada wytrwałości i samozaparcia, aby napisać dłuższy program. Asembler rzadko jest wykorzystywany nawet przez profesjonalistów.**

**Do programowania mikrokomputerów najczęściej wykorzystywany jest BASIC. Jest to język tak prosty, iże osoba znająca angielski może opanować podstawowe zasady w ciągu zaledwie kilku godzin. Jednak po krótkim okresie fascynacji, użytkownika mikrokomputera zaczyna irytować długi czas oczekiwania na wynik realizacji programu. Coraz częściej okazuje się też, że nowo wymyślony program w żaden sposób nie daje się „upchnąć" w pamięci mikrokomputera...**

**Dlatego właśnie postanowiliśmy lansować wśród „członków" mikroKLANU język względnie wysokiego poziomu o nazwie FORTH. Za jego pomocą można uzyskać wyjątkowo zwięzły kod maszynowy (często porównywalny z programami pisanymi bezpośrednio w języku maszynowym!). Implementacje FORTH'A istnieją dla niemal wszystkich typów popularnych mikroprocesorów. Tak więc, niezależnie od posiadanego sprzętu, każdy może korzystać z tego języka!**

**Niebawem INFORMATYKA przedstawi obszerny opis FORTH'A, z zachowaniem pietyzmu należnego opisowi języka. W mikroKLANIE planujemy natomiast cykl adresowany do hobbystów. Po krótkim wprowadzeniu, mającym pomóc w myśleniu kategoriami FORTH'A, udostępnimy nasze łamy entuzjastom tego języka. FORTH da się lubić. I zapewne warto go polubić!**

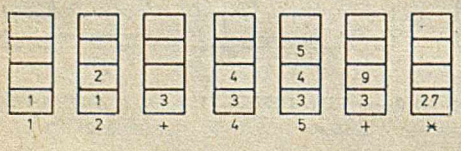
# FORTH (1)

### Stosy

Działania języka FORTH opierają się na przekazywaniu wszelkich parametrów przez stos. Język operuje głównie na stosie danych (LIFO, ang. Last In First Out) oraz na stosie adresowym podprogramów (RETURN--STACK). Można również odwoływać się do zmiennych prostych, stałych i tablic — lecz jest to wykonanie pewnych procedur, które swe działanie opierają na stosie.

### Notacja odwrotna

FORTH używa standardowo odwrotnej notacji Łukasiewicza — znanej jako notacja polska. Opiera się ona na zasadzie, że najpierw podaje się argumenty, a następnie polecenie operacji (np. 1+2+4 = ? zapisujemy jako 1 2 4 + +). Przy wykonywaniu obliczenia (1+2)*(4+5) stan stosu przedstawia rysunek:




### Słownik

Procedury języka FORTH nazywa się słowami i zapisuje się w słowniku. Podstawowy słownik standardu FORTH-79 zawiera 128 słów. Programowanie w tym języku polega na tworzeniu nowych słów, które zawierają pewną sekwencję odwołań do słów już istniejących. Przy wykonaniu danego słowa jego składowe są wywoływane kolejno jak procedury, a przekazywanie parametrów odbywa się za pośrednictwem stosu. Każde nowe słowo może być dołączone do słownika i wykorzystywane przy definiowaniu następnych. Można zatem tworzyć różne problemowe odmiany języka (np. do sterowania procesem, do grafiki, system operacyjny itd.).

### Semikompilacja

W wyniku kompilacji nowego słowa pod nagłówkiem z nazwą tworzony jest spis adresów kolejnych słów (procedur, które składają się na dane słowo) oraz stałych, które mają być umieszczone na stosie. Wykonanie należy do interpretera, który przeszukuje słownik (porównując nazwy w nagłówkach) i przekazuje sterowanie do części wykonawczej danego słowa.

### Przykładowy program

**Możemy zdefiniować słowo BIN w sposób następujący:**

```
:     (początek definicji)
BIN   (nazwa słowa)
2     (stała — na wierzchołek stosu)
BASE  (adres zmiennej BASE — na wierz-
       chołek stosu. Zmienna BASE określa
       podstawę systemu liczbowego, w
       którym uzyskuje się wyniki)
!     (zapisanie pod adresem pobranym
       ze stosu (tj. adresem BASE) na-
       stępnej wartości pobranej ze stosu
       (tj. stałej 2))
;     (koniec definicji)
```

Definiując analogicznie:

```
: OKT 8 BASE ! ;
: DEC 10 BASE ! ;
: HEX 16 BASE ! ;
```

możemy wykonać następujące obliczenia:

```
HEX   (ustalenie systemu szesnastkowego)
A     (wpisanie na stos stałej dziesięć)
DE    (powrót do systemu dziesiętnego)
.     (polecenie wydruku lub wyświetlenia
       ostatniej pozycji ze stosu)
CR    (koniec polecenia)
10    (wynik)
OK    (komunikat systemu o wykonaniu
       polecenia)
```

W podobny sposób wykonuje się poniższy ciąg rozkazów:

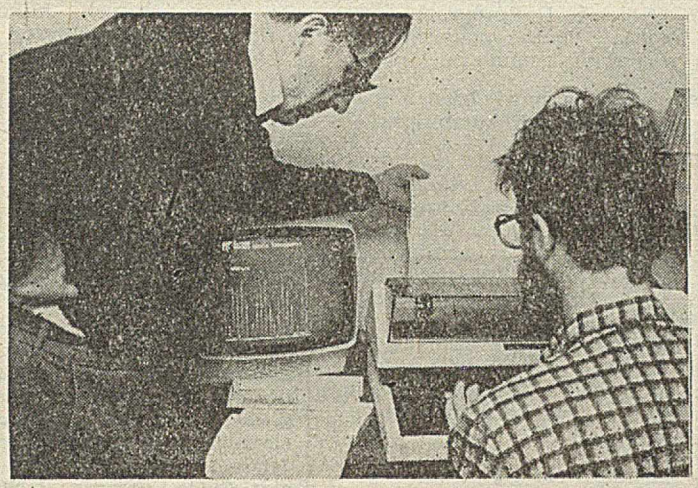
```
BIN 1111 DEC . CR 15 OK
BIN 1111 OKT . CR 17 OK
DEC 10 BIN . CR 1010 OK
HEX FF OKT . CR 377 OK
```

**MAREK CZARZYŃSKI**
ABAKUS WARSZAWA

Mikrokomputer BBC

## Wywiad z posiadaczami BBC

**— Andrzejem Pląskowskim, doktorem chemii, i Witoldem Pląskowskim, magistrem matematyki, którzy zaczynają prowadzić w Warszawie usługi komputerowe przy użyciu własnego mikrokomputera**



**µK: — BBC znaczy British Broadcasting Corporation i kojarzy się z angielską rozgłośnią radiową. Co to ma wspólnego z mikrokomputerem?**

**AWP:** — Wyjaśnienie jest proste. Rozgłośnia BBC ogłosiła przetarg na produkcję seryjną mikrokomputera do celów dydaktycznych — z zamiarem masowego wykorzystania go we własnych programach radiowo-telewizyjnych. Przetarg wygrała firma ICORN ATOM i narodził się mikrokomputer BBC.

**µK: — Cel rozgłośni jest wprawdzie ambitny, ale ma podłoże wyraźnie komercjalne. Zrozumiałe byłyby więc obawy przed takim zakupem. Gdy nie chce się człowiek poddawać panowaniu masowych środków przekazu...**

**AWP:** — Przeciwnie. Od czasu zakupu mikrokomputera prawie nie oglądamy telewizji. Natomiast o zakupie komputera właśnie tej firmy zadecydowały jego walory użytkowe. Przed podjęciem ostatecznej decyzji prowadziliśmy długotrwałe konsultacje. Konkurencja na brytyjskim rynku mikrokomputerowym jest bardzo silna. W 1982 roku, gdy robiliśmy rozeznanie, bardzo popularny był mikrokomputer SHARP. Jednak znajomi radzili wstrzymać się z zakupem — aż do pojawienia się BBC.

**µK: — Jakie są te walory użytkowe? Pozornie kształcącym, a faktycznie ogłupiającym grom telewizyjnym też przypisuje się szereg zalet...**

**AWP:** — Nie w tym rzecz. Prawie wszystkie gry skasowaliśmy już po kilku tygodniach użytkowania, po prostu do gier go szkoda. BBC ma bardzo dobrze opracowany BASIC i najlepszą — zdaniem niektórych — grafikę na świecie (oczywiście w tej klasie sprzętu). Choć jest stosunkowo drogi, sam komputer kosztował 399 funtów.

**µK: — To może kilka słów o samym komputerze.**

**AWP:** — BBC jest oparty na mikroprocesorze typu 6502, ma 32 K słów pamięci użytkowej i 32 K słów przeznaczonych na pamięć stałą ROM. Możliwa jest współpraca z telewizorem systemu PAL lub SECAM, zależnie od zakupionego mikroobwodu — w ośmiu kolorach. Klawiatura jest typowa, QWERT, z dodatkowym rzędem klawiszy funkcyjnych, którym można na stałe przyporządkować dowolne funkcje (obejmujące jednak nie więcej niż 1 linię do 255 znaków). Oczywiście, w zestawie jest także drukarka mozaikowa (80 znaków w wierszu) i stacja dysków elastycznych, której niestety nie mamy. Jako pamięci masowej używamy magnetofonu kasetowego. Do wyposażenia rzadziej spotykanego w takim sprzęcie należą cztery wejścia analogowe i osiem dwustanowych wejść—wyjść cyfrowych. Translatory języków programowania są zawarte w pamięci ROM. Obecnie, oprócz BASICA, powinno być już w sprzedaży kilka innych „kostek", na przykład z PASCALEM.

**µK: — Jakie są możliwości BASICA poza rozbudowaną grafiką?**

**AWP:** — Oprócz typowych instrukcji, które ma każdy język, BBC BASIC ma instrukcje strukturalne:

```
REPEAT..UNTIL
FOR..NEXT
IF..THEN..ELSE
ON..GOTO
ON..GOSUB
```

i szereg funkcji standardowych, jak: sinus, cosinus, tangens i odpowiednie

- ZOINTE przy Instytucie Technologii Elektronowej wydał informację katalogową o wprowadzanym do produkcji przez CEMI zestawie układów mikroprocesorowych (MCY 7880, UCY 74S424 UCY 74S428). Nie byłoby nic nadzwyczajnego w fakcie, że producent informuje o parametrach własnego wyrobu, gdyby nie dotychczasowa praktyka CEMI. Informacja jest stosunkowo obszerna (zawiera nawet opis funkcjonalny — aż trudno uwierzyć!). Do ciemniejszych stron wydarzenia należy zaliczyć nakład (500, słownie: pięćset egzemplarzy!) i cenę zeszytów (np. 230 zł za opis CPU). Zastanawiający jest również fakt, że tym razem pominięto milczeniem (przynajmniej we wspomnianym wydawnictwie), jaki układ był pierwowzorem MCY 7880. Zdaje się, że konsekwencją „zabawy w chowanego" jest istne kuriozum: przetłumaczono na język polski mnemoniki wszystkich rozkazów. Pozostaje zapytać — dlaczego, dla lepszego kamuflażu, nie opracowano wersji ASSEMBLERA pismem klinowym? Byłaby niewątpliwie równie dogodna dla nieźle rozpowszechnionego już w kraju oprogramowania. I jeszcze próbka tłumaczenia: INTEL-owski mnemonik CNZ (ang. Call if Not Zero) to w „polskiej" wersji — KNZ (chyba od Koll jeśli Nie Zero). No cóż, nie będziemy kollać: autor! autor!

- Klub Mikrokomputerowy ABAKUS ogłosił wyniki konkursu na pomysł gry edukacyjnej. Przyznano trzy nagrody (mikrokomputer ZX81-1K, telewizor VELA, magnetofon kasetowy) i osiem wyróżnień (książki o ZX81 w języku angielskim).

  Oto laureaci:

  1. Dr Barbara Chrzan-Feluch (Instytut Matematyki Uniwersytetu Warszawskiego) — za zbiór pomysłów pt. „Matematyka w grach" dla dzieci w wieku 7—16 lat. Pomysły dotyczyły: tabliczki mnożenia, obliczania wielkości kątów i pól figur, orientacji w układzie współrzędnych.
  2. Andrzej Więckowski — za pomysł pt. „Biuro Matrymonialne". Gra jest zbliżona koncepcyjnie do popularnych niegdyś „Dyrektorów". Jej zadaniem jest symulacja warunków rozwoju małej firmy. (A. Więckowski otrzymał także wyróżnienie za pomysł „Sieć bramek logicznych").
  3. Pracownia Zastosowań Naukowo Technicznych ZETO—Wrocław — za pomysł pt. „Kolorowe nutki". Gra ma służyć do elementarnej edukacji muzycznej: określania wysokości dźwięku w gamie.

- Do 31 maja br. można nadsyłać prace w ogłoszonym przez klub ABAKUS konkursie na program gry edukacyjnej. Pierwszą nagrodę stanowi mikrokomputer ZX SPECTRUM! Ogłoszony został również konkurs wewnątrz-klubowy (ze względu na charakter nagrody) na program dla dzieci... w wieku 1—2 lat. Laureat otrzyma 1000 punktów, co w rozliczeniach klubowych równoważne jest np. możliwości korzystania z ZX SPECTRUM przez 500 godzin.

- 28 grudnia ub.r. klub ABAKUS liczył 110 członków. Klub czynny jest codziennie w godzinach 14—20.

- W krakowskiej „Piwnicy pod Baranami" również powstał klub mikrokomputerowy.

- W połowie lutego w warszawskim Pałacu Młodzieży (PKiN) odbędzie się Komputerowy Turniej Szachowy. Uczestnicy będą mogli grać przeciwko komputerowi, lub — za pośrednictwem komputera — w parach przezeń skojarzonych. Zapisy partii będą oczywiście drukowane przez komputer. Współorganizatorem Turnieju jest klub ABAKUS.

- W dniach 12—27 listopada ub.r. w Muzeum Techniki odbyła się wystawa pt. „Komputer dla każdego". Prezentowano wyroby firm parających się techniką mikroprocesorową. Rewelacji nie było. Prym wiódł ABAKUS.

- Powstał zapowiadany (Informatyka nr 5/83) Klub Użytkowników Mikroprocesorów (KUM) — zgłosiło się ponad 100 osób. Przypominamy: KUM zrzesza profesjonalistów, amatorów i instytucje. Celem działania klubu jest wymiana informacji i usług. Planowane jest założenie biblioteki oprogramowania w myśl zasady: wkładasz do wspólnej puli swoją umowną jednostkę programową — otrzymujesz inny, potrzebny ci program. Przewidywane są specjalne premie za wykrycie ewentualnych błędów i usprawnienia programów bibliotecznych.

  Klub zamierza wydawać coroczne spisy nazw, nazwisk, adresów, telefonów wraz z omówieniem sposobu korzystania z techniki mikroprocesorowej — dla wszystkich członków. Będą także periodycznie rozsyłane oferty produktów i usług (na koszt ogłaszającego) do wszystkich zrzeszonych. W biuletynie będzie można również zamieścić prośbę o pomoc w rozwiązaniu określonego problemu. KUM rozpoczął rozmowy w sprawie preferencyjnego traktowania potrzeb członków klubu.

