8 1988

informatyka

Prof. W. M. Turski o kanonicznym kroku procesu programowania

Bazy danych w biurach

Realizacja przetwarzania obrazów

WYDAWNICTWO NOT SIGMA

Cena 250 zł

informatyka

8 1988

Nr 8

Miesięcznik Rok XXIII
Sierpień 1988

Organ Komitetu
Naukowo-Technicznego NOT
ds. Informatyki

Materiałów nie zamówionych redakcja nie zwraca

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

RSW „PRASA-KSIĄŻKA-RUCH"
PRASOWE ZAKŁADY GRAFICZNE
ul. Dworcowa 13, 85-950 BYDGOSZCZ

Zam. 2414/88. E-8
Obj. 4,0 ark. druk. Nakł. 9000 egz.

ISSN 0542-9951. INDEKS 36124

Cena egzemplarza 250 zł

WYDAWNICTWO NACZELNA ORGANIZACJA TECHNICZNA
SIGMA
CZASOPISM I KSIĄŻEK TECHNICZNYCH

00-950 Warszawa
skrytka pocztowa 1004
ul. Biała 4

W NUMERZE:

W NAJBLIŻSZYCH NUMERACH:

- **Wojciech Mokrzycki dokonuje przeglądu technik grafiki komputerowej**
- **Danuta Kruszewska charakteryzuje system operacyjny IPIX, opracowany dla komputerów typu IBM PC w Instytucie Podstaw Informatyki PAN w Warszawie**
- **Roman Podraza przedstawia charakterystykę zasad funkcjonowania systemów sterowania przepływowego oraz opisuje model takiego sterowania, nadający się do konstruowania procesora obliczeń funkcjonalnych dla systemu o zastosowaniach uniwersalnych**
- **Jerzy Franczyk opisuje Język Modlan do projektowania układów cyfrowych, opracowany w Instytucie Elektroniki Politechniki Śląskiej**
- **Janusz Zalewski prezentuje protokół komunikacyjny Kermit, opracowany specjalnie do przesyłania plików**
- **Jan Ryżko dokonuje przeglądu rynku drukarek**

WŁADYSŁAW M. TURSKI
Instytut Informatyki
Uniwersytet Warszawski

POLSKIE TOWARZYSTWO INFORMATYCZNE

Pierwsza wersja artykułu została przedstawiona podczas Szkoły Jesiennej PTI „Współczesne kierunki rozwoju informatyki", Mrągowo, 7–11 grudnia 1987 r.

Kanoniczny krok procesu programowania (1)

W większości znanych metod budowy oprogramowania wyróżnia się kilka odrębnych faz jego projektowania i realizacji. Każdą z wyróżnionych faz ma do spełnienia specyficzne zadanie (np. ustalenie wymagań funkcjonalnych czy określenie przepływu danych między modułami projektowanego oprogramowania). Z zadaniem realizowanym w danej fazie wiążą się wyspecjalizowane metody postępowania, języki opisu i narzędzia pracy, w tym – specyficzne dla danej fazy oprogramowanie narzędziowe. Po dostarczeniu klientowi gotowego oprogramowania użytkowego (i, ewentualnie, po uruchomieniu tego oprogramowania) rozpoczyna się faza pielęgnacji (ang. maintenance), trwająca tym dłużej im bardziej udany jest dostarczony produkt programistyczny. Analiza kosztów poszczególnych faz cyklu życia takiego produktu (od wstępnego projektu do ostatecznego zaniechania użytkowania gotowego wyrobu) wskazuje, że ponad 3/4 wydatków pochłania faza pielęgnacji. W dużej mierze wynika to z całkowitej prawie nietechnologiczności tej fazy, dla której nie tworzy się matematycznie uzasadnionych metod postępowania, specjalnych języków opisu ani narzędzi pracy.

Okazuje się jednak, że wszystkie znane fazy projektowania, realizacji i pielęgnacji oprogramowania można wyrazić jako proste kombinacje ciągów kroków kanonicznych i elementarnej operacji wycofywania się [5]. Przyjęcie takiego modelu cyklu życia oprogramowania pozwala skupić wysiłek badawczy i konstrukcyjny na metodach postępowania, językach opisu i narzędziach pracy przydatnych w kroku kanonicznym; uzyskane wyniki (teoretyczne i praktyczne) znajdują bezpośrednie i natychmiastowe zastosowanie w całym cyklu życia oprogramowania. Ceną stosowania metod opartych na koncepcji kroków kanonicznych jest wydatne zwiększenie ilości prac pomocniczych (np. dokumentacyjnych), która rośnie co najmniej liniowo z liczbą kroków, a przy stosowaniu metody kroków kanonicznych liczba wykonywanych kroków jednolitego procesu projektowania–realizacji–pielęgnacji oprogramowania jest zazwyczaj bardzo wielka. Dlatego też stosowanie takich metod wymaga bardzo sprawnego wspomagania technicznego. W praktyce oznacza to konieczność stosowania tzw. zespolonych środowisk programistycznych IPSE (ang. integrated programming support environments).

WPROWADZENIE[1)]

Przedstawienie metody kroków kanonicznych wymaga ustalenia poglądu na to, czym właściwie jest program i specyfikacja oraz na czym polega proces tworzenia programu użytkowego i jego dalsza ewolucja w okresie stosowania. Niezbędne informacje wstępne przedstawię z konieczności skrótowo, odsyłając czytelników do pracy [6] i książki [10] po więcej szczegółów.

Z czysto pragmatycznego punktu widzenia przyjmujemy, że program użytkowy powstaje po to, aby przy jego pomocy można było rozwiązywać pewne zadania odnoszące się do jakiegoś aspektu realnego świata. Opis zadania musi więc nawiązywać do realiów innych niż komputerowe (bywają oczywiście wyjątki, ale zdarzają się one stosunkowo rzadko: np. wtedy, gdy zadanie dotyczy organizacji obliczeń lub prezentacji wyników). Zbudowany już program komputerowy jest tworem czysto formalnym, którego sens wynika z przyjętej interpretacji składających się nań symboli i ich kombinacji (to, w jaki sposób przypisuje się ów sens programowi, zależy od wyboru semantyki: np. operacyjnej czy denotacyjnej, nie ma to jednak wpływu na generalny charakter związku programu z jego znaczeniem).

Mamy więc do czynienia z taką sytuacją: z jednej strony realia wybranych aspektów pewnej dziedziny świata rzeczywistego, odzwierciedlone w opisie zadania, z drugiej zaś – komputerowy program i jego znaczenie, twory z natury rzeczy formalne. Nie wdając się w niepotrzebną tu dyskusję o wyższości tych czy innych rodzajów semantyki, warto zauważyć, że wyniki wykonania programu na sprawnym komputerze wynikają ściśle kalkulacyjnie – a więc w sposób formalny! – ze znaczenia programu.

Informatyczne rozwiązanie problemu prowadzi przeto od dziedzin realnych do formalnych; w praktycznym postępowaniu styk tych dziedzin jest nieunikniony. Z wielu względów wygodnie jest przyjąć, że styk ten następuje w specyfikacji, którą traktujemy jako teorię o dwu modelach – w świecie rzeczywistym i w świecie programów. Różnice między nimi mogą być pokaźne. Jako teoria świata realnego, specyfikacja podlega rygorom nauk przyrodniczych, m. in. jej słuszność podlega sprawdzeniu doświadczalnemu. Jako teoria programu, specyfikacja podlega rygorom matematycznym; w szczególności, związek poprawności programu względem specyfikacji podlega dowodowi. Jako teoria świata rzeczywistego, specyfikacja jest jego abstrakcją. Jako teoria programu, specyfikacja jest przepisem charakteryzującym jego zasadnicze cechy. Wynika stąd dualny, deskryptywny i preskryptywny (normatywny) charakter specyfikacji. (W pracy zbiorowej [2] można znaleźć wiele przykładów ilustrujących bądź jeden, bądź drugi aspekt specyfikacji; łączne traktowanie obydwu pochodzi z pracy [7], por. także łatwiej dostępny raport [9]).

Jak dotąd, związek specyfikacji ze światem rzeczywistym nie zyskał powszechnego uznania jako pełnoprawny przedmiot zainteresowania informatyki. Mówi się często, że praca informatyka zaczyna się **po ustaleniu formalnej specyfikacji zadania.** Doświadczenia poważnych firm żyjących w wytwórstwa oprogramowania użytkowego, zwłaszcza zaś oprogramowania dla jednostkowych klientów, wskazują jednak na fundamentalne znaczenie jakości tego związku. Truizmem jest bowiem stwierdzenie, że koszt korygowania specyfikacji funkcjonującego oprogramowania wyraźnie przekracza koszty poprawiania wszystkich innych błędów w nim dostrzeżonych. Jeśli więc ostatecznym celem przedsięwzięcia programistycznego jest zaspokojenie potrzeb klienta, co można osiągnąć tylko dzięki właściwemu funkcjonowaniu systemu informatycznego w realnym środowisku, to efektywność pracy informatyka zależy od umiejętności sprawdzenia (i ewentualnego skorygowania) specyfikacji **zanim zostanie zbudowane poprawne względem niej oprogramowanie.**

Czysto formalne (kalkulacyjne) metody sprawdzania specyfikacji, aczkolwiek wielce przydatne np. przy badaniu jej niesprzeczności [4] – są z natury rzeczy bezsilne wobec pytania o jej słuszność względem realiów, w których będzie ostatecznie funkcjonować zrealizowany system. W ostatnich latach rozwijają się dwa kierunki badawcze, mające na celu utechnologicznienie procesu sprawdzania specyfikacji; określa się je hasłami:
1) wykonywalnych specyfikacji (ang. executable specification),
2) szybkiego makietowania (ang. rapid prototyping).

Ponieważ ta (skądinąd pasjonująca) tematyka nie mieści się w ramach zakreślonych dla niniejszego artykułu, ograniczymy się do szkicowego streszczenia głównych idei tych dwu kierunków, nie zawsze zresztą ściśle rozgraniczanych [1, 3].

[1)] Podział artykułu na części i śródtytuły pochodzą od redakcji (przyp. red.).

Idea wykonywalnych specyfikacji polega na tym, że pisze się je w specjalnym języku, dla którego istnieje algorytm interpretacyjny, czyli dla którego można skonstruować program generujący następstwa (w obowiązującym dla danego języka systemie dedukcyjnym) z napisanej specyfikacji. Mając do dyspozycji taki program, możemy postępować na dwa sposoby: generować ciąg następstw (twierdzeń wypływających ze specyfikacji) i sprawdzać, czy zachodzą one w rzeczywistości, albo formułować twierdzenia opisujące interesujące nas zjawisko zachodzące w rzeczywistości i próbować udowodnić te twierdzenia w teorii wyznaczonej przez specyfikację. Zależnie od wybranego sposobu postępowania powinniśmy nieco zmieniać algorytm interpretacyjny; w jednym przypadku mamy bowiem do czynienia z **generatorem**, a w drugim – z **weryfikatorem** twierdzeń. I w jednym, i w drugim przypadku czysto kalkulacyjne algorytmy są zapewne zbyt mało sprawne (nieskończoność zbioru następstw i jego nieciekawa morfologia stanowią główne przeszkody czysto mechanicznego postępowania); prawie na pewno nie uda się tu uniknąć stosowania reguł heurystycznych. Ponadto, bezpośrednia postać konsekwencji specyfikacji może być mało czytelna dla klienta, co zmusza do ich przekładu na inny język, np. graficzny, bardziej przystępny dla użytkownika, dla którego projektujemy system i który **wie** czy dane zjawisko występuje, czy nie, w interesującej go dziedzinie.