  Miejmy nadzieję, że nie będzie się to odbywać kosztem inych amatorów, nie zrzeszonych w klubie.
  Dla minimalizacji obsługi administracyjnej założono komputerową bazę danych ułatwiającą odnalezienie żądanej informacji, jak również przygotowywanie spisów i biuletynów.
  Do klubu można przytąpić wypełniając ankietę i przesyłając niewielkie wpisowe. Ankietę klubową można otrzymać od założyciela klubu — Andrzeja Drożniaka; ul. Pytlasińskiego 14 m. 4, 00-707 WARSZAWA; tel. dom. 41-26-01.

- **Informacje mikroKLANU prosimy przesyłać bez zwłoki — pod adresem INFORMATYKI: 00-041 Warszawa, ul. Jasna 14/16, p. 244.**

funkcje odwrotne (arecin, arccos, arctan), pierwiastek kwadratowy, logarytm naturalny i dziesiętny, funkcja wykładnicza, moduł, funkcje logiczne AND, OR i NOT, obliczenia stopni kątowch, funkcja losowa i odczyt wejścia analogowego. Liczby w BASICU mogą mieć 10 cyfr znaczących i znak. Dużą zaletą translatora jest jego szybkość. Do programów w BASICU można włączać podprogramy napisane w języku asemblera.

**µK. — A co można powiedzieć o wadach czy ograniczeniach komputera?**

**AWP:** — Jego słabą stroną jest drukarka i klawiatura. Poza tym system załamuje się przy dużych zbiorach. W praktyce długość programu nie może przekroczyć 5000 linii przy operowaniu liczbami całkowitymi i 3000 przy liczbach rzeczywistych. Tego ograniczenia unika się, oczywiście, przy pracy z dyskiem elastycznym. Ze względu na dość mały wymiar strony tekstu, stosunkowo niewygodnie pracuje się z telewizorem o dużym ekranie. Obecnie używamy Neptuna 150.

**µK: — Zgłębiliście Panowie już wszystkie możliwości BBC?**

**AWP:** — Skądże, do tego nam daleko. Intensywnie pracujemy dopiero od kilku miesięcy.

**µK: — A jaki cel Panom w tym przyświeca?**

**AP:** — Obaj pracowaliśmy w instytutach naukowych. Po sprowadzeniu mikrokomputera, rozwiązałem przy jego użyciu pewien niewielki problem. Syn, z zawodu matematyk, stwierdził: „Tata, przecież to można zrobić dużo prościej". I tak się zaczęło. Obecnie używamy go do różnych obliczeń w zagadnieniach praktycznych, np. w określaniu właściwości wybuchów, w pomiarach natężenia ścieków metodą korelacyjną, w częstotliwościowych metodach identyfikacji...

**µK: — Z jakim skutkiem?**

**AP:** — Zadziwiająco pozytywnym. W automatyzacji wielu nawet skomplikowanych obliczeń ten mikrokomputer jest niezastąpiony, znakomicie wyręcza człowieka. Co ciekawe, ta technika promieniuje. Córka, z zawodu zootechnik, zaczęła go używać do obliczeń statystycznych, kilka instytucji po nawiązaniu współpracy z nami nosi się z zamiarem zakupienia BBC. W innych przypadkach, na podstawie naszych opracowań lub założeń, realizuje się zamówione w firmach krajowych wyspecjalizowane urządzenia oparte na mikroprocesorach.

Spośród szeregu podręczników dotyczących mikrokomputera BBC warto wymienić następujce:

Angell O., Jones B. J.: Advanced Graphics with the BBC Microcomputer. Macmillan Press, London, 1982
Birnbaum J.: Assembly Language Programming for the BBC Microcomputer. Macmillan Press, London, 1982
James M.: The BBC Micro — An Expert Guide. Granada Publishing, London, 1983
Scanlon L. J.: 6502 Software Design. Howard Sams, Indianapolis (IN), 1980.

**µK. — Chyba wynika to ze specyficznej sytuacji naszego rynku komputerowego, tzn. braków i cen.**

**AP:** — Nie tylko. Instytut w Manchester gdzie przebywałem zakupił kilkanaście sztuk do celów wyłącznie naukowych.

**µK: — Dziękuję za rozmowę. Życzymy Panom jeszcze większych postępów z BBC. A łamy mikroKLANU czekają na opis ciekawszych rozwiązań.**

Rozmowę prowadził
**JANUSZ ZALEWSKI**

ciąg dalszy ze str. 12

Jeszcze ważniejszą sprawą jest związek między specyfikacjami a weryfikacją. Aby przeprowadzić skuteczną weryfikację programu, trzeba wiedzieć co ma być jej podstawą. Najlepiej wrócić do punktu, w którym opisano intencje, a więc porównać program ze specyfikacjami. Oczywiście, byłoby jeszcze lepiej, gdyby istniała możliwość sprawdzania zgodności projektu z opisem wymagań, ale na razie nie jest to możliwe.

Język przeznaczony do opisu specyfikacji, aby dobrze spełniał swoją rolę, powinien — po pierwsze — być zbliżony do języka, którym posługuje się zleceniodawca, a po drugie — być czytelny dla wszystkich osób zaangażowanych w rozwiązywanie problemu (np. dla osób testujących program).

*

Wiele wysiłku włożono w rozwój języków przeznaczonych do **projektowania programów** (ang. program design) **lub procesów** (ang. process design). Skrót PDL (ang. Program Design Language) jest używany w znaczeniu nie tylko ogólnym, oznaczając języki przeznaczone do opisu projektu programu, lecz także jako nazwa niektórych języków. W znaczeniu ogólnym PDL stanowi półformalną notację umożliwiającą opisywanie projektu programu bez konieczności kodowania tego programu. Wiele języków przeznaczonych do projektowania programów (procesów) pozwala stosować różny sposób opisu poszczególnych kroków programu (procesu) z jednoczesnym uwzględnieniem ich weryfikacji. Dotyczy to przede wszystkim możliwości pisania poszczególnych części programów — albo przy użyciu formalnej notacji, albo języka, który jest wygodny i naturalny dla projektanta. Przykładowo — można napisać:

```
    WHILE
podatekubezp < maxwartość
    DO
odejmij podatekubezp
    ENDDO
```

gdzie słowa WHILE, DO i ENDDO są częścią formalnej notacji natomiast słowa pisane małymi literami są przykładem nieformalnego sposobu opisu pewnego procesu. Ponieważ używanie języka do projektowania związane jest z możliwością stosowania nieformalnej notacji, to języki te zwykle nie są przeznaczone do realizacji maszynowej.

Wiąże się z tym ważny, lecz dotychczas nie rozwiązany problem, dotyczący przejścia od języków projektowania do języków wykonywalnych (np. FORTRANU). W niektórych przypadkach zarówno pierwsze, jak i drugie są językami sformalizowanymi, a więc sprawa przejścia pomiędzy nimi jest prosta. Zamieniając język projektowania na „zwykły" język wysokiego poziomu (który może być przetłumaczony na kod wewnętrzny), programista postępuje zwykle w sposób dość mechaniczny. Zamiana taka może być jednak tylko częściowo zautomatyzowana.

*

W dyskusjach na temat „cyklu życia" programów lub tych jego fragmentów, które dotyczą opracowania konkretnego programu częściej używane jest określenie **programowanie** niż **kodowanie.** Termin „programowanie" zazwyczaj obejmuje część procesu projektowania programu. Ze względu na konieczność jednoznacznego rozróżnienia poszczególnych etapów „cyklu życia", autorka wybrała „kodowanie" dla określenia etapu pisania programu, który następnie można wprowadzić do maszyny i ewentualnie wykonać.

Rozważając problem języków programowania z punktu widzenia inżynierii i metodologii programowania dochodzimy także do sprawy „programowania strukturalnego". Jest to kolejny przykład pojęcia, które ma tyle znaczeń, ilu jest ludzi zainteresowanych tym problemem. Według autorki język programowania dostosowany do programowania strukturalnego powinien zawierać tylko następujące elementy sterujące:

- instrukcje, które będą wykonywane sekwencyjnie
- pewne typy instrukcji iteracyjnych (mogą to być np. konstrukcje typu "WHILE...DO..." lub "FOR I = ...DO...")
- instrukcje decyzyjne, których reprezentantem jest konstrukcja "IF...THEN...ELSE...".

Wszystkie te instrukcje spełniają ogólny warunek, który zakłada, że dopuszcza się tylko takie konstrukcje sterujące wykonaniem programu, które mają jedno wejście na początku i jedno wyjście na końcu.

Uwzględniając techniczny aspekt inżynierii programowania najlepiej kodować programy w językach wysokiego poziomu, takich jak COBOL, FORTRAN lub PL/I. Program napisany w takim języku jest znacznie czytelniejszy niż program w języku asemblerowym; łatwiejsze są także: usuwanie błędów oraz inne etapy „cyklu życia", a szczególnie dokumentowanie. W raporcie Shaw'a i in. (1977) znajduje się porównanie kilku języków, a mianowicie języków COBOL, FORTRAN, JOVIAL i IRONMAN (który był podstawą dla ADY) — z punktu widzenia wyróżnionych aspektów inżynierii programowania.

Z kodowaniem wiąże się problem tworzenia języków wysokiego poziomu, które są naturalne dla problemu użytkownika, chociaż różnią się od większości „zwykłych" języków wysokiego poziomu. Są to języki określone poprzednio jako „języki przeznaczone do specjalnych zastosowań". Przykładami najbardziej rozpowszechnionych języków tego typu są: COGO (COordinate GeOmetry) dla inżynierów budowlanych oraz APT (Automatically Programmed Tool) do sterowania numerycznego obrabiarek.

Zaletami takich języków jest większa czytelność programów oraz możliwość szybkiego opanowania umiejętności programowania (ze względu na podobieństwo notacji stosowanej w danej dziedzinie zastosowań).

*

**Testowanie** nie jest ściśle związane z językami programowania, chociaż jest to niezwykle ważny etap „cyklu życia" programu. Autorka pomija problem istnienia w językach programowania cech ułatwiających testowanie, a także rozwój języków, które są pomocne w pisaniu programów generujących testy. W literaturze poświęca się testowaniu wiele uwagi, lecz większość problemów poruszanych w publikacjach nie wiąże się bezpośrednio z językami programowania.

Autorka zwraca uwagę na fakt, że testowanie i weryfikacja nie są synonimami. Zainteresowanych odsyła do dwóch publikacji: jednej na temat weryfikacji (London, 1979), a drugiej na temat testowania Goodenough, 1979).

*

Każdy wymieniony poprzednio etap „cyklu życia" wymaga **dokumentacji** opisującej i wyjaśniającej zarówno stosowany w nim formalizm, jak i jego treść. Gdyby była możliwa całkowita automatyzacja drogi od opisu wymagań poprzez opis specyfikacji i projektu aż do programu, to wówczas dokumentacja nie byłaby potrzebna z wyjątkiem tych jej fragmentów, które są przeznaczone dla operatorów komputerów.

*

Bardzo kontrowersyjnym tematem jest **weryfikacja,** ale autorka nie chce wdawać się w rozważania dotyczące takich spraw jak np. moment i sposób weryfikacji, jej poziom oraz konieczność wprowadzenia, ponieważ interesują ją przede wszystkim relacje między weryfikacją programu a językami programowania. Jak już stwierdzono, gdyby specyfikacje były jasno opisane i gdyby mogły być automatycznie (lub przynajmniej półautomatycznie) tłumaczone na program wykonywany, to weryfikacja stałaby się sprawą znacznie łatwiejszą. Na razie jednak nie jest to jeszcze możliwe.

Z punktu widzenia języków programowania istnieją obecnie dwa różne podejścia do sprawy weryfikacji. Pierwsze — polega na włączaniu do istniejących języków programowania pewnych dodatkowych instrukcji lub zapewnieniu im takich cech, które ułatwią weryfikację programów. Drugie — polega na opracowywaniu nowych języków uwzględniających narzędzia do weryfikacji (takim językiem jest np. EUCLID, opisany przez Lampson'a i in. w 1977 r.).

Wadą obecnych projektów języków uwzględniających weryfikację programów jest konieczność bardzo dokładnego

sprawdzenia zgodności programu wykonawczego ze specyfikcjami. Przykładowo — jeśli specyfikacje mówią, że trzeba sporządzić 10 kopii określonego raportu, to język powinien ułatwić sprawdzenie, że napisany (i wykonywany) program będzie sporządzał rzeczywiście 10 kopii.

*

Pod pojęciem **konserwacja programu** kryją się następujące problemy: usuwanie błędów, usuwanie niedoskonałości projektu oraz ulepszanie programu. Aby omówić relacje między tymi trzema problemami a językami programowania, należałoby po prostu powtórzyć część poprzednich rozważań na temat niektórych elementów języków.

Warto więc tylko zauważyć, że pewne cechy nowoczesnych języków (np. ogólność danych, modularyzacja) sprawiają, że konserwacja programów napisanych w takich językach jest znacznie łatwiejsza.

*

Dużym zainteresowaniem cieszą się prace nad rozwojem języków ułatwiających pisanie **niezawodnych programów.** Opracowano specjalne języki, nadając im cechy, które według ich projektantów zapewniają większą niezawodność programów. Niestety brak jest danych (lub przynajmniej nie są one znane autorce), na podstawie których można by było jednoznacznie określić cechy języków mające wpływ na niezawodność programów. W związku z tym wybór cech, które powinien mieć taki język i które mają prowadzić do większej niezawodności, jest po prostu oparty na domysłach, że jakiś styl programowania lub cecha języka prowadzą do bardziej niezawodnego programu.

I tak — na przykład — niektórzy twierdzą, że jeśli język nie pozwala na domyślne definiowanie danych (a więc gdy wszystkie dane muszą być zadeklarowane), to programy są bardziej niezawodne. Inni uważają, że nie powinno być automatycznych konwersji danych. Jeżeliby nawet zgodzić się z takim podejściem, to nie można nie docenić wagi kontrargumentu, który mówi, że jeżeli każe się programiście pisać więcej niż potrzeba i jeśli zmusza się go do robienia rzeczy, które może zrobić kompilator, to postępowanie takie także nie prowadzi do zwiększenia niezawodności programów. Żądanie, aby programista pisał założenia w sposób możliwy do sprawdzenia przez kompilator jest według autorki bronią obusieczną.

Oprócz publikacji, w których są przedstawione indywidualne odczucia, było na szczęście kilka prób gromadzenia wiarygodnych danych eksperymentalnych na temat tych cech języków programowania, co do których przypuszczano, że są powodem proporcjonalnie największej (lub najmniejszej) liczby błędów popełnianych przez programistów (np. Gannon, 1977).

*

Z inżynierią programowania wiąże się ściśle sprawa potencjalnej **przenośności programu,** tzn. możliwości przeniesienia uruchomionego programu z jednej maszyny (lub systemu operacyjnego) na inną maszynę (inny system) oraz problem otrzymywania dla danego języka takich samych wyników z różnych translatorów.

Rzecz w tym, aby można było zmieniać wymagania i żądania w stosunku do danej instalacji, a więc zmieniać środowisko sprzętowe, bez konieczności dokonywania jakichkolwiek zmian w istniejących programach. Sprawa ta ma także ważny aspekt ekonomiczny — sprzedawanie i zakup przenośnego oprogramowania jest znacznie bardziej korzystne.

Obecnie największym zainteresowaniem cieszy się przenośność pomiędzy różnymi maszynami, przy czym chodzi tu oczywiście o maszyny, które różnią się architekturą lub listą rozkazów.