Idea szybkiego makietowania polega na tym, żeby ze specyfikacji możliwie szybko wyprowadzić poprawny, choć być może niezbyt sprawny program. Eksperymentalna eksploatacja tego programu pozwala sprawdzić słuszność specyfikacji. Aby w pełni docenić sens takiego postępowania, należy zauważyć, że różnica kosztów między makietą (tj. naprędce skonstruowanym programem poprawnym) a ostateczną, handlową wersją programu może być bardzo pokaźna. Tworząc makietę można posługiwać się różnymi protezami, modułami wziętymi z bibliotek gotowych, uniwersalnych (a przeto nieoptymalnych) rozwiązań. Przeznaczając makietę do próbnej eksploatacji można osłabić wymagania odnośnie do dokumentacji użytkowej i konstrukcyjnej, a przede wszystkim – nie liczyć się z koniecznością jakiejkolwiek modyfikacji, gdyż makietę, która pomyślnie przeszła próby sprawdzające specyfikację i tak zaraz się wyrzuca, a w przypadku wykrycia błędów (specyfikacji!) – tworzy się następną.

SPECYFIKACJE FORMALNE

Niezależnie od tego, czy i jakimi metodami sprawdzania słuszności specyfikacji będziemy się posługiwać, przyjmujemy, że:

1) specyfikacja jest teorią żądanego rozwiązania informatycznego,

2) sprawdzenie słuszności specyfikacji jest zabiegiem, którego wykonanie dopuszcza inne niż czysto formalne (kalkulacyjne) metody postępowania i kryteria oceny.

Ponadto, zgodnie z ustalonym już poglądem nowoczesnej inżynierii oprogramowania, przyjmujemy, że jedynym kryterium oceny **poprawności** programu względem specyfikacji jest matematycznie ścisły dowód spełnienia ustalonego związku formalnego. Nie oznacza to oczywiście, że nalegamy na aposteroryczne dowodzenie poprawności oprogramowania; przeciwnie, postępowanie takie uważamy za ostatnią deskę ratunku. Prawidłowy przebieg procesu budowy oprogramowania powinien być taki, aby poprawność oprogramowania można było zagwarantować na mocy konstrukcji.

Niech **S** będzie specyfikacją programu. Aby **S** można było zaprezentować, musimy dysponować pewnym językiem **J0** i logiką **L0**.

Reguły **J0** określają składnię zdań, którymi wyrażamy specyfikację. Pomijamy tu całą masę szczegółów technicznych, dotyczących składniowego wyróżnienia poszczególnych kategorii językowych, alfabetu, nazewnictwa zmiennych, funktorów, sygnatury, a także ewentualnej wielotypowości języka. W każdym konkretnym przypadku języka użytego do pisania specyfikacji, szczegóły tego rodzaju muszą być oczywiście ustalone, ale nie wydaje się rzeczą rozsądną preferowanie jakiegoś jednego rozwiązania, tak samo jak nie jest rzeczą rozsądną forsowanie jakiegoś jednego języka programowania.

Reguły **L0** składają się z dwu części: **reguł wnioskowania** i **aksjomatów logicznych**. Reguły wnioskowania mają postać:

jeśli zdania **Z0, Z1, ..., Zn** są twierdzeniami, to zdanie **Z** też jest twierdzeniem

Dobrze znanym przykładem reguły wnioskowania jest tzw. **reguła odrywania** (modus ponens):

jeśli **A** i **A⇒B** są twierdzeniami, to **B** też jest twierdzeniem.

Reguły wnioskowania zapisuje się często w postaci ułamka, którego licznik stanowią przesłanki, a mianownik – wniosek, np.:

$$\frac{A,\ A \Rightarrow B}{B}$$

Aksjomaty logiczne są zbiorem zdań, których prawdziwość postulujemy dla wszystkich dziedzin stosowania wybranej logiki. Inaczej powiedziawszy, wybór aksjomatów logicznych ogranicza (na ogół bardzo liberalnie!) zbiór dziedzin, w których możemy bezpiecznie posługiwać się **L0**. Typowym przykładem aksjomatu logicznego jest **zasada wyłączonego środka** (tertium non datur), głosząca:

z dwu zdań **Z** i **nonZ** dokładnie jedno jest prawdziwe

Aksjomaty logiczne i reguły wnioskowania wybiera się tak, by posługując się językiem **J0** można było opisywać problemy powstające w interesujących nas dziedzinach. Wybór **L0** charakteryzuje nasz pogląd na to, jakiego rodzaju wnioskowania są typowe (przydatne) w pewnej klasie problemów.

Ze szkoły średniej pamiętamy logikę arystotelesowską, nie jest ona jednak jedyną logiką używaną we współczesnej informatyce. Tytułem przykładu wymienimy kilka innych logik, jakimi posługują się informatycy: logika klauzul Horna (podstawa Prologu) jest dość istotnym ograniczeniem arystotelesowskiej; różne logiki modalne, zwłaszcza zaś tzw. logiki temporalne, służą do opisu zjawisk zachodzących w obliczeniach współbieżnych; logiki niemonotoniczne są stosowane w komputerowych modelach wypowiedzi w języku etnicznym.

Opis konkretnego problemu – specyfikację – wyrażamy formułując **aksjomaty specyficzne** (pozalogiczne). Podobnie do aksjomatów logicznych, są one zdaniami, których prawdziwość jest postulowana; w przeciwieństwie do aksjomatów logicznych, aksjomaty specyficzne wyodrębniają stosunkowo wąską klasę dziedzin, w których są spełnione. Weźmy przykład:

dla dowolnych dwu różnych elementów, **x** i **y**, istnieje element **z**, taki że **x**⟨**z**⟨**y**

Jak łatwo widzieć, aksjomat ten jest spełniony w dziedzinie liczb rzeczywistych, ale nie w dziedzinie liczb całkowitych (przyjmując naturalną interpretację symbolu ⟨).

Zgodnie z naszym poglądem na naturę specyfikacji, przyjmujemy, że jest ona teorią, czyli zbiorem wszystkich następstw aksjomatów pozalogicznych, tj. wszystkich syntaktycznie poprawnych zdań języka **J0**, które w logice **L0** można wyprowadzić z tych aksjomatów. Zbiór następstw jest zazwyczaj niekończony, dlatego też skupiamy naszą uwagę na samych aksjomatach pozalogicznych, charakteryzujących teorię w znacznie zwięźlejszy sposób. Zauważmy, że pytania o niesprzeczność i zupełność tak pojmowanej specyfikacji nabierają dobrze określonego znaczenia.

Specyfikacja jest niesprzeczna, jeśli w języku **J0** nie ma takiego zdania **Z**, że zarówno **Z**, jak i **nonZ** jest następstwem ustalonych aksjomatów pozalogicznych. Sprzeczne specyfikacje nie mają, naturalnie, żadnego sensu praktycznego. Specyfikacja jest zupełna, jeśli każde poprawne zdanie języka **J0**, **Z**, lub jego zaprzeczenie **nonZ** należy do teorii przyjętej za specyfikację. Przypomnijmy, że każda sprzeczna teoria jest w tym sensie zupełna; to elementarne spostrzeżenie powinno być przestrogą wobec przywiązywania nadmiernej wagi do zupełności specyfikacji. Nieco dalej wskażemy inne, bardzo praktyczne obiekcje wobec dążenia do formułowania zupełnych specyfikacji.

ZADANIE PROGRAMISTYCZNE

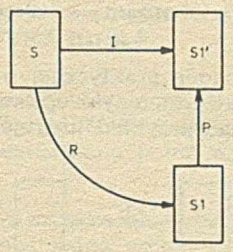
Zadanie programisty (zespołu programistów) polega na zrealizowaniu specyfikacji **S** w języku **J1** z logiką **L1**. W przyjętej konwencji oznacza to, że poszukujemy takiego poprawnie skonstruowanego **rozszerzenia P** systemu **S1** = {**J1,L1**} na system **S1'**, żeby istniała dobrze określona interpretacja
I: S→P
spełniająca następujący warunek (**W**):

jeśli **Z** jest następstwem **S** w systemie {**J0,L0**}
to **I(Z)** jest następstwem **P** w systemie {**J1,L1**}.

W powyższym sformułowaniu warunku poprawności **P** względem **S** uczyniliśmy kilka uproszczeń technicznych; pełne sformułowanie można znaleźć w książce [10], do której odsyłamy czytelników zainteresowanych szczegółami (stanowiącymi o matematycznej poprawności przyjętych konwencji).

Przez **następstwo** rozumiemy tutaj **wynikanie zgodne z regułami wnioskowania;** oznacza to, że jeśli logiki **L0** i **L1** różnią się nie tylko składniowo (np. pod względem użytych symboli zmiennych i funktorów), lecz również co do reguł wnioskowania, to relacje następstwa w obu rozważanych systemach mogą być nader różne. W codziennej praktyce programowania różnice między logikami **L0** i **L1** są zazwyczaj czysto techniczne, wynikają z konieczności uwzględnienia odmiennych symboli zmiennych i funktorów oraz z tego, że rozszerzenie **P** wprowadza dodatkowe symbole. Czytelnicy, którzy chcą sobie uzmysłowić praktyczny sens rozszerzenia **P**, mogą – bez obawy popełnienia grubej pomyłki – wyobrazić sobie **P** jako program w języku **J1**, czyli syntaktycznie poprawny zbiór deklaracji typów, zmiennych i procedur.

Schematycznie, zadanie programistyczne możemy przedstawić jak na rys. 1. Realizacja **R** specyfikacji **S** w **S1** = {**J1,L1**} jest wynikiem rozszerzenia **P** i interpretacji **I**.



Rys. 1. Schematyczne przedstawienie zadania programistycznego

Jest rzeczą oczywistą, że przy zadanej specyfikacji **S** i systemie lingwistycznym {**J1,L1**} możliwe jest wiele różnych rozszerzeń **P** realizujących **S** w **S1**.

Jeśli, na przykład, specyfikacja opisuje problem sortowania ciągów liczbowych, a **S1** jest poziomem lingwistycznym, powiedzmy Pascala, to zarówno program sortowania naiwnego, **P1**, jak i program sortowania dystrybucyjnego, **P2**, dopełniają realizacji **S** w **S1**. Programów takich, różniących się między sobą pod istotnymi względami (np. zajętości pamięci i czasu wykonywania) może być bardzo wiele. Wszystkie one są jednak w tej samej mierze podyktowane specyfikacją **S** i wyborem **S1**. Ponieważ system **S1'** powstający z rozszerzenia **S1** przez **P** można traktować jako model teorii **S**, dochodzimy do wniosku, że w jednej i tej samej dziedzinie **S1** teoria **S** może mieć wiele niezupełnie podobnych modeli, czego nie da się pogodzić z kategorycznością (silną formą zupełności) teorii **S**.

W tym, co powiedzieliśmy dotychczas, dominują elementy opisowe: używając trochę dziwnej terminologii przedstawiliśmy dobrze znaną sytuację. Czas przejść do konstruktywnego wykorzystania wprowadzonych pojęć i związków.

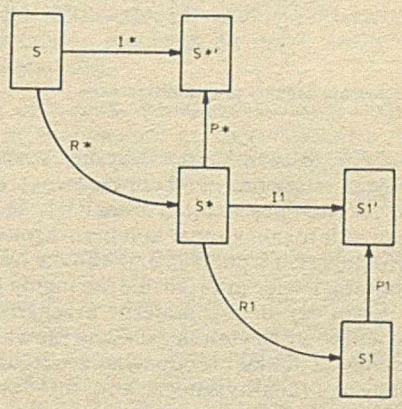
Gdyby przy zadanych **S** i **S1** można było od razu zgadnąć odpowiednie rozszerzenie **P**, zadanie programisty byłoby niezbyt trudne. Załóżmy, że tak jest w istocie, tj. że dysponując **S** i **S1**, w mniej lub bardziej magiczny sposób możemy otrzymać **P**. Jak można się przekonać, że **P** jest właściwe? Dosłowne sprawdzenie warunku (**w**) jest niemożliwe, ponieważ teoria **S** liczy nieskończenie wiele zdań. Na szczęście istnieją dwie metody postępowania, pozwalające opanować ten problem. Po pierwsze, możemy spróbować przeprowadzić tzw. meta-dowód, tj. dowód pewnej wypowiedzi o wszystkich twierdzeniach teoryj **S** i **S1'**; na ogół nie jest to jednak łatwe. Po drugie zaś, jeśli zadbamy, by interpretacja **I** spełniała pewne dość naturalne warunki techniczne (szczegóły opisano w książce [10]), to możemy posłużyć się ogólnym wynikiem, głoszącym:

jeśli aksjomaty pozalogiczne teorii S są w interpretacji I twierdzeniami teorii S1', to wszystkie zdania teorii s w interpretacji I też są twierdzeniami S1'.