Problem przenoszenia programów między maszynami pojawił się już dość dawno. Na ogół jednak mało wiemy — chociaż chyba jest to sprawa ważniejsza — o przenośności między systemami operacyjnymi i kompilatorami. Ten problem zasługuje na rozwinięcie, ponieważ w praktyce kompilatory zmieniane są znacznie częściej niż systemy operacyjne, a te ostatnie — zmieniane (lub modyfikowane) częściej niż maszyny. Niektóre instalacje wymagają dla poszczególnych języków, więcej niż jednego kompilatora (różne kompilatory dla tego samego języka mogą być wykorzystywane do różnych zadań, np. różnego typu optymalizacji w celu otrzymywania wydajnego programu wynikowego).

Barierą na drodze do przenośności mogą być także zmiany systemu operacyjnego, lecz interesują nas one, gdy mają wpływ na kompilatory.

Większość firm produkujących kompilatory stara się oferować produkty lepsze od poprzednich lub od tych, które dostarczają ich konkurenci. Niestety, prowadzi to zazwyczaj do „spustoszenia" wśród istniejących programhów. Jest to spowodowane w pewnym stopniu tym, że wiele języków nie ma ścisłych definicji, a dla tych, które je mają, nie ma pewności, że kompilator pracuje zgodnie z definicją. Chociaż istnieje wiele sformalizowanych metod definiowania syntaktyki i semantyki języków, to jednak tylko niewielu twórców kompilatorów potrafi z nich korzystać w sposób gwarantujący zgodność kompilatora z definicją języka.

Problem weryfikacji kompilatorów jest takim samym problemem, jak weryfikacja innych programów, ponieważ użytkownik korzystający z kompilatora również chce wiedzieć, czy jest on zgodny ze specyfikacjami, tzn. w tym przypadku z definicją języka. Nie można wierzyć w to, żeby w jakimkolwiek przypadku przeprowadzono prawdziwą weryfikację kompilatora. Wprawdzie prowadzono eksperymentalne prace tego typu, ale nie mogą być one stosowane na skalę przemysłową. „Nowy" kompilator może więc oczywiście spowodować, że poprawne dotąd programy staną się w nowych warunkach niepoprawne.
Zdarza się też często, że kompilator znormalizowanego języka ma jednak dodatkowe cechy, których nie ma w standardzie, a do których użytkownicy na ogół chętnie i szybko się przyzwyczajają, ponieważ ułatwiają im one pracę. Nowy kompilator może nie mieć takich samych dodatków, jak poprzedni, a za to zawierać inne. Wtedy często okazuje się, że istniejące programy nie są już poprawne. Wreszcie wiele kompilatorów w czasie tworzenia programów wynikowych korzysta ze specyficznych cech konkretnego systemu operacyjnego. Programiści często wykorzystują te informacje o swoich programach. Naturalnie utrudnia to przenośność, bo różnice pomiędzy systemami operacyjnymi mogą być bardzo subtelne i przez to trudne do wykrycia.

Aby możliwa była przenośność pomiędzy maszynami, nie powinno być wcale lub tylko niewiele takich cech i elementów programu lub języka, które wiążą się ściśle z konkretną maszyną. Znacznie łatwiej jest to jednak powiedzieć, niż zrealizować. Oprócz cech oczywistych, które zależą od maszyny (np. długość słowa, mająca wpływ na dokładność wszystkich obliczeń numerycznych) istnieją również cechy mniej oczywiste (np. porównywanie ciągów, które ma wpływ na sposób realizacji testów nierówności nienumerycznych), a także bardzo subtelne (np. czy zero traktowane jest jak liczba dodatnia, ujemna, czy jak liczba bez znaku).

Przenośność programu można osiągnąć prawie zawsze, jeśli tylko chce się za to zapłacić wydajnością. Przykładem może być język, który ma rozkaz pozwalający czytać taśmę magnetyczną od końca. Jeśli programista wykorzystuje taki rozkaz w programie i potem przenosi ten program na inną maszynę, która nie ma takich możliwości sprzętowych, to rozkaz ten można oczywiście zasymulować, ale będzie to prawdopodobnie tak nieefektywne, że aż trudne do zaakceptowania.

**Opracowały:**

**HALINA CIECHOMSKA, TERESA WÓJCIEKIAN**

**na podstawie referatu Jean E. Sammet, przedstawionego na konferencji SEAS AM'82**

**W dotychczasowym rozwoju informatyki w Polsce dominuje autonomiczność rozwiązań systemów informatycznych, tj. samodzielne tworzenie zbiorów danych wejściowych i ograniczanie się na wyjściu wyłącznie do potrzeb własnej instytucji. Istnieją tylko nieliczne przykłady zasilania systemów informatycznych zbiorami danych na taśmie magnetycznej, tworzonymi w systemach instytucji współpracującej. Mając na uwadze korzyści — przy takich rozwiązaniach (eliminacja błędów, obniżka kosztów przygotowania danych oraz przyspieszenie obiegu informacji gospodarczej) — przedstawiamy je na łamach INFORMATYKI.**

**Pierwsze dwie prezentacje dotyczyły przekazywania zbiorów danych statystycznych: „Współdziałanie resortowego systemu informatycznego MMiPM z innymi krajowymi systemami handlu zagranicznego" — L. Kasman, INFORMATYKA, nr 8—9/82 oraz „Przekazywanie informacji statystycznych na nośnikach magnetycznych" — S. Semczuk, INFORMATYKA, nr 3/83.**

**Poniższy artykuł omawia rozwiązanie w sferze rozliczeń finansowych, widziane ze strony instytucji współpracującej z bankiem. Spodziewamy się, że niebawem wypowie się w tej sprawie także Narodowy Bank Polski. (Red.)**

Współpraca energetyki z bankiem

# Wymiana danych na taśmach magnetycznych

W większości zakładów energetycznych w kraju do rozliczania należności za energię elektryczną i gaz od indywidualnych i uspołecznionych odbiorców jest stosowany system informatyczny ZBYT. System ten usprawnia pracę handlowej i technicznej obsługi odbiorców w rejonach energetycznych — ułatwia prowadzenie ewidencji, wyliczanie i wystawianie rachunków, sporządzanie sprawozdań statystycznych, rejestrację wpłat, windykację zadłużeń oraz gospodarkę licznikową. Daje on również możliwość prowadzenia wieloprzekrojowych analiz zużycia energii elektrycznej i gazu.

System opiera się na rocznym cyklu odczytywania wskazań urządzeń pomiarowych zainstalowanych u odbiorców z uwzględnieniem odczytów kontrolnych w krótszych odstępach czasu. W ciągu roku odbiorcy regulują rachunki okresowe, wyliczone przez komputer w oparciu o wielkości zużycia z poprzedniego roku. Na podstawie danych uzyskanych z odczytu wskazań urządzeń pomiarowych, wyliczane są i drukowane raz w roku rachunki rozliczeniowe. Kwoty rachunków okresowych i rozliczeniowych dla odbiorców uspołecznionych pobierane są z kont bankowych płatników za poomcą poleceń pobrania, a w przypadku zwrotów — poleceń przelewu.

System ZBYT został opracowany przez zespół projektantów i programistów z Zakładu Systemów Masowych Centrum Informatyki Energetyki (CIE) w oparciu o projekt wstępny, przygotowany w 1975 roku przez zespół ekonomistów, informatyków i specjalistów obsługi odbiorców, powołany przez ówczesne Zjednoczenie Energetyki.

Podstawowe moduły systemu wdrożono do eksploatacji w 1978 roku — początkowo w dwóch pilotowych rejonach energetycznych. Po kilku miesiącach doświadczeń system udostępniono pozostałym rejonom energetycznym. Obecnie obsługuje on 11,8 mln odbiorców energii elektrycznej oraz 3,9 mln odbiorców gazu. Rozliczenia z ok. 1 mln odbiorców uspołecznionych realizowane są w obrocie bezgotówkowym. Eksploatacja systemu odbywa się głównie w ośrodkach obliczeniowych resortu energetyki, zlokalizowanych w dużych miastach. Pozostałe zakłady energetyczne, oddalone od tych ośrodków, korzystają z usług przedsiębiorstw ZETO i innych placówek informatycznych.

Rozwiązania systemu dotyczące pobierania należności w obrocie bezgotówkowym (od odbiorców uspołecznionych), są dostosowane do wymogów bankowego podsystemu rozliczeń pieniężnych BANKTAM. Umożliwiają one współpracę z NBP polegającą na wykorzystywaniu zapisanych na taśmie magnetycznej w ośrodkach eksploatujących system ZBYT — dyspozycji rozliczeniowych. Współdziałanie obu systemów rozpoczęto w 1980 roku. Polega ono na comiesięcznym przekazywaniu przez CIE do Centrum Elektronicznego NBP w Warszawie taśm magnetycznych z zapisem dyspozycji rozliczeniowych dotyczących odbiorców uspołecznionych obsługiwanych przez Zakład Energetyczny Warszawa—Miasto.

### System ZBYT

System ten składa się z ok. 50 programów napisanych w językach PLAN i COBOL. Jest on przystosowany do eksploatacji na standardowym zestawie komputera ODRA 1305 (pamięć taśmowa, czytnik kart i drukarka wierszowa). Jednostki sterujące taśm magnetycznych dostosowane są do zapisu i odczytu taśm na przewijaczach PT-3 również w kodzie ISO, co pozwala na ich przekazywanie pomiędzy komputerem ODRA 1305 a innymi maszynami cyfrowymi (NCR, CDC, RIAD, MERA-9150).

Do wprowadzania danych źródłowych na nośniki maszynowe stosowane są rejestratory MERA-9150 oraz dziurkarki kart ARITMA. Dodatkowym wyposażeniem ośrodków eksploatacyjnych systemu są maszyny produkcji firmy BÖWE do cięcia wydruków oraz składania i zszywania książeczek wpłat.

Jak już wspomniano przetwarzanie odbywa się w cyklach miesięcznych. Dla średniej wielkości rejonu energetycznego, obsługującego ok. 60 tys. odbiorców, realizacja takiego cyklu pochłania ok. 20 godzin pracy komputera ODRA 1305. Podstawowe zbiory systemu — kartoteki odbiorców — są w wielu przypadkach (dla rejonów o dużej liczbie odbiorców) zapisywane na kilku krążkach taśmy magnetycznej (na jednym krążku mieszczą się zapisy dotyczące ok. 60 tys. odbiorców).

Ze względu na dużą miesięczną liczbę danych i wynikający stąd znaczny łączny czas pracy komputerów — oprogramowanie systemu zostało zrealizowane z ukierunkowaniem na minimalizację czasu przetwarzania. Cel ten osiągnięto przez wielofunkcyjność programów, która umożliwia ograniczenie liczby operacji czytania i zapisywania rekordów w pamięci zewnętrznej, a także dzięki zastosowaniu języka PLAN.

### Współdziałanie systemów ZBYT i BANKTAM

Współdziałanie pomiędzy tymi systemami obejmuje w chwili obecnej jedynie odcinek pobierania należności od odbiorców uspołecznionych z ich kont bankowych. Polega ono na przekazywaniu z systemu ZBYT taśm

magnetycznych z zapisem dyspozycji rozliczeń należności oraz wydruków komputerowych uwzględniających zredukowanie liczby kopii poleceń pobrania i przekazu. Po opublikowaniu przez NBP w latach 1978—1979 wytycznych dla jednostek gospodarki uspołecznionej korzystających z komputerów do sporządzania dyspozycji rozliczeniowych dla NBP, zasady współpracy zostały udoskonalone i dostosowane do wymogów podsystemu BANKTAM.

Pobieranie należności od odbiorców uspołecznionych rozliczanych w systemie ZBYT przebiega następująco:
- komputer wylicza, zapisuje na taśmie magnetycznej i drukuje rachunki dla płatników wraz z poleceniami pobrania i przelewu oraz niezbędnymi zestawieniami zbiorczymi
- wydrukowane rachunki wysyłane są płatnikom
- polecenia pobrania i przelewu kwoty są przekazywane do oddziału bankowego
- taśma magnetyczna z zapisami poleceń pobrania i przelewu jest przekazywana do bankowego ośrodka obliczeniowego.

Zgodnie z wytycznymi NBP, system ZBYT kontroluje poprawność struktury numerów kont bankowych poszczególnych płatników, a także zgodność poszczególnych członów numerów kont bankowych z ich cyframi kontrolnymi. Inne programowane kontrole zapewniają poprawność zapisów w drukowanych przez maszynę cyfrową poleceniach pobrania i przelewu oraz w przekazywanych do Centrum Elektronicznego NBP taśmach magnetycznych. W efekcie zredukowana została liczba koniecznych kopii dokumentu, a także uproszczona ich treść

Do wytycznych bankowych dostosowano również strukturę zbioru taśmowego z dyspozycjami rozliczeniowymi. W zbiorze tym każda dyspozycja stanowi oddzielny rekord, a ponadto tworzony jest rekord początkowy (identyfikujący zleceniodawcę) oraz rekord końcowy (podający sumaryczną liczbę i wartość kwot poszczególnych rodzajów dyspozycji oraz sumy kontrolne zbioru). Taśma magnetyczna z opisanym w ten sposób zbiorem jest przekazywana do ośrodka obliczeniowego NBP na jeden dzień przed terminem realizacji zapisanych na niej dyspozycji.

Współdziałanie systemów ZBYT i BANKTAM dało m.in. następujące korzyści:
- wyeliminowanie znacznej części żmudnej pracy biurowej pracowników handlowej obsługi odbiorców w rejonach energetycznych — dzięki automatyzacji drukowania dyspozycji rozliczeniowych
- usprawnienie obiegu informacji oraz przyspieszenie spływu gotówki na konta rejonów energetycznych w wyniku uproszczenia procesu przygotowania dokumentów.
- oszczędność zużycia papieru w wyniku zmniejszenia liczby kopii bankowych dokumentów rozliczeniowych
- eleminację pracochłonnej rejestracji danych (z dokumentów źródłowych w bankowym ośrodku obliczeniowym) w wyniku otrzymywania gotowych zbiorów taśmowych z dyspozycjami.

Dostarczenie zbiorów na taśmach magnetycznych do ośrodka obliczeniowego NBP nie daje jeszcze wszystkich spodziewanych korzyści, związanych z eksploatacją systemu ZBYT, ponieważ nie wprowadzono — jak dotąd — wszystkich zaplanowanych form współdziałania systemów. W wytycznych systemu BANKTAM przewidziano bowiem następujące trzy etapy współpracy jednostek gospodarki uspołecznionej z NBP:

**etap I:** równolegle z taśmą magnetyczną do placówek NBP dostarczany jest komplet drukowanych przez komputer dokumentów rozliczeniowych i ich zbiorczych zestawień
**etap II:** NBP rezygnuje z otrzymywania wydruków zbiorczych zestawień dokumentów rozliczeniowych
**etap III:** (docelowy): NBP rezygnuje z otrzymywania wszelkich wydruków oprócz jednostronicowego „Zbiorczego polecenia zleceń rozliczeniowych", będącego dowodem przekazania zbioru taśmowego.

Wymiana taśm magnetycznych odbywa się na razie na poziomie etapu I i to wyłącznie w Warszawie, choć powszechne stosowanie systemu ZBYT umożliwia wprowadzenie tej formy również w innych miastach, oczywiście pod warunkiem zainteresowania ze strony placówek NBP. Energetyka oczekuje również realizacji następnych etapów współpracy, w których zwolnienie z obowiązku sporządzania wydruków dla NBP da znaczne oszczędności zużycia papieru i czasu pracy komputerów w jej ośrodkach informatycznych.