Wystarczy więc przeprowadzić skończoną liczbę dowodów, by przekonać się o spełnieniu warunku (**w**), a tym samym o poprawności wskazanej realizacji specyfikacji **S** na poziomie językowym {**J1,L1**}.

Nie jest to jeszcze wszystko, co można uzyskać z przyjętego wzorca postępowania programistycznego, ale nie jest to zły początek: badanie poprawności realizacji sprowadziliśmy do dobrze określonego zadania o czysto kalkulacyjnym charakterze! (Podkreślmy całkowitą zgodność z ideologią wyłożoną na początku: badanie słuszności specyfikacji może wymagać postępowania przyrodniczego, badanie poprawności jej realizacji jest procesem czysto formalnym!)

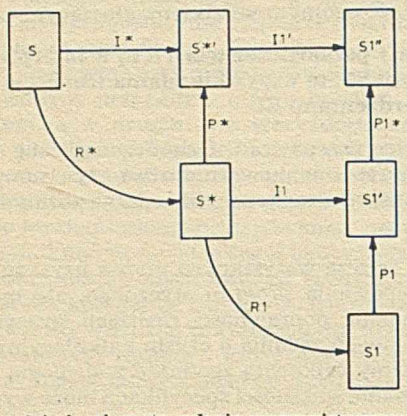
Jeśli odgadnięcie rozszerzenia **P** okazuje się zbyt trudne, to można zastosować postępowanie, które nazwiemy **interpolacją lingwistyczną.** Ustalamy taki poziom lingwistyczny {**J*,L***}, żeby:
1) specyfikację **S** można było łatwo zrealizować na poziomie {**J*,L***},
2) poziom {**J*,L***} był bliższy zadanego poziomu {**J1,L1**} niż poziom startowy {**J0,L0**}.
Jak znaleźć taki poziom {**J*,L***}? Na to pytanie nie ma ogólnej odpowiedzi! Wybór tego poziomu dyktuje pomysłowość i doświadczenie programisty. One też są jedynymi kryteriami „odległości" między poziomami lingwistycznymi. Konstruowanie oprogramowania niewątpliwie zawiera w sobie spory element twórczości ludzkiej, która nieprędko podda się formalizacji. (Czytelnikowi, kórego martwi albo zgoła szokuje to, że w tak formalnym podejściu do procesu programistycznego przyznaje się poczesne miejsce twórczości ludzkiej, należy się zapewnienie, że na wybór takiego stanowiska wpłynęła nieubłagana wymowa faktów. Ponadto, uznawszy doniosłą rolę twórczości, lepiej ograniczyć jej zakres do jednego, dobrze określonego miejsca w procesie budowy oprogramowania: do postulowania poziomów lingwistycznych. Reszta – a jest to ogromna część! – staje się ściśle kalkulacyjna. Zresztą, rachunek odgrywa sporą rolę i w samym postulowaniu poziomów lingwistycznych, np. przy badaniu właściwości syntaktycznych kreowanego poziomu.)



Rys. 2. Rozszerzenie schematu zadania programistycznego

Jeśli uda nam się pomyślnie zrealizować specyfikację **S** na poziomie {**J*,L***}, to możemy przejść do kolejnego kroku. Powstaje przy tym pytanie, co potraktować jako zadanie owego następnego kroku? Realizacje teorii **S*** rozszerzonej o program

P*, czy realizację samej **S***? Zanim odpowiemy na to pytanie, przyjrzyjmy się schematowi na rys. 2, którego dolny „trójkąt" jest dokładnym powtórzeniem górnego (z pominięciem różnic oznaczeń). Jeśli wariant realizacji specyfikacji **S** na poziomie {**J1**,**L1**} za pomocą tego schematu ma być w ogóle brany pod uwagę, to musimy dysponować stosunkowo prostą metodą wypełnienia prawego górnego rogu (rys. 3).

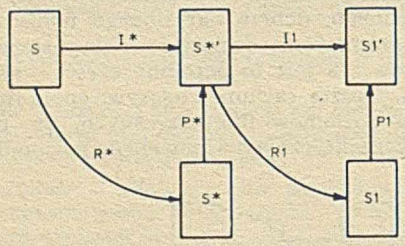


Rys. 3. Dalsze rozwinięcie schematu zadania programistycznego

Okazuje się (niestety, po szczegóły znowu musimy Czytelnika odesłać do [10]), że można sformułować warunki wystarczające na to, by rozszerzenie **P1*** było po prostu **I1(P*)**, interpretacja zaś **I1'** była określona jak następuje:

wyrażenia należące do języka **J*** interpretuje się w **I1'** tak samo jak w **I1**, a wyrażenia należące do rozszerzenia **P*** pozostawia bez zmiany. Realizację specyfikacji **S** w systemie **S1** wyznacza rozszerzenie $\mathbf{P} = \mathbf{P1} \times \mathbf{P1}'$ i interpretacja $\mathbf{I} = \mathbf{I}^* \times \mathbf{I1}'$, gdzie przez $\times$ oznaczono złożenie przekształceń. Z praktycznego punktu widzenia, po zrealizowaniu **S** w **S*** i **S*** w **S1**, realizację **S** w **S1** otrzymuje się czysto mechanicznie (algorytmicznie).