**Perspektywy rozwoju współpracy systemu ZBYT z systemami bankowym i pocztowym**

Najbardziej pracochłonną i uciążliwą czynnością w systemie ZBYT jest rejestracja wpłat dokonywanych w placówkach pocztowych i PKO przez odbiorców indywidualnych. Możliwości usprawnienia tego procesu zarysowują się w dwóch kierunkach.

Pierwszym z nich jest wprowadzenie analogicznej formy pobierania należności od odbiorców indywidualnych, przy pomocy taśm magnetycznych Takie rozwiązanie może być jednak zrealizowane w przypadku znacznego rozpowszechnienia rachunków oszczędnościowo-rozliczeniowych PKO wśród odbiorców oraz składaniu przez nich stałych dyspozycji płatniczych z tych rachunków. Niezbędnym warunkiem jest tu funkcjonowanie informatycznego systemu obsługi rachunków oszczędnościowo-rozliczeniowych co obecnie występuje w większości oddziałów PKO na terenie całego kraju. Koncepcję tę przyjęli już twórcy założeń systemu rozliczania należności za energię elektryczną i gaz już w roku 1975, gdy postanowiono przekazać czynności związane z przyjmowaniem gotówki od odbiorców indywidualnych do urzędów pocztowych i oddziałów PKO. Jednak mała popularność rachunków oszczędnościowo-rozliczeniowych PKO powoduje brak rozwoju tej formy rozliczeń.

Drugą możliwością usprawnienia procesu rejestracji wpłat odbiorców indywidualnych w urzędach pocztowych i oddziałach PKO jest wprowadzenie w tych placówkach informatycznych systemów rejestracji danych na nośnikach magnetycznych. Zbiory magnetyczne z informacjami o przyjętych wpłatach mogłyby być przekazywane z placówek przyjmujących wpłaty do zainteresowanych instytucji — to jest w pierwszej kolejności do oddziałów bankowych, a następnie do właścicieli kont bankowych, na rzecz których są dokonywane wpłaty. Tą drogą ośrodki obliczeniowe energetyki dostawałyby dane o wpłatach zapisane na taśmach magnetycznych, bez potrzeby ręcznej kontroli, uzgadniania i ponownego przepisywania w rejonach energetycznych i w ośrodkach przygotowania danych.

Obie zaprezentowane tu koncepcje wymagają oczywiście szczegółowych uzgodnień ze strony informatyków oraz specjalistów do spraw rozliczeń finansowych i organizacji pracy, a następnie decyzji jednostek nadrzędnych instytucji, pomiędzy którymi proponuje się przekazywanie zbiorów magnetycznych zamiast dokumentów papierowych.

**ANNA STELMASKA**
Centrum Informatyki Energetyki
Warszawa

# Nowości sprzętu komputerowego

W ramach VI Śląskich Dni Organizacji odbyła się w Katowicach w dn. 11—13 października 1983 wystawa sprzętu mini- i mikrokomputerowego. Wzięli w niej udział ważniejsi dostawcy sprzętu aktualnie dostępnego w Polsce. Warto odnotować szczególnie ciekawe eksponaty:

• **Zestaw mikrokomputerowy VT 20/IV** prezentowany przez węgierską firmę VIDEOTON. Tym razem przedstawiono zestaw odbiegający jakościowo od znanych nam rozwiązań. Składa się on z czterech procesorów lokalnego przetwarzania, współpracujących z terminalami typu NDN 52500 i klawiaturami, oraz z procesora wejścia—wyjścia, obsługującego pamięci masowe i drukarkę. Procesory stanowiskowe (terminalowe) oparto na mikroprocesorach Z-80A z pamięcią ROM 1 KB i RAM 4 KB. Procesor wejścia—wyjścia zbudowany również na mikroprocesorach Z-80A i pamięci ROM 6 KB i RAM 64 KB komunikuje się z otoczeniem przez kanały DMA oraz asynchroniczne łącza V-24. Kanały typu DMA pozwalają na przyłączanie do ośmiu jednostek dyskowych o pojemności 5—50 MB oraz drukarki o szybkości 300—1200 wierszy/min. System wyposażony jest w translatory COBOLU, BASICA, FORTRANU i PASCALA. Procesor wejścia—wyjścia wyposażony jest w system zarządzania prostą bazą danych. Przesyłanie danych odbywa się przy wykorzystaniu języka zleceń.

**Mikrokomputer MP** prezentowany przez krajowego producenta MERA-KFAP. Mikrokomputer oparty jest na rozwiązaniach firmy polonijnej IMPOL. Zestaw składa się z modułu procesora z pamięcią 64 KB, monitora ekranowego opartego na telewizorze NEPTUN, klawiatury, jednej lub dwóch jednostek dysków elastycznych oraz drukarki mozaikowej (D-100 lub D-200). System dostosowany jest do pracy z systemem operacyjnym CP/M i dostarczany z kompilatorami BASICA. Jest przeznaczony do wyposażenia stanowisk pracy w instytucjach badawczych i biurach projektów, głównie z uwagi na małe gabaryty, łatwość montażu oraz niską cenę.

• **Mikrokomputer MERITUM I** prezentowany przez zakłady MERA-ELZAB. Wzorowany na koncepcji najpopularniejszych rozwiązań światowych, oparty jest na mikroprocesorze Z-80 i dostępnych elementach krajowych. Wyposażony jest w 14 KB pamięci ROM (przeznaczonej na implementację języka BASIC) oraz 16 KB pamięci RAM, układy dla przyłączenia drukarki, sterowania magnetofonem kasetowym (jako pamięcią zewnętrzną) oraz sterowania standardowym odbiornikiem TV przeznaczonym do wyświetlania tekstów. Mikrokomputer jest wyposażony w rozszerzoną wersję języka BASIC. Producent zapowiada rozwój systemu i włączenie do niego układów obsługi pamięci masowej na dyskach elastycznych oraz odbiornika telewizji kolorowej.

Ponadto wystawione zostały systemy ROBOTRON 5120, MERA 60 i ELWRO 5/3 znane już z poprzednich ekspozycji (m.in. na Targach Poznańskich) oraz systemy oprogramowania użytkowego na mikrokomputerze osobistym typu ZX81 — oferowane przez przedsiębiorstwo NOWATECH.

## V SZKOŁA MIKROPROCESOROWA

**Poniedziałek, 5 marca 1984**

- Prof. Zdzisław Pawlak: V Generacja
- Dr Artur Krepski: PASCAL Mt+ oraz dialekt języka PASCAL dla mikrokomputerów
- Mgr Tomasz Rawiński: System operacyjny UNIX i język C — przyszłościowe narzędzia pracy programistów systemowych mikrokomputerów 8-, 16- i 32-bitowych
- Doc. dr inż. Jacek Kamler: Teletekst i mikrokomputery — nowe środki masowej popularyzacji informatyki
- Mgr inż. Wojciech Trojniar, mgr inż. Krzysztof Plusczok: FORTH — przykłady zastosowań w przemyśle — pokazy działania FORTHA na systemie RTDS-8

**Wtorek, 6 marca 1984**

- Doc. dr inż. Andrzej Hildebrandt: CHILL — język projektowania systemów telekomunikacyjnych; postępy prac nad krajową implementacją języka
- Doc. dr inż. Romuald Marczyński: Wpływ mikroprocesorów na architekturę komputerów
- Mgr Piotr Zapendowski: Zagadnienia komunikacji z operatorem w systemach mikroprocesorowych; minimalizacja interfejsu do wyświetlaczy numerycznych
- Mgr inż. Henryk Kózka: Oprogramowanie małej abonenckiej centrali telefonicznej sterowanej mikroprocesorem

**Środa, 7 marca 1984**

- Mgr Leszek Wilk: Mikroinformatyczna rewolucja edukacyjna
- Dr Janusz Zalewski: Raport dla Klubu Rzymskiego: Mikroelektronika i społeczeństwo — na dobre i złe
- Dyskusja panelowa nt. Raportu dla Klubu Rzymskiego (Romuald Marczyński, Leszek Wilk, Lech Zacher, Władysław Matwin, Adam B. Empacher, Andrzej Kobus).

**Zgłoszenia i szczegółowe informacje:**
**Centrum Szkolenia Informatycznego, ZETO Łódź, ul. Hutora 69, 90-558 Łódź, tel. 647-70, teleks 885208 ZETO PL.**

## GIEŁDA INFORMACJI

### STEP-BAZA

ZETO Poznań proponuje użytkownikom komputerów RIAD z systemem operacyjnym DOS ulepszoną wersję pakietu STEP-BAZA. Różni się ona od pierwowzoru wbudowaniem funkcji umożliwiających aktualizację pól rekordów na poziomie zbioru systemowego SEZAM. Dzięki temu, w wyniku identycznego jak w wersji pierwotnej procesu generacji, program modyfikacji wszystkich kartotek jest wzbogacony o funkcje aktualizacji dowolnego pola wraz z sygnalizacją ewentualnych nadmiarów. Ulepszenia te pozwalają tworzyć bazy danych w pełni przydatne do systemów ewidencyjnych.

**Informacji udziela: mgr Bogdan Niemczyk; ZETO Poznań, ul. Fredry 8a, 61-967 Poznań, tel. 516-34, teleks 0413380.**

### Bezpośredni dostęp do R-32

ZETO Wrocław oferuje dla użytkowników komputerów R-32 posiadających monitory ekranowe MERA 7900 lokalne i zdalne (odpowiedniki IBM 3270), terminale wsadowe (IBM 2270, 2780, 3780), terminale specjalizowane, np. bankowe, zdalne jednostki sterujące, programowane itp. — SYSTEM BEZPOŚREDNIEGO DOSTĘPU z miejsca pracy użytkowników oddalonych od centralnego komputera. Obejmuje on następujące dziedziny:

- rozliczenia bankowe
- ubezpieczenia
- obsługa magazynów w przedsiębiorstwach
- obsługa kadr w przesiębiorstwach
- informacja ogólna dla kierownictwa przedsiębiorstwa
- planowanie produkcji w przedsiębiorstwie
- rezerwacja miejsc w biurach podróży
- zbieranie danych o sprzedaży w handlu
- zbieranie danych statystycznych
- zbieranie danych z linii produkcyjnych
- rozliczenia w handlu zagranicznym
- ewidencja zamówień, nadzór nad ich bieżącym wykonywaniem
- obsługa księgowości.

System zapewnia ciągły dostęp do danych problemowych związanych z nadzorowanymi zagadnieniami oraz umożliwia przedstawienie wyników bezpośrednio w miejscu pracy — od gabinetu dyrektora (informacje ogólne dla kierownictwa) do okienka kasowego (obsługa księgowości i finansów). ZETO Wrocław oferuje także gotowe systemy aplikacyjne, usługi obejmujące pisanie nowych systemów aplikacyjnych, a także — dwuetapowy kurs dla aktualnych i przyszłych użytkowników programów obsługujących wymienione dziedziny. Oferowany system jest odpowiednikiem systemu CICS firmy IBM.

**Adres: Zakład Elektronicznej Techniki Obliczeniowej; ul. Ofiar Oświęcimskich 7/14, 50-069 Wrocław; tel. centr. 44-54-31 do 37; teleks 0712533 ZETO PL.**

# Pakiety czy programy na zamówienie?

W ciągu ostatnich niewielu lat poglądy na temat efektywności metod tworzenia oprogramowania kilkakrotnie wywracały się do góry nogami. Aż do późnych lat siedemdziesiątych metoda kupowania pakietów była dla większości zastosowań uważana za nie najlepszą. Pakiety traktowano jak „prowincję" oprogramowania i niewielu szefów odpowiedzialnych za przetwarzanie danych chciało je stosować. Pogląd ten opierał się na dobrze uargumentowanym przekonaniu, że pakiety są za mało elastyczne, mają złą dokumentację i przynoszą więcej kłopotów niż są warte. Argumentem przeciw pakietom, zwłaszcza w Europie, było też istnienie utrwalonych tradycją metod prowadzenia działalności różniących się w zależności od przedsiębiorstwa. W istocie, jedyną dziedziną, w której pakiety mogły spełniać wiodącą rolę, były listy płac, gdzie zasady systemu zależą głównie od warunków zewnętrznych, tzn. regulacji prawnych i umów zbiorowych.

Na przełomie dekad nastąpił znaczny wzrost sprzedaży pakietów. Było to spowodowane z jednej strony czynnikami ekonomicznymi (produkcja programów stawała się coraz droższa), z drugiej zaś tym, że pakiety były dużo lepsze od swoich poprzedników. Również przyczynił się tutaj dużo większy niż w latach siedemdziesiątych niedobór wykwalifikowanych informatyków, coraz częściej obciążonych konserwacją eksploatowanych systemów. Nowych programów po prostu nie miał kto pisać.

Wskaźnikiem dużego popytu, jakim cieszą się obecnie pakiety, mogą być przykładowe dane rynkowe. Zgodnie z nimi, rynek pakietów dla różnych zastosowań pochodzących od niezależnych (innych niż producenci sprzętu) wytwórców wykazywał obroty 31,2 mln dol. w 1979 roku i 89 mln w 1983. Oznacza to 30% wzrost roczny, a w niektórych krajach, np. w Wielkiej Brytanii, w ciągu dwóch lat (1979—1981) nastąpiło nawet podwojenie obrotów. Wskaźnikiem wzrostu znaczenia pakietów mogą być także zyski prawdopodobnie największego dziś w świecie przedsiębiorstwa w dziedzinie oprogramowania — MANAGEMENT SCIENCE AMERICA. Obecnie MSA oczekuje, jako pierwsze w tej dziedzinie, przekroczenia bariery 100 mln dol. obrotu rocznie — wyłącznie ze sprzedaży oprogramowania zastosowaniowego.

Na liczby te nakłada się oczywiście nagły rozwój mikroinformatyki, gdyż sprzedaje się więcej pakietów do tych zastosowań niż do tradycyjnego przetwarzania danych. Ale — z drugiej strony rozróżnianie tych dwóch dziedzin staje się coraz trudniejsze. W miarę jak przedsiębiorstwa włączają zastosowania mikrokomputerów do swoich dotychczasowych systemów przetwarzania danych, różnice stają się nieistotne. Obecne kierunki wskazują, więc, że przyszłość należy do pakietów. Ale czy pakiety programów to rzeczywiście dobre i perspektywiczne rozwiązanie problemu rozwoju oprogramowania?

W tym samym czasie, gdy pakiety zdobywały coraz większą popularność, obserwowano innego rodzaju prace nad rozwojem oprogramowania, które mogą wskazywać na jego przeciwny kierunek. Na wszystkich poziomach rynku informatycznego nastąpił wzrost efektywności tego, co nazywamy oprogramowaniem narzędziowym. Od słynnych systemów mikrokomputerowych, takich jak np. LAST ONE (DJ AI DATA SYSTEMS), poprzez generatory kodu języka COBOL (np. DATA LOGICS READYCODE), aż po skomlipkowane narzędzia typu PROGRAMMERS' WORK BENCH wbudowane w systemy operacyjne UNIX, realizowała się rewolucja w podejściu do tworzenia programów. Wykonany na zlecenie rządu Wielkiej Brytanii raport Alveya idzie nawet dalej, kładąc nacisk na tak ezoteryczne pomysły, jak inżynieria oprogramowania czy systemy typu „ekspert". We wszystkich tych przypadkach podkreśla się rolę bardziej efektywnych sposobów produkcji programów dopasowanych do potrzeb klienta, niż uniwersalne pakiety.