Zreferowany powyżej wynik można rozciągnąć na dowolną skończoną liczbę kroków: **S0**→**S1**→...→**Sn**, gdzie **S0** jest poziomem lingwistycznym początkowej specyfikacji, a **Sn** – poziomem ostatecznej implementacji programu, przy czym **wszystkie kroki mają identyczną strukturę!**

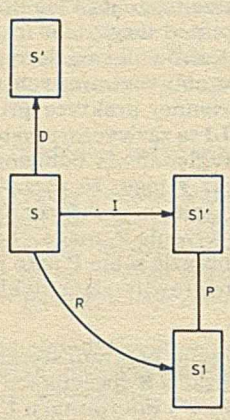


Rys. 4. Odmiana schematu zadania programistycznego

Do dość podobnego wyniku doszlibyśmy stosując schemat z rys. 4, pozornie prostszy nawet od poprzedniego, odpowiadający realizacji **S*'** (czyli **S*** poszerzonej przez **P***) w **S1**. Schemat ten wydaje się niezwykle atrakcyjny: w kolejnym kroku realizujemy nie język kroku poprzedniego, lecz uzyskany w owym poprzednim kroku program, a więc praktycznie nie mamy już nic więcej do zrobienia; nawet czysto mechaniczne przekształcenia nie są już potrzebne. Wnikliwy Czytelnik zapewne zauważył, że rozważany obecnie schemat dokładnie odpowiada metodzie programowania zastępującego (analitycznego, ang. top-down). Tego też schematu używano w pracy [8] i, w znacznym stopniu, w pracach [5] i [7]. Podobnie jak rozważany uprzednio, i ten schemat (realizacji pośredniego poziomu językowego rozszerzonego o program) można rozciągnąć na dowolną liczbę kroków. W stosunku do poprzedniego ma on jednak dwie poważne wady.

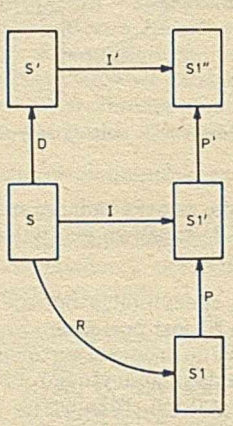
Po pierwsze, ponieważ dla każdego kroku musimy sprawdzić spełnienie warunku (**w**), im mniej jest aksjomatów, których interpretacje należy udowodnić na nowym poziomie językowym, tym lepiej. W drugim schemacie postępowania musimy udowodnić nie tylko prawdziwość interpretacji aksjomatów pozalogicznych języka, lecz także wszystkich aksjomatów dodanych przez rozszerzenie (program). Niekiedy jest to bardzo poważne zwiększenie zakresu pracy (np. gdy rozszerzenie ma charakter czysto definicyjny – wprowadza nowe nazwy czy symbole, całkowicie równoważne kombinacjom wyrażeń z języka podstawowego); jeśli jednak rozszerzenie wprowadza istotnie nowe jakości semantyczne (do czego prędzej czy później musi dojść w procesie konstruowania każdego niebanalnego programu!), to ilość pracy wyraźnie się zwiększa.

Po drugie, postępowanie zgodne z pierwszym schematem jest ogólniejsze: realizując język (albo cały poziom lingwistyczny, jeśli w danym kroku rzeczywiście zmieniamy logikę), zachowujemy swobodę zmian programu napisanego w tym języku bez konieczności powtarzania najtrudniejszej części następnego kroku. Po zmianie programu należy tylko obliczyć jego nową interpretację, a to jest czynnością mechaniczną.



Rys. 5. Ilustracja rozszerzenia specyfikacji przez klienta

Dwa te powody, nader praktycznej natury, zdecydowały o przyjęciu za podstawę metodyki opisanej w [10] schematu pierwszego. Następujący przykład ilustruje zalety tego wyboru.



Rys. 6. Zrealizowanie poszerzonej specyfikacji S' na poziomie językowym S1

Przypuśćmy, że rozwiązaliśmy zadanie programistyczne, tj. za pomocą programu **P** zrealizowaliśmy specyfikację **S** na poziomie językowym **S1** = {**J1**,**L1**}. Jak to w życiu zwykle bywa, nasz klient zdecydował się jednak rozszerzyć początkowe zadanie, czego wyrazem jest poszerzenie specyfikacji **S** o dodatkowe wymagania **D**. Schematycznie, możemy przedstawić to tak, jak na rys. 5. Nowe zadanie polega na zrealizowaniu poszerzonej specyfikacji **S'** na poziomie językowym **S1**. Bez trudu można jednak zauważyć, że jest to zadanie identycznej postaci co rozważane uprzednio zadanie „wypełnienia prawego górnego rogu"! Chodzi o znalezienie takiego rozszerzenia **P'** i takiej interpretacji **I'**, aby **S"** było pożądaną realizacją (rys. 6). Jeśli **D** spełnia pewne dość

dokończenie na s. 18

FRANK FOERSTERLING
Instytut Informatyki i Techniki Obliczeniowej
Akademia Nauk NRD
Berlin

Wcześniejszą wersję tego artykułu przedstawiono na X Międzynarodowym Seminarium nt. Zarządzania Bazami Danych, 19–24 października 1987, Cedzyna koło Kielc.

Wymagania stawiane bazom danych w biurach (1)

Praca biurowa jest zajęciem wykonywanym głównie przez ludzi, a nie automaty, gdyż wymaga rozwiązywania problemów, obsługi sytuacji nietypowych, przeprowadzania pertraktacji i wielu innych form aktywności, które z trudem poddają się analizie ilościowej. Liczba osób zatrudnionych w biurach stale rośnie, lecz wyposażenie techniczne biura nie w pełni odpowiada obecnym potrzebom. Wprowadzenie takich nowych środków technicznych, jak systemy informacyjne powiązane z komputerowymi systemami komunikacji, stwarza możliwości rozwoju nowego rodzaju systemów przeznaczonych dla biur i usprawniających ich funkcjonowanie – tak zwanych biurowych systemów informacyjnych (BSI).