Najlepszym przykładem będzie tu chyba metoda Systematics, którą opracował dr Kit Grindley z URWICK DYNAMICS. Systematics powstała w połowie lat sześćdziesiątych jako język analizy systemów, ale gdy Grindley pracował nad nią przez jakiś czas, stwierdził, że będzie bezużyteczna, dopóki nie będzie mogła być użyta jako podstawa do generacji kodu. Obecnie uważa swoją metodę za sposób tworzenia dostępnego natychmiast oprogramowania — zrób program szybko i, jeśli nie spełnia wymagań, wyrzuć go bez żalu. Pomysł „prototypowych" programów zwraca uwagę na istotny problem związany z gotowymi pakietami. W większości przypadków użytkownik nie wie, jaki system jest potrzebny i dlatego trudno mu wybrać taki, który spełni wszystkie jego nadzieje i potrzeby. Sprzedawcy pakietów będą przekonywać, że ich programy są dostatecznie elastyczne, by poradzić sobie z konkretnym problemem. W niektórych przypadkach jest to zapewne prawda, ale w świetle doświadczeń, wydaje się nieprawdopodobne, by pakiety mogły dać się adaptować do potrzeb użytkownika w sytuacjach, gdy po średnio lub długookresowym wykorzystaniu zawiódł system budowany na zamówienie.

Jeszcze raz trzeba zwrócić uwagę na postęp w sposobie myślenia o oprogramowaniu użytkowym w ostatnich kilku latach. Pojęcie cyklu życia programu, pochodzące z prac nad tworzeniem i konserwacją systemów operacyjnych, zaczyna być rozumiane przez twórców zastosowań. Systemy nie są statycznym, ale dynamicznym odzwierciedleniem niektórych aspektów rzeczywistego świata — i muszą uwzględniać możliwość zmieniania się. Przykłady tego są widoczne w zmianach metod programowania. Modularne, strukturalne kodowanie ułatwia proceduralne zmiany w systemach — gdy jakaś procedura staje się nieużyteczna, może być łatwo zastąpiona przez aktualną wersję. Podobne podejście zastosowano w stosunku do danych, wraz z rosnącym wykorzystaniem systemów baz danych i słowników danych. W razie potrzeby bieżących zmian, programiści nie muszą już śledzić treści każdego programu, szukając danych, które należy poprawić. Prosta zmiana w słowniku danych oznacza (lub powinna oznaczać), że takie długotrwałe zajęcia należą do przeszłości. Jeszcze prostszym przykładem jest usunięcie z programów wszystkich stałych i umieszczenie ich w jednym zbiorze, co znakomicie ułatwia późniejsze zmiany. Wszystkie takie pomysły wskazują, że potrafimy coraz lepiej programować. Oczywiście, z tego samego powodu pakiety też są dużo lepsze niż były kiedyś.

Wraz ze znaczącą poprawą metod nastąpił jednak jeszcze większy ilościowy i jakościowy wzrost zapotrzebowania na programy użytkowe. Dziesięć lat temu głównym zajęciem ośrodków obliczeniowych było wprowadzanie tak dobrze sprecyzowanych zastosowań, jak gospodarka magazynowa czy przetwarzanie zamówień. Tego typu zadania byłyby dużo łatwiejsze do realizacji przy użyciu dziś dostępnych narzędzi, niemniej dzisiejsze zadania są dużo bardziej złożone. Zleceniodawca nie zadowala się wypełnianiem paczki formularzy i otrzymywaniem po tygodniu pokaźnego stosu wyników. Teraz chce, żeby system pozwalał mu — z jednej strony — być zarządcą danych, a z drugiej — wykonywać na nich operacje nie dające się z góry określić. I jak zwykle wzrost potrzeb użytkowników znacznie wyprzedza możliwości zaspokojenia ich przez ośrodki obliczeniowe.

Na marginesie warto przytoczyć interesujące dane z kwartalnika COMPUTING — otóż okazuje się, że kierowniczy personel informatyczny uważa za swoje najważniejsze zadanie przewidywanie przyszłych technik i metod. To zrozumiałe. Kluczowe kierunki rozwoju, mające wpływ na oczekiwania użytkownika, są zawsze

skoncentrowane na nowych rozwiązaniach problemów. Nowe języki programowania, techniki zarządzania bazami danych, nowe urządzenia peryferyjne, czy wreszcie systemy typu „ekspert" lub sztuczna inteligencja — zawsze pojawiają się w referatach i artykułach znacznie wcześniej niż staną się praktycznym rozwiązaniem rzeczywistych zadań. I nie można winić użytkownika, gdy żąda tych przyszłościowych metod już dziś — skoro wydają mu się najlepszym sposobem rozwiązania jego problemów.

Obecnie potrzeby użytkowników są jeszcze większe, rozbudzone przez przetwarzanie rozłożone, dostępne dzięki tanim mikrokomputerom i terminalom. Dodatkowym wsparciem jest szeroka dostępność gotowego oprogramowania dla tego taniego sprzętu, w sytuacji gdy użytkownik na ogół nie potrafi albo nie ma czasu pisać własnych programów.

Większość mikrokomputerów trafia obecnie do rąk tych, których przedstawiciele producentów nazywają nieskomplikowanymi użytkownikami lub „nieprogramistami". Wraz z odpowiednimi pakietami oprogramowania mikrokomputery sprzedaje się na fali naiwnego entuzjazmu nowego rodzaju użytkowników, wierzących, że to właśnie rozwiązanie ich problemów.

Co dzieje się, gdy blednie pierwsze olśnienie? Co dzieje się, gdy nieprogramista musi zakasać rękawy i zacząć programować, by dokonać w systemie pożądanych zmian? Zawodowi informatycy na pewno przypomną sobie uczucie rozczarowania, jakie stało się ich udziałem, gdy odkrywali, że system przez nich napisany (zwykle debiut) jest całkowicie poza ich kontrolą. Tyle, że oni mieli przynajmniej do pomocy bardziej doświadczonych kolegów. Jak sobie z tym poradzą ci niefachowcy? Bez wątpienia zwrócą się do zawodowców i naiwnie stwierdzą: „komputer się popsuł". Dalej, jeśli ta maszyna to mikroprocesorowy terminal przyłączony do dużej sieci, jakich głupstw mogą narobić swoimi ślepymi próbami, zanim zwrócą się do informatyków?

Mądrzy ludzie odpowiedzialni za informatykę są ostrożni w rozpowszechnianiu takich urządzeń — a głównym powodem tego rozpowszechniana jest dostępność taniego oprogramowania w formie pakietów. Opierają się też próbom zastąpienia efektów pracy zawodowych programistów przez programy ze sklepowej półki.

Nie należy oczywiście sądzić, że pakiety programów użytkowych są w ogóle niepotrzebne. W odpowiednich warunkach, w odpowiednim stopniu kontrolowane, mogą być bardzo użyteczne, ale zastosowanie pakietu musi być traktowane tak samo, jak każde wdrożenie nowego programu. Do celów konserwacji musi być dostępny kod źródłowy, by zawodowi programiści mogli się z nim bliżej zapoznać. I nie jest ważne, jak dobra jest dokumentacja, bo w końcu to program pracuje, a nie jego podręcznik. Jeśli trzeba się oprzeć na zewnętrznych dostawcach, należy upewnić się, czy odpowiednie zobowiązania są w umowie wyrażone czarno na białym i czy dostawca jest zdolny sprostać wymaganiom. Warto również zadbać o to, by użytkownik był odseparowany od technicznych aspektów pakietu. Oczywiście, musi on mieć możliwość wykonania tego, czego sobie życzy, ale nigdy kosztem niebezpiecznego systemu. Wszyscy nowicjusze w dziedzinie informatyki odczuwają nieprzepartą chęć majstrowania i szukania słabych punktów systemów. Rozsądny szef ośrodka musi upewnić się, że takie majstrowanie nie zakłóci prawidłowego działania całego systemu.

Obecnie panuje przekonanie, że pakiety mogłyby być rozwiązaniem wszystkich problemów użytkowych, choć istnieją też co do tego wątpliwości. Jak zwykle — pożyjemy, zobaczymy. Ale zanim to się rozstrzygnie, byłoby głupotą pozbywać się możliwości stosowania własnych rozwiązań.

* * *

Wielu Czytelników zapewne pomyśli teraz: ależ ci Anglicy mają kłopoty — za dużo taniego sprzętu i oprogramowania! Niektórzy nawet pozazdroszczą takich problemów. Ale przecież i w Polsce, choć jeszcze dotyczy to raczej dużych komputerów, coraz częściej mamy do czynienia z pakietami. I nie zawsze wywołują one wyłącznie entuzjazm. Nagłe rozprzestrzenianie się mikrokomputerów też w końcu musi nastąpić. A co wtedy? Czy nie lepiej już dziś uwzględnić te ostrzeżenia w swoich planach? Uczenie się na błędach jest tanie tylko wtedy, gdy są to cudze błędy. A nie ma żadnego usprawiedliwienia dla tych, którzy popełniają błędy wiedząc, że ktoś już je popełnił.

**Oprac. MAREK SOBCZYK**

**na podst. Computer Management z marca 1983**



# CACM

# ćwierćwiecze istnienia

**Na czym oszczędzać łatwo? Choćby na literaturze naukowo-technicznej. Oszczędność to wprawdzie pozorna, ale — jak uczy doświadczenie — działania pozorne są nadal bardzo cenione. Przekonują się o tym zwłaszcza mniejsze ośrodki, do których dopływ zagranicznych publikacji został niemal zatrzymany. Aby choć w minimalnym stopniu ułatwić tym ośrodkom dostęp do zagranicznych źródeł, postanowiliśmy zamieszczać co pewien czas przegląd roczników poważnych światowych czasopism informatycznych. Ze względu na ograniczoną objętość podobnych przeglądów, poszczególne artykuły omawiane będą w sposób bardzo skrótowy. Widzimy jednak możliwość zamieszczania wyczerpujących opracowań tych prac, które spotkają się ze szczególnym zainteresowaniem Czytelników.**

**Na początek proponujemy przegląd artykułów zamieszczonych w jubileuszowym numerze CACM, rozpoczynającym drugie ćwierćwiecze istnienia tego pisma.**

**REDAKCJA**

W dwudziestą piątą rocznicę powstania Communications of the Association for Computing Machinery ukazał się numer jubileuszowy tego pisma [1], a w nim zbiór tych artykułów opublikowanych w CACM, które „wstrząsnęły światem". Wpływ, jaki w ciągu ostatniego ćwierćwiecza wywarły owe artykuły na sposób myślenia zarówno praktyków, jak i teoretyków informatyki, jest tak znaczny, że warto im poświęcić nieco miejsca na łamach naszego miesięcznika. Opracowując ten obszerny materiał stanąłem wobec poważnego problemu: które prace omówić mniej pobieżnie, a o których tylko wspomnieć. Ponieważ pominięcia jakiegokolwiek z przypomnianych przez CACM tekstów nie da się uzasadnić merytorycznie, przyjąłem skrajnie subiektywne kryterium wyboru — własny gust, zakładając jednak, iż o każdym artykule napiszę przynajmniej kilka słów.

[1] CACM, vol. 26, No. 1, January 1983

Dwa pierwsze artykuły, które redakcja CACM umieściła wśród najważniejszych publikacji w dwudziestopięcioletniej historii tego pisma, dotyczą **sposobu kompilacji programów ALGOLOWYCH** [18, 10]. W pierwszym z nich (w porządku chronologicznym) Samelson i Bauer wprowadzili do metod translacji technikę użycia stosu, która spowodowała prawdziwy przewrót w dziedzinie budowy kompilatorów. Natomiast Irons zauważył, że rekurencyjna postać reguł gramatycznych opisujących ALGOL (w notacji Backusa-Naura) może być łatwo przekształcona na układ algolowych procedur rekurencyjnych. Podejście to w sposób naturalny doprowadziło do koncepcji zapisywania kompilatorów w języku, który ma być kompilowany.

Przeskakując o dziesięć lat do przodu, znajdziemy w CACM jeszcze jeden ważny artykuł z dziedziny translacji [7], w którym J. Earley przedstawił efektywny algorytm sprawdzający, czy dane wyrażenie należy do języka opisanego gramatyką bezkontekstową. Jeśli **n** jest długością wyrażenia wejściowego, to ilość czasu zużyta przez ten algorytm jest w najgorszym przypadku proporcjonalna do $n^3$. Co więcej — dla szerokich klas języków bezkontekstowych liczba ta zmniejsza się do $n^2$, a często nawet do **n**.

**Obsługa tablic rozproszonych**, to kolejny dział informatyki, którego burzliwy rozwój znalazł swe odzwierciedlenie na łamach CACM. W pracy Maurera [12] zawarty jest pomysł, aby rozpatrywać funkcje rozpraszające jako funkcje oparte o modularną arytmetykę (a nie jak dotychczas — jako operacje niskiego poziomu działające na bitach). Wykazał on ponadto, iż jako rozmiary tablic lepiej wybierać liczby pierwsze niż potęgi dwójki. Innym ważnym wynikiem Maurera jest porównanie trzech znanych wcześniej metod usuwania kolizji, przy czym metoda kwadratowa okazała się lepsza niż metoda pseudolosowa i liniowa.

Nieco inny charakter ma artykuł Morrisa [13], stanowiący podsumowanie dotychczasowych prac związanych z tablicami rozproszonymi. Przedstawione w nim zostały zarówno wyniki praktyczne, jak i analiza teoretyczna rozpatrywanych metod. Dwa lata później opublikowana praca J. R. Bella [1] porusza jeden z problemów spotykanych we wczesnych algorytmach obsługi tablic rozproszonych — tworzenie się grup kluczy o tych samych ciągach adresów dostarczanych przez procedurę usuwania kolizji (ang. clustering). Metoda kwadratowa eliminowała grupy tworzące się przy metodzie liniowej, niemniej jednak pozostawiała inny rodzaj grup kluczy. Natomiast algorytm Bella całkowicie rozwiązał ten problem, zmniejszając tym samym średni czas poszukiwania kluczy w tablicy.

Kolejna porcja artykułów dotyczy **systemów operacyjnych.** Składają się na nią trzy publikacje z lat 1968—1974. Pierwsza z nich [3] stanowi jedną z najważniejszych prac dotyczących problematyki zarządzania pamięcią wewnętrzną maszyny. P. J. Denning wprowadził w niej pojęcie zbioru roboczego, jako minimalnego obszaru pamięci potrzebnego do efektywnego wykorzystania procesora przy stronicowaniu pamięci. Wykazał on, iż wielkość zbioru roboczego można mierzyć dynamicznie. Jego praca zapoczątkowała liczne badania nad analitycznymi modelami zachowania się programów.

Odmienną tematykę podjął w swoim artykule [6] E. W. Dijkstra. Otóż przedstawił on niezwykle istotną koncepcję strukturalizacji wieloprogramowego systemu operacyjnego poprzez stworzenie hierarchii abstrakcyjnych maszyn implementowanych jako warstwy oprogramowania. Pomysł Dijkstry jest stosowany do dziś przy tworzeniu dużych systemów, bowiem hierarchiczna modularyzacja okazała się niezwykle silnym narzędziem organizacji tych systemów w sposób umożliwiający ogarnięcie ich struktury logicznej przez jednego człowieka.

Trzeci artykuł z omawianej grupy [16] zawiera opis koncepcji i implementacji jednego z najbardziej rozpowszechnionych systemów operacyjnych — UNIXA. Ponieważ system ten zostanie wkrótce opisany na łamach INFORMATYKI, nad pracą Ritchiego i Thompsona nie będę się tu zatrzymywać.

W związku z rozwojem systemów operacyjnych i pojawieniem się wieloprogramowania, dostrzeżono wiele zagadnień charakterystycznych dla **programowania współbieżnego.** Jednym z prekursorów prowadzenia badań w tej dziedzinie był E. W. Dijkstra. W 1965 roku opublikował on pracę [5], która zapoczątkowała erę zainteresowania synchronizacją procesów współbieżnych. Przedstawione tam rozwiązanie problemu sekcji krytycznej polega na wprowadzeniu globalnych tablic logicznych. W późniejszych pracach Dijkstra zaproponował bardziej efektywny sposób zapewnienia wzajemnego wykluczania procesów przy użyciu semaforów.

Zbliżoną problematyką zajęli się także Dennis i Van Horn w [4]. Zdefiniowali oni operacje istotne przy tworzeniu programów współbieżnych. Najbardziej znane spośród nich (i używane do dziś) są niepodzielne instrukcje lock i unlock określone na semaforach binarnych I, o następującym działaniu:

lock w = [**while** w = 1 **do od;** w: =1]
unlock w = [w: = 0]

Nie mniej istotne jest nowe podejście Dennisa i Van Horna do konstruowania systemów operacyjnych. W ich ujęciu system jest abstrakcyjną maszyną dostarczającą użytkownikowi wygodnych operacji służących do manipulowania zasobami komputera. Zauważmy że w podobnej koncepcji zawarta jest pośrednio cała idea abstrahowania typów danych, odkryta na nowo dopiero w latach siedemdziesiątych. W następnych latach prowadzone były liczne prace mające na celu uzyskanie języków programowania współbieżnego bogatszych od języka rozważanego przez Dennisa i Van Horna o strukturalne narzędzia synchronizacji i komunikacji między procesami. Każde z tych narzędzi nastawione było na rozwiązanie pewnej klasy problemów. Natomiast w 1978 roku C. A. R. Hoare [9] zaproponował bardzo prosty język, w którym można rozwiązać wszystkie rozważane problemy w sposób efektywny. Jego pomysł polegał na wprowadzeniu takiej metody komunikacji między procesami, w której transmisja danych następuje jedynie wtedy, gdy żąda jej zarówno odbiorca, jak i nadawca danych.

Odmienną dziedziną — **metodologię programowania** — reprezentują artykuły N. Wirtha i D. L. Parnasa [21, 15] (poprzednio omawiane artykuły miały z tą dziedziną związek mniej lub bardziej pośredni). Praca [21] ukazała się po słynnym liście Dijkstry o programowaniu strukturalnym. Wirth zaproponował w niej metodę, w której celem programisty powinno być uzyskanie takiej struktury programu, która da się opisać za pomocą warstw oprogramowania. Pierwsza warstwa zawiera zapis algorytmu w pewnym abstrakcyjnym języku programowania, zaś następne implementują pojęcia z tego języka za pomocą coraz prostszych konstrukcji. Ostatnią warstwą jest język bazowy. Natomiast Parnas zajął się w swym artykule techniką specyfikowania modułów programów w terminach relacji wejścia—wyjścia. Specyfikacja powinna być zupełna, tzn. powinna zawierać wystarczające informacje — zarówno dla implementatorów modułu, jak i dla jego użytkowników (w tym dokładny opis współpracy modułów). Niedopuszczalne jest, aby programiści przy tworzeniu kolejnych modułów korzystali ze szczególnych cech realizacyjnych innego modułu; bowiem takie postępowanie praktycznie uniemożliwia modyfikację niewielkich nawet fragmentów oprogramowania. Metoda Parnasa stanowi pierwszy bardzo wyraźny krok w kierunku abstrakcyjnych typów danych.

Już od piętnastu lat dowodzi się na szerokim świecie **poprawności programów.** A spowodował to słynny artykuł [8], w którym C.A.R. Hoare, wychodząc od metody zaproponowanej przez Floyda (1967), przedstawił zbiór reguł dowodzenia częściowej poprawności programów. Wprowadził on formuły postaci **P** {**Q**} **S**, oznaczające: „jeśli dane spełniają warunek **P** i Program **Q** się zatrzymuje, to wyniki jego spełniają warunek **S**". System Hoare'a jest skończony, nie jest zatem pełny. Wielu autorów podawało przykłady właściwości programów, które nie mogą być udowodnione za pomocą tego systemu. Nie zmienia to faktu, iż metoda Floyda-Hoare'a do dziś nie straciła swej praktycznej użyteczności. Przypomnijmy krótko reguły tej metody, które spowodowały rewolucję w teorii i praktyce programowania:

D0. P{x: = f} S gdzie P uzyskujemy z S przez zastąpienie wszystkich wystąpień zmiennej x przez wyrażenie f
D1. z P{Q}R oraz R → S wnioskujemy P{Q}S;
z P{Q}R oraz S → P wnioskujemy S{Q}R
D2. z P{$Q_1$}$R_1$ oraz $R_1${$Q_2$}R wnioskujemy P{begin $Q_1$ ; $Q_2$ end}R
D3. z P ⌐B{Q}P wnioskujemy P{while B do Q} ∧ B P. >

Znacznie bogatszy zbiór reguł Czytelnik może znaleźć np. w książce Alagića i Arbiba „Projektowanie programów poprawnych i dobrze zbudowanych" (por. z recenzją zamieszczoną w INFORMATYCE nr 7—8/1983).

**Przetwarzanie danych** reprezentowane jest w jubileuszowym numerze CACM przez dwa artykuły. W pierwszym z nich [19], bardzo wczesnym (1963), Sussenguth przedstawił propozycję organizacji pliku w strukturę drzewiastą, umożliwiającą przeszukiwanie i modyfikacje pliku w czasie proporcjonalnym do log N, gdzie N jest liczbą rekordów zawartych w pliku. Natomiast drugi artykuł [2] zawiera coraz powszechniej przyjmowaną koncepcję relacyjnych baz danych. W porównaniu z hierarchicznym i sieciowym modelem dla baz danych, model relacyjny charakteryzuje się prostotą i przejrzystością. Codd rozważa bazy danych jako relacje matemtayczne, wprowadzając takie operacje określone na relacjach, jak: permutacja, rzut, połączenie, złożenie i obcięcie relacji. Istota koncepcji Codda polega na dostarczeniu podstaw do stworzenia języka programowania wysokiego poziomu zorientowanego na problematykę baz danych, który dawałby maksymalną niezależność programów od rzeczywistej, maszynowej reprezentacji danych, a tym samym uwalniał programistów od myślenia nad technicznymi szczegółami bazy danych.

W 1976 roku Diffie i Hellman postawili problem stworzenia tzw. **szyfru publicznego**, w którym zaszyfrowanie i odszyfrowanie informacji wymagałoby użycia różnych kluczy. Artykuł Rivesta, Shamira i Adlemana [17] stanowi rozwiązanie tego problemu. Dzięki ich metodzie procedura szyfrująca może być znana wszystkim. Jej znajomość nie wystarcza jednak do odszyfrowania informacji. Złamanie szyfu przy zastosowaniu najszybszych znanych metod musiałoby zająć ok. $10^9$ lat dla informacji zakodowanej za pomocą liczby dwustucyfrowej (dla liczb pięciosetcyfrowych czas ten wzrasta do $10^{25}$ lat).

Dziedzinę **sztucznej inteligencji** porusza zaledwie jeden spośród artykułów wybranych przez redakcję CACM [20]. Opisany jest w nim program ELIZA, który umożliwiał konwersację pomiędzy człowiekiem a komputerem. Komputer dawał wiarygodne odpowiedzi bez próby zrozumienia przedmiotu rozmowy. Główna procedura programu była bardzo prosta: tekst był czytany i badano w nim obecność słów kluczowych. Jeśli słowo takie zostało znalezione, zdanie przetwarzano zgodnie z transformacją przypisaną temu słowu, w przeciwnym przypadku program czynił uwagę natury ogólnej lub odwoływał się do informacji uzyskanej poprzednio. Podstawowym wynikiem Weizenbauma był w tym przypadku zadziwiający fakt, iż stosunkowo prosty zbiór reguł transformacji może dać wiarygodną rozmowę.

Aby nasz przegląd był kompletny, wspomnijmy jeszcze o dwóch artykułach. Pierwszy z nich [11] dotyczy **uzyskiwania funkcji numerycznych** odpornych na błędy zaokrągleń. Kuki i Cody zbadali w nim i porównali z punktu widzenia problematyki błędów zaokrągleń różne metody implementacji arytmetyki zmiennoprzecinkowej. Druga praca [14] związana jest z **rozproszonymi sieciami komputerowymi**. Oba artykuły są bardzo istotne z punktu widzenia współczesnej informatyki, jednak bardziej szczegółowe przedstawienie metod Kuki'ego i Coddy'ego, czy też sieci ETHERNET wymagałoby znacznie więcej miejsca niż mogliśmy poświęcić na cały przegląd.

Na zakończenie podsumujmy: najliczniej reprezentowane w powyższym przeglądzie działy informatyki, to metody kompilacji, tablice rozproszone, systemy operacyjne i równoległość. W przyszłym ćwierćwieczu — jak sądzę — należy się spodziewać że rozszerzona zostanie problematyka związana ze współbieżnością, sztuczną inteligencją i sieciami komputerowymi (być może również z metodologią i teorią programowania).

LITERATURA


[1] Bell J. R.: The Quadratic Quotient Method: A Hash Code Eliminating Secondary Clustering (1970). CACM, vol. 26, nr 1, 1983
[2] Codd E. F.: A Relational Model of Data for Large Shared Data Banks (1970), ibid.
[3] Denning P. J.: The Working Set Model for Program Behaviour (1968), ibid.
[4] Dennis J. B., Van Horn E. C.: Programming Semantics for Multiprogrammed Computations (1966), ibid.
[5] Dijkstra E. W.: Solution of a Problem in Concurrent Programming Control (1965), ibid.
[6] Dijkstra E. W.: The Structure of „THE" Multiprogramming System (1968) ibid.
[7] Earley J.: An Efficient Context — Free Parsing Algorithm (1970), ibid.
[8] Hoare C. A. R.: An Axiomatic Basis for Computer Programming, (1969), ibid.
[9] Hoare C. A. R.: Communicationg Sequential Processes (1978), ibid.
[10] Irons E. T.: A Syntax Directed Compiler for ALGOL 60 (1961), ibid.
[11] Kuki H., Cody W. J.: A Statistical Study of the Acurracy of Floating Point Number Systems (1973), ibid.
[12] Maurer W. D.: An Improved Hash Code for Scatter Storage (1968), ibid.
[13] Morris R.: Scatter Storage Techniques (1968), ibid.
[14] Metcalfe R. M., Boggs D. R.: Ethernet: Distributed Packet Switching for Local Computer Networks (1975), ibid.
[15] Parnas D. L.: A Technique for Software Module Specification with Examples (1972), ibid.
[16] Ritchie D. M., Thompson K.: The UNIX Time — Sharing System (1974), ibid.
[17] Rivest R. L., Shamir A., Adelman L.: A Method for Obtaining Digital Signatures and Public — Key Cryptosystems (1978), ibid.
[18] Samelson K., Bauer F. L.: Sequential Formula Translation (1960), ibid.
[19] Sussenguth E. H., Jr.: Use of Tree Structures for Processing Files (1963), ibid.
[20] Weizenbaum J.: ELIZA — A Computer Program for the Study of Natural Language Communication between Man and Machine (1966), ibid.
[21] Wirth N.: Program Development by Stepwise Refinement (1971), ibid.


ANDRZEJ SZAŁAS

## KONFERENCJE

# Międzynarodowy przepływ danych

IBI — Międzynarodowe Biuro Informatyki (Intergouvernmental Bureau for Informatics), będące agendą Organizacji Narodów Zjednoczonych, organizuje w dniach 26—29 czerwca 1984 r. w Rzymie II światową konferencję na temat polityki międzynarodowego przepływu danych (Second World Conference on Transborder Data Flow Policies).

Należy przypomnieć, że ekonomiczne, prawne i społeczno-kulturalne aspekty międzynarodowego przepływu danych przedstawiono po raz pierwszy w czerwcu 1980 r. na podobnej konferencji zorganizowanej przez IBI. W programie obecnej konferencji zaakcentowane zostaną te aspekty międzynarodowego przepływu danych, które szczególnie interesują kraje rozwijające się. Powinno to dopomóc tym krajom w zaprojektowaniu i przyjęciu właściwej strategii i polityki w tej dziedzinie. Innym celem konferencji będzie ustalenie uniwersalnych zasad międzynarodowego przepływu danych, a co najmniej sformułowanie wstępnych założeń do określenia takich zasad.

Organizatorzy podkreślają, że międzynarodowy przepływ danych jest jednym z najważniejszych elementów rozwoju informatyki międzynarodowej, a także — poprzez ułatwienie i przyspieszenie obiegu informacji — istotnym stymulatorem rozwoju międzynarodowej wymiany gospodarczej.

Szczegółowych informacji na temat konferencji udziela: **IBI, Department of Policies, P. O. Box 10253, 00144 Rome, ITALY.**

# Słownictwo z zakresu inżynierii oprogramowania

Wśród kilkuset terminów zdefiniowanych w normie IEEE Std 729 „Standard Glossary of Software Engineering Terminology" są i takie, których jeszcze nie używano w języku polskim, a to dlatego, że jak na razie niektóre pojęcia są raczej obce naszej praktyce zawodowej. Wydaje się, że warto je omówić w dziale terminologicznym INFORMATYKI, bo może w przyszłości nie będą stanowiły tylko pustych haseł w naszym języku.

Jednym z tych haseł jest **egoless programming**, co można przetłumaczyć jako **programowanie bezosobowe**, które według definicji stanowi podejście do opracowywania oprogramowania oparte na pojęciu grupowej odpowiedzialności za produkt programowy. Celem tego podejścia jest ochrona programisty przed zbytnim utożsamieniem się z wynikiem swojej pracy, co uniemożliwia mu obiektywną ocenę tego wyniku. Zwróćmy uwagę, że w dążeniu do doskonałości rezygnuje się w poważnym stopniu z jednoosobowej odpowiedzialności za wynik pracy!

Oczywiście, podanie definicji terminu w normie oznacza na ogół, że ma on rację bytu i jest dość trwałym elementem kultury językowej. Dlatego dużą niespodzianką dla nas jest zatem zdefiniowanie zarówno wymienionego jak i następnego terminu — **chief programmer team** — obu raczej nie znanych naszemu systemowi pojęć informatycznych. Chief programmer team czyli **zespół głównego programisty** jest to grupa osób powołana do opracowania oprogramowania, złożona z głównego programisty, jego zastępcy, sekretarza (bibliotekarza) oraz dodatkowych programistów i innych specjalistów, którzy stosują pomocnicze procedury do polepszenia wzajemnej łączności i optymalnego wykorzystania własnych umiejętności.