Pierwsza część artykułu zawiera sformułowanie podstawowych definicji i charakterystyk biura, przegląd niektórych modeli biura według ich klasyfikacji oraz omówienie ogólnej architektury biurowych systemów informacyjnych.

CHARAKTERYSTYKA BIURA

W dowolnej organizacji biuro jest tą jej częścią, która obsługuje przepływ informacji w celu wykonania jednego lub wielu **zadań.** Praca biura jest realizowana przez wykonywanie czynności biurowych.

Czynność biurowa oznacza jakąkolwiek działalność w biurze, którą może wykonać jedna osoba. Do typowych czynności biurowych można zaliczyć np. redagowanie tekstu, porządkowanie archiwum, wysłanie listu, podjęcie decyzji.

Procedura biurowa jest to plan wykonania ustrukturowanego zestawu czynności biurowych. Ze względu na ścisłe odseparowanie od siebie obszarów działania i odpowiedzialności pracowników, procedura biurowa stanowi normalny i pożądany sposób wypełnienia zadania stawianego przed biurem. Przykładem procedury biurowej jest realizacja zlecenia kupna.

Pojęcie **modelu biura** odnosi się do abstrakcyjnego opisu semantyki funkcjonowania złożonego biura.

Biurowy system informacyjny jest to model biura zrealizowany w systemie komputerowym.

Biuro ma pewne cechy charakterystyczne, właściwe tylko jemu lub bardziej znaczące niż w odmiennych systemach.

FRANK FÖRSTERLING urodził się w 1958 r. w Berlinie. W 1982 roku ukończył studia matematyczne na uniwersytecie w Charkowie (ZSRR). Jest pracownikiem naukowym w Instytucie Informatyki i Techniki Obliczeniowej Akademii Nauk NRD. Uczestniczy w pracach nad Komputerowym Systemem Komunikatów (Computer-based Message System) dla środowiska rozproszonego. Interesuje się głównie systemami zarządzania bazami danych dla zastosowań niestandardowych (szczególnie dla biur) oraz rozproszonymi biurowymi systemami informacyjnymi.

Biuro jest systemem zdominowanym przez **ludzi.** Niezależnie od wszystkiego, zasadnicza większość prac biurowych będzie wykonywana przez pracowników. Człowiek może jednak popełniać błędy, reagować w niespodziewany sposób lub zaniedbywać swoje obowiązki. W biurowym systemie informacyjnym powinien występować jakiś mechanizm zaradczy, przeciwdziałający takim i podobnym nieprawidłowościom. Innym ważnym wymaganiem wynikającym z dominującej pozycji ludzi w biurze jest zaakceptowanie BSI przez pracowników. Niezbędny jest więc przyjazny użytkownikowi sposób komunikowania się z BSI.

Biuro jest systemem **dynamicznym.** W większości dziedzin działalności biura można spodziewać się częstych zmian. Nie za każdym razem udaje się zakończyć dana czynność biurową przed pojawieniem się nowej czynności.

Biuro jest **systemem współbieżnym.** Praca osób zatrudnionych w biurze jest w dużym stopniu równoległa. Czynności należące do tej samej procedury lub związane z tymi samymi przechowywanymi informacjami wymagają synchronizacji.

Biuro jest **systemem otwartym.** Częste zmiany obszarów działalności biura uniemożliwiają dokonanie jego kompletnego opisu.

Biuro **podlega ewolucji.** Czynności mogą być zmieniane, zmienia się też struktura informacji (np. struktura dokumentów). „Najbardziej niestabilne aspekty biura, to ograniczenia nakładane na jego obiekty składowe: dla przykładu – upoważnienie pracownika do wykonania pewnej operacji" [2].

Biuro jest **systemem złożonym.** Musi uwzględniać wiele informacji różnych typów. Zazwyczaj istnieje wielka liczba obiektów informacyjnych, które pozostają w relacji określonej przez wiele (np. kilkanaście) powiązań. Struktura tych obiektów jest na ogół bardzo złożona, np. hierarchiczna lub sieciowa. Manipulowanie takimi złożonymi obiektami i ich skoordynowanie wymaga bardzo mocnych narzędzi, zdolnych do pełnego wyrażenia semantyki.

Biuro jest **systemem wielonośnikowym** (ang. multimedia). Informacje są dostarczane za pośrednictwem wielu środków przekazu: listów, dokumentów, obrazów, głosu (przez telefon) itd. Obsługa informacji wpływających wymaga zastosowania nowych typów danych.

Biuro jest **systemem uzależnionym czasowo,** przy czym uzależnienie to nie zawsze jest jasno określone. Często jest łatwo powiedzieć, jak dana czynność ma być wykonana, lecz jednocześnie nie jest możliwe określenie, kiedy ma być ona wykonana. W związku z tym uzależnieniem czasowym ważną rolę w pracy biura, np. w procesie iteracyjnego tworzenia projektu dokumentu przeznaczonego na użytek zebrania, odgrywa pojęcie wersji (wariantu).

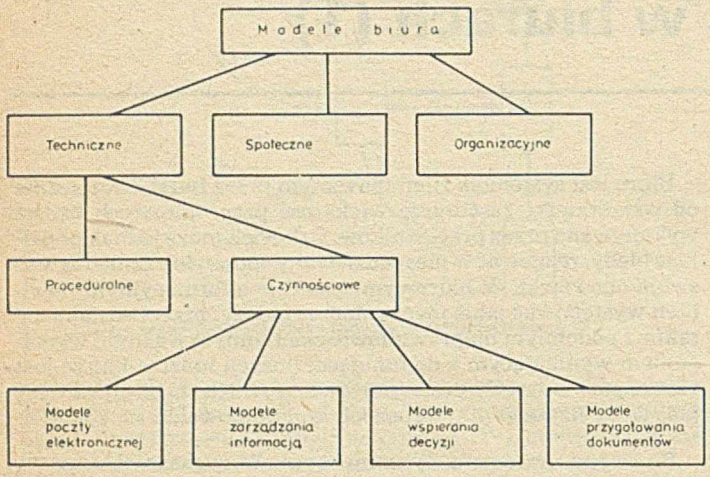
Biuro jest **systemem elastycznym.** Te same obiekty informacyjne muszą być prezentowane różnym klientom w różnych postaciach, np. w postaci tabelarycznej albo graficznej.