Definicja terminu **główny programista** (ang. **chief programmer**) stanowi przykład jak łatwo się zgubić w tworzeniu systemu pojęć: główny programista jest to kierownik zespołu głównego programisty. Na szczęście, wyjaśnienie terminu jest bardziej zrozumiałe od jego definicji: doświadczony programista, którego obowiązki polegają na wytworzeniu zasadniczej części oprogramowania tworzonego przez zespół, koordynowaniu działalności zespołu, dokonywaniu przeglądu prac innych członków zespołu i znajomości opracowywanego oprogramowania pod względem technicznym. **Zastępca głównego programisty** (ang. **backup programmer**) ma mniej więcej te same obowiązki, tzn. wnosi istotny wkład w tworzenie oprogramowania, pomaga głównemu programiście w dokonywaniu przeglądu prac innych członków zespołu, zastępuje go gdy zachodzi potrzeba i, oczywiście, wykazuje techniczną znajomość opracowywanego oprogramowania (w myśl zasady „co dwie głowy to nie jedna"). Rola sekretarza (bibliotekarza) w zespole głównego programisty polega na założeniu, kontrolowaniu i pielęgnowaniu biblioteki programów. Warto dodać, że według omawianej normy **biblioteka programów** (ang. **software library**) jest to kontrolowany zbiór programów i odpowiedniej dokumentacji, służący do tworzenia, używania i pielęgnacji oprogramowania.

Czy tak zorganizowany zespół wystarcza do wytworzenia doskonałego oprogramowania? Tak, jeśli wie, że musi mieć **project notebook**, a dokładniej **software development notebook. Project notebook** czyli **kartoteka projektu** jest to centralna zbiornica materiałów pisemnych dotyczących projektu, takich jak plany, raporty, notatki służbowe itp. **Software development notebook** czyli **kartoteka rozwoju oprogramowania** stanowi zbiór materiałów dotyczących wytwarzania określonego modułu programowego.

Praca zespołu wytwarzającego oprogramowanie podlega okresowej kontroli na poszczególnych etapach cyklu rozwojowego produktu. **Milestone** czyli **zakończenie etapu** jest to zaplanowane wydarzenie służące do oceny postępu prac, za które odpowiada członek lub kierownik grupy projektowej. Przykładami takich wydarzeń są: przegląd formalny (ang. formal review), sporządzenie specyfikacji, dostawa produktu.

Na ogół, zdajemy sobie sprawę, że w procesie wytwarzania oprogramowania istotną rolę odgrywają narzędzia. **Narzędziem programowym** (ang. **software tool**) nazywa się program komputerowy stosowany do opracowania, testowania, analizy lub pielęgnacji innego programu albo jego dokumentacji. Zintegrowany zestaw narzędzi dostępnych przy użyciu jednego języka poleceń, służący do wspomagania możliwości programowych przez cały cykl życia programu, nazywa się **środowiskiem programowym** (ang. **program support environment**). Typowe środowisko składa się z narzędzi do projektowania, redagowania, kompilowania, ładowania, testowania programu oraz zarządzania konfiguracją i projektem.

Ze środowiska wyodrębnia się na ogół tzw. **łoże testowe** (ang. **test bed**), tj. środowisko testowe złożone ze sprzętu, oprzyrządowania programowego, symulatorów i innego oprogramowania pomocniczego, niezbędnego do testowania systemu lub jego składników. **Oprzyrządowanie programowe** (ang. **instrumentation tool**) jest to narzędzie programowe, które ustawia liczniki lub inne wskaźniki (ang. **probes**) w kluczowych miejscach innego programu, w celu dostarczenia danych statystycznych o jego wykonaniu.

Rola jednego z ciekawszych narzędzi, którego w naszych warunkach raczej się nie stosuje, polega na **zarządzaniu konfiguracją** (ang. **configuration management**). Warto więc wiedzieć, że według wymienionej normy zarządzanie konfiguracją jest to proces identyfikowania i definiowania elementów konfiguracji systemu, kontrolowania ich odmian przez cały czas życia systemu, rejestrowania stanu elementów konfiguracji i żądania zmian, weryfikowania pełności i poprawności elementów konfiguracji. **Elementem konfiguracji** (ang. **configuration item**) nazywa się zbiór jednostek sprzętowych i programowych traktowanych jako całość do celów zarządzania konfiguracją.

Oprócz narzędzi, wytwórcy oprogramowania stosują w pracy także dobrze określone metody organizacyjne, mające na celu polepszenie jakości produktu finalnego. Tych metod jest dość dużo, a spośród zdefiniowanych w normie warto podać dwie. **Inspection** jest to **formalna metoda oceny oprogramowania**, w której wymagania, projekt lub kod programu są szczegółowo sprawdzane przez pojedynczą osobę lub grupę osób, z wyłączeniem autorów, w celu wykrycia defektów, niezgodności z normami lub innych istotnych szczegółów. **Walk-through** jest to **przegląd** polegający na tym, że projektant lub programista przedstawia fragment opracowanego przez siebie projektu lub kodu jednemu lub kilku członkom zespołu, którzy stawiają pytania i wnoszą uwagi dotyczące metody, stylu, możliwych błędów, niezgodności z normami i innych istotnych szczegółów.

Miał rację Claude Levi Strauss twierdząc, że język kształtuje świadomość. Oby nam ukształtował informatyczną.

**JANUSZ ZALEWSKI**

■ Oprogramowanie pracujące w systemie CP/M będzie dostępne dla wszystkich komputerów osobistych zbudowanych w nowym standardzie MSX. Umożliwi to MSX-DOS, 8-bitowy system operacyjny napisany przez MICROSOFT. Ponad dwudziestu japońskich producentów, którzy podjęli się wytwarzania komputerów zgodnych z opracowanym w MICROSOFT standardem sprzętu i oprogramowania MSX, otrzymało ten system w IV kwartale ubiegłego roku. MSX-DOS używa formatu dyskowego zgodnego z dotychczasowym „przebojem" MICROSOFT — systemem MS-DOS (dokładnie — z wersją używaną w IBM PERSONAL COMPUTER). Jednocześnie użytkownik będzie miał możliwość odczytywania dysków zapisanych w formacie CP/M i jednokierunkowej konwersji z tego formatu MS-DOS. Kazuhiko Nishi, wiceprezes firmy ASCII CORP., która reprezentuje MICROSOFT w Japonii, twierdzi, że opracowano także nowe wersje interpreterów i kompilatorów dotychczas stworzonych przez MICROSOFT, jak również programów zastosowaniowych — np. Multiplan czy Microsoft Word.

*

■ Już w końcu roku rozpoczęły się dostawy systemów pamięci na dysku optycznym, opracowanych w firmie STORAGE TECHNOLOGY CORP. OSS 7600 pozwala techniką laserową zapisywać i odczytywać 4 G bajty na jednostronnym dysku o średnicy 14 cali, przy prędkości transmisji 3 M bajty/s. Każdy dysk — dla którego czas przechowywania określa się na 10 lat, w porównaniu z 1—3 lat dla tradycyjnej taśmy magnetycznej — może zawierać równowartość 2 mln stron maszynopisu, tzn. tyle co 40 szpul taśmy. Cena: ok. 130 tys. dol.

*

■ Firma DIGITAL EQUIPMENT CORP. wprowadziła na rynek nowe pakiety procesorowe. LSI-11/73 ma zapewnić czterokrotnie większą przepustowość niż LSI-11/23, tzn. jego osiągi będą na poziomie minikomputera PDP-11/44. Jest to jednocześnie pierwsze zastosowanie wytwarzanego przez DEC w technologii CMOS mikroprocesora MICRO/J11, który łączy 16-bitowe szyny zewnętrzne z 32-bitową pracą wewnętrzną. Pakiet KXT11-C służy do usprawnienia działania większych instalacji, spełniając funkcje procesora peryferyjnego. Ostatni, Falcon-Plus, zaprojektowany do zastosowań czasu rzeczywistego, wymagających tanich, niewielkich komputerów, pracuje w oparciu o programy zapisane w pamięci ROM. Wszystkie trzy pakiety wyposażone są w standardową dla wyrobów DEC magistralę Q-bus.

*

■ IBM przekroczy prawdopodobnie w tym roku barierę 40 mld dolarów obrotów i 5 mld zysków. Wskazują na to wyniki uzyskane w pierwszym półroczu 1983. W okresie tym, przy obrotach 17 877 mln dol. (wzrost o 18% w stosunku do ub.r.), koncern osiągnął 2320 mln dol. zysku (24% wzrostu). Warto tu wspomnieć, że w wielu innych czołowych firmach związanych z informatyką proporcje te są odwrotne, tzn. nawet przy wysokim tempie wzrostu obrotów obserwuje się spadek dynamiki wzrostu zysków. W IBM można zauważyć również zmianę proporcji udziału poszczególnych sektorów — ponad połowa (9327 mln dol./ pochodzi ze sprzedaży sprzętu. Głównie dzięki systemom dyskowym 3380, komputerom serii 308X oraz komputerom osobistym PC i XT osiągnięto w tej dziedzinie wzrost aż o 47%. Natomiast wyraźny spadek (13% w stosunku do roku 1982) zanotowano w dziale wynajmowania, choć nadal obroty w nim są znaczące — 4982 mln dol. Pozostała część obrotów — 3588 mln dol. — przypadała na usługi, których wzrost utrzymywał się na średnim poziomie (17%).

*

■ Od 19 do 24 września 1983 Poznań był miejscem spotkania specjalistów z dziedziny baz danych, zorganizowanego przez Środowiskowy Ośrodek Informatyki Politechniki Poznańskiej przy współudziale Komitetu Naukowo-Technicznego ds. Informatyki Rady Wojewódzkiej NOT oraz Komisji Zastosowań Komputerów Komitetu Danych dla Nauki i Techniki PAN (Polskiego Komitetu Narodowego CODATA). Seminarium było również związane z pracami prowadzonymi w ramach Rady ds. Zastosowań Środków Techniki Obliczeniowej Komisji Międzyrządowej ds. ETO, co spowodowało udział i wygłoszenie referatów przez gości z Węgier i NRD. Natomiast z kraju najsilniej, oprócz gospodarzy, reprezentowane były środowiska wrocławskie i krakowskie. Wygłoszono ponad 30 referatów (organizator przewiduje ich opublikowanie), zgrupowanych wokół tematu tego spotkania: „Problemy tworzenia centrów danych dla nauki i techniki (projektowanie i implementacja baz danych)". Omówiono doświadczenia we wprowadzaniu systemu RODAN w przedsiębiorstwie, a także próby śledzenia pewnych procesów myślowych człowieka — na potrzeby algorytmizacji wstępnych etapów prac nad bazami danych.

*

■ Na początku listopada ub.r. odbyła się oficjalna premiera mikrokomputera domowego IBM PERSONAL COMPUTER JUNIOR, który dotąd był określany komputerem PEANUT. Przedstawiono od razu dwie wersje: Pierwsza ma 64 KB pamięci operacyjnej, jednostkę pamięci kasetowej i kosztuje w USA 669 dol. Druga, za 1269 dol., ma dwukrotnie większą pamięć operacyjną i jednostkę dyskietek. W obu — sercem jest mikroprocesor INTEL 8038, a jego pracą steruje odpowiednio przystosowana wersja systemu operacyjnego MS-DOS. Zapewnia to pełną zgodność programową ze starszym bratem — IBM PC. Dodatkowo PC Jr może być wyposażony w modem, który umożliwia łączność z innymi systemami. Ciekawostką jest bezprzewodowe (podczerwień) połączenie jednostki centralnej z klawiaturą, nawet z odległości 6 metrów. Ponieważ prace nad JUNIOREM opóźniły się i produkcja seryjna ruszą w I kwartale 1984 ominie on więc okres prezentów (Gwiazdka!). Jednak wysoka w tej klasie cena może wskazywać, że IBM i tak nie liczył by jego wyroby trafiły pod choinkę. (M)

*

■ Rząd Stanów Zjednoczonych postanowił do roku 1988 skomputeryzować prawie wszystkie działania władz federalnych. Program ten o nazwie REFORM 88 składa się z dziewięciu kierunków prac i ma zaoszczędzić w ciągu sześciu lat 2000 dol. na jednego mieszkańca USA. Za najważniejsze projekty uważa się pełną automatyzację prac nad budżetem, rozszerzenie sieci telekomunikacyjnej do potrzeb poczty elektronicznej pomiędzy ministerstwami i wewnątrz nich, a także standaryzację 332 niezgodnych ze sobą systemów księgowości. Inny projekt — modernizacji systemu rozliczeń — ma sam zaoszczędzić 2 do 3 mld dol. w ciągu trzech lat. Tak więc program REFORM 88 można nazwać samofinansującym się. (M)

*

Jedni wchodzą na rynek, inni muszą go opuścić. Firma TEXAS INSTRUMENTS najpierw zrezygnowała z planów nowego komputera domowego TI-99/8. Obecnie musiała zaprzestać produkcji znanego TI-99/4A. Tak więc pierwszy z wielkich tego działu musiał się wycofać. Fachowcy twierdzą przy tym, że na pewno nie jest to ostatni. Resztki TI-99/4A będą sprzedane po 59 dol. (tzn. z obniżką 30 dol.), a firma zapowiada dalszą walkę w dziedzinie mikrokomputerów profesjonalnych. (M)

*

■ W Bratysławie odbyła się we wrześniu ub. r. międzynarodowa konferencja COMPCONTROL '83 poświęcona zastosowaniom techniki komputerowej w przemyśle maszynowym. Obok delegacji z państw RWPG w konferencji uczestniczyli przedstawiciele Danii, Francji, Japonii, Norwegii, RFN i Stanów Zjednoczonych. Zgłoszono 34 tradycyjne referaty oraz 79 referatów-plansz (w tym niemal 20 z Polski). Zainteresowani obszernymi materiałami z konferencji mogą się z nimi zapoznać w Ośrodku Doskonalenia Kadr SIMP w Warszawie. (S)

*

■ Balatonszeplak (Węgry) było miejscem posiedzenia stałej Komisji Telekomunikacji i Przemysłu Elektronicznego RWPG, w której zasiadają ministrowie odpowiednich resortów lub ich zastępcy z poszczególnych krajów członkowskich. Na spotkaniu postanowiono utworzyć jednolitą bazę podzespołów, przeprowadzić standaryzację technologii oraz rozszerzyć specjalizację i współpracę naukowo-techniczną. Z konkretnych przedsięwzięć na wspomnienie zasługują: opracowanie nowej generacji odbiorników telewizyjnych, sprzętu studyjnego, telewizji przewodowej, magnetowidów i gramowidów, ujednolicenie systemów central telefonicznych i transmisji cyfrowej, a także podjęcie produkcji bardzo czystych metali i związków chemicznych na potrzeby przemysłu elektronicznego. (M)

**Fuglewicz P.: MODULA-2 — język lat osiemdziesiątych (1). Charakterystyka języka**

INFORMATYKA 1984, nr 1, s. 2

Pierwsza część omówienia języka MODULA-2, zawierająca genezę jego powstania, podstawowe właściwości ilustrowane prostymi przykładami, porównanie z innymi językami oraz perspektywy zastosowania.

**Фуглевич П.: MODULA-2 — язык восьмидесятых годов (I). Характеристика языка**

INFORMATYKA 1984, № 1, стр. 2

Первая часть обсуждения языка MODULA-2, содержащая описание происхождения языка и основных свойств иллюсрированных простыми примерами, а также сравнение с другими языками и перспективы применений.

**Małuszyński J. T.: Odtwarzanie bazy danych techniką dzienników częściowch**

INFORMATYKA 1984, nr 1, s. 6

Charakterystyka techniki dzienników częściowych, ilustrowana przykładem rozwiązania w systemie odtwarzania bazy danych CRASH, współdziałającym z systemem zarządzania bazą danych dla komputera DEC 10. Omówiono zalety i wady tej techniki w porównaniu do „klasycznej" techniki dzienników.

**Малушиньски Й. Т.: Восстановление базы данных техникой частичных журналов**

INFORMATYKA 1984, № 1, стр. 6

Характеристика техники частичных журналов иллюстрированная примеров решений в системе восстановления базы данных CRASH взаимодействующей с системой управления базой данных для ЭВМ DEC 10. Обсуждено преимущества и недостатки этой техники по сравнению с „классической" техникой журналов.

**Paprzycki M., Urbaniec K.: Ujednolicenie komputerowego wspomagania prac inżynierskich**

INFORMATYKA 1984, nr 1, s. 10

Omówienie rozwoju komputerowego wspomagania prac inżynierskich na świecie, a także aktualnej sytuacji krajowej w oparciu o przeprowadzone badania ankietowe. Wskazano na pilną konieczność normalizacji rozwiązań oraz określono najpilniejsze zadania w tej dziedzinie.

**Папшицки М., Урбанец К.: Машинная поддержка инженерных работ**

INFORMATYKA 1984, № 1, стр. 10

Обсуждение развития машинной поддержки инженерных работ в мире, а также настоящего положения в стране на базе проведенных анкетных исследований. Указана срочная необходимость стандартизации решений и определены наиболее неотложные задачи в этой области.

**Fuglewicz P.: MODULA-2 — a language for 80's (1). Characteristics of the language**

INFORMATYKA 1984, No. 1, p. 2

First part of MODULA-2 presentation, which includes origin of its creation, main features, illustrated with simple examples, comparison with other programming languages and prospects of its application.

**Fuglewicz P.: MODULA-2 — eine Sprache achtziger Jahre (1). Eine Charakteristik der Sprache**

INFORMATYKA 1984, Nr. 1, S. 2

Erster Teil einer Präsentierung von MODULA-2, der ihre Genesis, wichtigste Eigenschaften, mit einfachen Beispielen illustriert, sowie ein Vergleich mit anderen Programmiersprachen und Anwendungsaussichten umfasst.

**Małuszyński J. T.: Data base recovery by logical logging technique**

INFORMATYKA 1984, No. 1, p. 6

Characteristics of logical logging technique, illustrated with a solution example of CRASH data base recovery system, which cooperates with data base management system for DEC 10 computer. Advantages and disadvantages of this technique in comparison with „classic" logging technique are discussed.

**Małuszyński J. T.: Wiederbildung einer Datenbank mittels Teiltagesbüchertechnik**

INFORMATYKA 1984, Nr. 1, S. 6

Eine Charakteristik der Tagesbüchertechnik mit einem Lösungsbeispiel von CRASH Datenbankwiederbildungssystem, das mit Datenbankverwaltungssystem für DEC 10 Rechner zusammenwirkt. Es wurden Vor-und Nachteile dieser Technik im Vergleich mit „klassischer" Tagesbüchertechnik besprochen.

**Paprzycki M., Urbaniec K.: Computer assisted design**

INFORMATYKA 1984, No. 1, p. 10

World-wide development of computer assisted design, as well as actual home situation based on inquiry research is discussed. Urgent standarization is pointed out and most important tasks in this area are determined.

**Paprzycki M., Urbaniec K.: Rechnerunterstützte Projektierung**

INFORMATYKA 1984, Nr. 1, S. 10

Elne Besprechung der weltweiten Entwicklung von rechnerunterstützten Projektierung, sowie der heutigen inländischen Lage auf Grund der durchgeführten Umfrage. Es wurde dringende Notwendigkeit der Normierung von Lösungen betont und wichtigste Aufgaben in diesen Bereich bezeichnet.

# Czy warto pisać

Czy uda się INFORMATYCE stworzyć środowisko... Wysiłek wart jest podjęcia. Zmiany na okładkach, nieco zmniejszona objętość (to już wbrew woli — raczej dlatego, że nie ma właśnie środowiska), zmiany stylu pracy redakcji i te wszystkie plany, gorąco dyskutowane na kolegiach — może wreszcie skupią czytelników wokół i c h pisma.

Nie jest dla mnie jasne, czy ludzie, których łączy jedynie wspólny zawód, mogą się czuć na tyle zintegrowani, że potrzebne im jest czasopismo środowiskowe. A sądzę, że INFORMATYKA pismem tylko czysto fachowym stać się nie może: choć — swoją drogą — powinna być ona nadal zawodową pomocą.

Dotychczas brakuje w INFORMATYCE odbicia tego, co się dzieje z polską informatyką, gdzie rzeczywiście ma sukcesy, gdzie klęski i dlaczego. Reportaż, wywiad czy zamówiony artykuł opisujący kawałek rzeczywistości, ale nie w jubileuszowych koniecznie barwach, powinien ożywić dotychczasowy dział z kraju. Tu udział czytelników — korespondencje, sygnały czy propozycje — ma największe znaczenie. Jeśli to nie wyjdzie — redakcja śmiało może podzielić się „zasługą".

Często powtarzany jest truizm, że kształt i poziom czasopisma zależy od tych, którzy je czytają. To prawda — choć nie bezpośrednio, nie dlatego, że czytelnicy w obliczu nie najlepszych (lub doskonałych) artykułów chwytają za pióra, aby napisać lepsze. Nie, rzecz nie w tym. Czasopismu potrzebni są krytycy, czytelnicy, którzy chcieliby napisać o n i m — choćby nawet złośliwie (karcąco?). Nie łudźmy się, cios krytyki nie zbawi natychmiast, jest jednak dla redakcji (redaktorów) zdrowym prysznicem, który pobudza do żywszego myślenia. Nie ma bowiem dla żadnej grupy, a kolegium redakcyjne jest też tylko grupą ludzi, gorszej sytuacji niż zamknięcie w wieży z kości słoniowej.

Nie pomagają, nieliczne zresztą, spotkania z czytelnikami, szczególnie wówczas, gdy jeszcze brak środowiska wokół pisma. Nawet odnawianie krwi w redakcji niewiele zmienia — nowe szybko daje się uładzić, temperatura nie pobudzana z zewnątrz staje się średnia i pozostają co najwyżej redakcyjne przepychanki, albo — co najgorsze — redakcyjne uprzejmości.

No dobrze, pomyśli czytelnik, a co mnie obchodzi sytuacja INFORMATYKI, cóż dadzą mi starania o to środowisko i ewentualny wzlot czasopisma. Cóż, nie wiem, czy da cokolwiek. Nie dziwmy się jednak, że grupa zawodowa, a w efekcie reprezentowana przez nią dziedzina, nie mają w trudnych czasach widoków na lepsze perspektywy, jeśli nawet nie wierzy w tak prostą rzecz, jaką jest skupienie się i stworzenie zaczynu właśnie środowiska (a przy okazji — stałe wychowywanie redakcji swojego, bądź co bądź, czasopisma).

Wyobraźmy sobie przez chwilę, że jednak się nam udało, że jesteśmy zwarci i mamy swoją, wspólnie tworzoną trybunę i forum zawodowe. I co dalej. Ano znowu nie wiem, może nadal nie będzie się naszej specjalności dobrze wiodło; zapewne nie trybuna o tym zadecyduje.

Ale będziemy mieli wówczas nie tylko pewność, że nie zaniedbaliśmy tak w końcu łatwego działania, ale również swoje czasopismo — tym bardziej przydatne, im więcej osób spośród nas poświęci mu nieco uwagi.

Być może to co napisałem brzmi, jak mdłe, pozytywistyczne, często teraz powtarzane apele o dobrą robotę, o aktywność — ale na swoim podwórku! Owszem, na swoim, znanym, ale nie takim znów maleńkim. Jest nas, żyjących z informatyki, jeszcze ponad czterdzieści tysięcy. Od tego, co potrafimy zrobić, a raczej — czy potrafimy przetrwać w jakim takim zdrowiu sporo zależy...

Każdy z nas coś robi, gdzieś działa, wypowiada mniej czy bardziej spontaniczne opinie. Mają one jednak dość ograniczony zasięg, a tym samym niewielką szansę zderzenia się lub współbrzmienia z innymi. Skutek tego to także brak rzetelnej analizy sytuacji informatyki w Polsce, całkowicie rozbieżne zdania o tym, czy jest czy nie jest niezbędna, czy daje coś czy tylko pochłania środki, czy jest z nią w końcu tak źle czy może tylko średnio. Choćby tylko o tym warto szerzej podyskutować. Z jasnej oceny rzeczywistości znacznie łatwiej wysnuwać sensowne prognozy, budować zamierzenia. Czy INFORMATYKA w tym pomoże? Zobaczymy. W każdym razie ma taki zamiar.

JANUSZ GWIAZDA

# Kalendarz

**Styczeń**

25—27, Paryż: IV kongres na temat sztucznej inteligencji — organizator: AFCET

**Luty**

14—17, Berlin Zachodni: VII kongres i wystawa sprzętu „Online '81" — organizator: Online GmbH

**Marzec**

12—15, Las Vegas (USA): XII konferencja „Interface '81" na temat transmisji i przetwarzania danych — organizator: The Interface Group, Inc.
19—21, San Diego (USA): międzynarodowa konferencja na temat przetwarzania mowy i sygnałów — organizator IEEE
21—23, Zurych (Szwajcaria): międzynarodowe sympozjum na temat wydajności komputerowych systemów telekomunikacyjnych — organizator: IBM Zürich Research Laboratory
26—28, Karlsruhe (RFN): konferencja na temat architektury i eksploatacji systemów komputerowych — organizator: Gesellschaft für Informatik
26—29, Orlando (USA): VII międzynarodowa konferencja na temat inżynierii oprogramowania — organizator: IEEE

**Kwiecień**

11—13, Paryż: STACS — konferencja na temat teoretycznych aspektów informatyki — organizator: AFCET oraz Gesellschaft für Informatik
17—19, Tuluza (Francja): VI międzynarodowe kolokwium na temat programowania — organizator: Université Paul Sabatier
23—25, Pitsburg (USA): konferencja na temat rozwoju oprogramowania użytkowego — organizator: ACM Sigsoft/Sigplan

**Maj**

14—17, Amsterdam (Holandia): międzynarodowa konferencja telekomunikacyjna IEEE — organizator: TACM

**Czerwiec**

4—6, Nicea (Francja): kolokwium na temat predyspozycji w programowaniu — organizator: AFCET
7—8, Sophia—Antipolis (Francja): konferencja na temat jakościowej oceny predyspozycji w programowaniu — organizator: AFCET
17—21, Kopenhaga: XXVI międzynarodowa konferencja TIMS — organizator: The Institute of Management Sciences

# Zasady prenumeraty

Zamówienia i przedpłaty na prenumeratę INFORMATYKI przyjmuje Zakład Kolportażu Wydawnictwa NOT SIGMA. Adres pocztowy: Wydawnictwo NOT SIGMA — Zakład Kolportażu, 00-950 Warszawa, skr. poczt. 1004. Konto bankowe: 1036-7490-139-11, III O/M NBP w Warszawie.

JEDNOSTKI GOSPODARKI USPOŁECZNIONEJ, INSTYTUCJE I ORGANIZACJE przesyłają zamówienia (w 1 egz.) zawierające: tytuł czasopisma, liczbę zamawianych egzemplarzy, okres prenumeraty i pełny adres zamawiającego z kodem pocztowym, oddział i nazwę banku z numerem konta bankowego zamawiającego oraz (ewentualnie) adresy odbiorców, którzy na zlecenie i koszt zamawiającego mają egzemplarze otrzymywać.

Warunkiem realizacji zamówienia jest równoczesne dokonanie odpowiedniej wpłaty na ww. konto Wydawnictwa NOT SIGMA.

Za prenumeratę nie wystawiane są rachunki i nie potwierdzane salda. Prenumeratorzy zbiorowi proszeni są o podawanie na dowodach wpłat (przelewach) znaku kancelaryjnego zamówienia, którego dotyczy wpłata.

Dopisując na zamówieniu PRENUMERATA STAŁA, zamawiający (tylko prenumeratorzy zbiorowi!) nie będą musieli corocznie ponawiać zamówienia, a jedynie dokonywać przedpłaty według aktualnie obowiązujących cen. Wydawnictwo przekazywać będzie co roku potwierdzenie kontynuacji prenumeraty.

PRENUMERATORZY INDYWIDUALNI dokonują wpłaty przekazem na ww. konto, pod powyższym adresem, podając na odwrocie odcinka dla adresata—posiadacza rachunku: tytuł czasopisma, liczbę zamawianych egzemplarzy oraz okres prenumeraty.

Do PRENUMERATY ULGOWEJ upoważnieni są członkowie stowarzyszeń naukowo-technicznych NOT, studenci, uczniowie szkół zawodowych. Warunkiem jej uzyskania jest poświadczenie blankietu dla nabywcy indywidualnego (na odcinku dla adresata) przez właściwe stowarzyszenie NOT, wyższą uczelnię lub szkołę zawodową.

Zamówienia i wpłaty przyjmowane są na okresy kwartalne, półroczne i roczne w terminach:

- do 15 listopada — na I kwartał, I półrocze i cały rok
- do 28 lutego — na II, III i IV kwartał
- do 31 maja — na II półrocze i IV kwartał.
- do 31 sierpnia — na IV kwartał.

Uwaga: Przy podawaniu kodu pocztowego i numeru konta bankowego obowiązuje bardzo czytelne pismo. Prenumerata nie wymaga specjalnego przekazu z czerwonym paskiem; wystarczy zwykły przekaz bankowy.

Prenumerata normalna: kwartalna — 225 zł, półroczna — 450 zł, roczna — 900 zł. Prenumerata ulgowa: kwartalna — 150 zł, półroczna 300 zł, roczna — 600 zł. Prenumerata ze zleceniem wysyłki za granicę jest dwukrotnie droższa.

Dodatkowych informacji o prenumeracie udziela: Zakład Kolportażu, tel. 40-00-21 w. 293, 299 oraz 40-35-89. Egzemplarze archiwalne można nabywać w Klubie Prasy i Informacji Technicznej w Warszawie, ul. Mazowiecka 12, tel. 27-43-65. Zamówienia na egzemplarze archiwalne należy kierować pod adresem Zakładu Kolportażu.

W najbliższych numerach:

- Jacek Owczarczyk o interakcyjnym systemie przetwarzania obrazów
- Jakub Tatarkiewicz o kodach kreskowych
- Marek Sobczyk i Andrzej Szałas o PROLOGU
- Jacek Mirkowski i Adam Piątkowski o laboratorium dydaktycznym z rozłożoną inteligencją
- Józef B. Lewoc o perspektywie Jednolitego Systemu
- Roman Żelazny o narzędziach inżynierii oprogramowania
- Witold Abramowicz o MODULI-2 i LILITH
- Teresa Wilczek i Cezary Zieliński o robotyce
- Jerzy Szyller o mikroprocesorach lat osiemdziesiątych

W każdym numerze — mikroKLAN