Biuro jest **systemem z niepełną informacją.** Normalna jest sytuacja, w której dostarczana informacja ma luki. BSI musi być

zdolny do tolerowania takiej niepełnej informacji i późniejszego jej uzupełniania.

KLASYFIKACJA MODELI BIURA

Ze względu na szybką ewolucję technoloii BSI i wyjątkowo wielką liczbę urządzeń, które mogą być stosowane w biurze, większość modeli, na których są opierane projekty i specyfikacje BSI, tworzy się na ile to możliwe niezależnie, na poziomie konceptualnym.



Rys. 1. Klasyfikacja modeli danych

Obecnie w literaturze można znaleźć wiele modeli biura o różnych zakresach zastosowań (rys. 1). Nie ma jednak wśród nich takiego modelu, który by w pełni pokrywał wszystkie aspekty działalności rzeczywistego biura.

Modele biura

Pierwszy poziom drzewa klasyfikacji reprezentuje różne sposoby widzenia biura. Występują następujące klasy modeli:

- klasa modeli technicznych,
- klasa modeli społecznych,
- klasa modeli organizacyjnych.

Celem **modeli społecznych** jest badanie roli istot ludzkich w funkcjonowaniu biura. Do tej klasy zaliczają się modele behawioralne. Celem **modeli organizacyjnych** jest logiczny podział organizacji na podgrupy (np. działy). Obydwie te klasy modeli pozostają poza zakresem dalszych rozważań. Na polu badawczym, wyznaczonym przez te klasy, koncentrują się głównie nauki społeczne.

Klasa **modeli technicznych** jest ukierunkowana na szczegółową analizę pracy biura. Głównym zagadnieniem w tej klasie jest techniczne wsparcie pracy biura. Ta właśnie klasa jest najbardziej związana z dziedziną informatyki.

Klasa modeli technicznych

Zależnie od tego, na który z dwóch głównych elementów biura kładzie nacisk dany model, rozróżnia się następujące podklasy:

- modele proceduralne,
- modele czynnościowe.

Celem **modeli proceduralnych** jest skoordynowana reprezentacja czynności biurowych, podkreślająca ich współbieżny charakter, z uwzględnieniem obsługi systuacji wyjątkowych; podstawowym elementem modelu proceduralnego jest czynność. „Nacisk kładzie się na stworzenie zintegrowanego obrazu wszystkich czynności wykonywanych w biurze dla realizacji określonych zadań, a nie na podanie opisu każdej czynności oddzielnie" [1]. Modele proceduralne opisują przepływ informacji i sterowania w biurze; podstaw matematycznych dostarcza teoria sieci Petriego. Do modeli proceduralnych należą m. in.:

- DOMINO (GMD),
- sieci nadzoru informacji (Xerox),
- model procedura biurowej (Illinois Institute of Technology).

Celem **modeli czynnościowych** jest stworzenie modelu stanowiska pracy. Do klasy tej należy kilkanaście podejść do modelowania. W celu lepszego zrozumienia różnorodnych zajęć, jakie musi podejmować pracownik biura, wprowadza się dalszy podział modeli:

- poczty elektronicznej,
- zarządzania informacją,
- wspierania decyzji,
- przygotowania dokumentów.

W modelach poczty elektronicznej bierze się pod uwagę głównie przesyłanie danych między różnymi osobami. Podstawą do rozważań są sieci komputerowe. Klasa ta obejmuje takie rozwiązania, jak:

- przesyłanie tekstu,
- przesyłanie głosu,
- przesyłanie wielonośnikowe,
- systemy obsługi komunikatów,
- konferencje komputerowe.

Celem zarządzania informacją jest wspomaganie pracownika biura przez udostępnienie środków umożliwiających mu patrzenie na ekran monitora jak na blat własnego biurka. Klasa ta obejmuje m. in. systemy:

- zarządzania bazami danych,
- operowania formularzami,
- wyszukiwania informacji.

Modele wspierania decyzji mają na celu wspomaganie pracy umysłowej w biurze. W klasie tej rozważa się takie rozwiązania, jak:

- systemy ekspertowe,
- arkusze kalkulacyjne (ang. spreadsheets),
- systemy grafiki prezentacyjnej.

W modelach przygotowania dokumentów kładzie się nacisk na tworzenie tekstu lub dokumentów. Istnieje wiele edytorów tekstu, różniących się możliwościami. Nową tendencją w tej grupie klasyfikacyjnej jest próba wprowadzenia standardów Budowy Dokumentów Biurowych (ang. Office Document Architecture) i Wymiennego Formatu Dokumentów Biurowych (ang. Office Document Interchange Formats).

Ewolucja modeli biura

Pierwsze modele biura należały do poszczególnych podklas klasy modeli czynnościowych. Pokrywały one niektóre obszary podklas modeli przygotowania dokumentów i zarządzania informacją, lecz bez żadnego związku ze stanowiskiem roboczym.

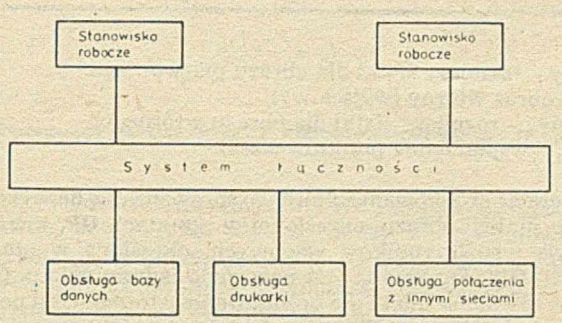
Nowe technologie łączności wywarły wpływ na utworzenie modeli oczty elektronicznej, a pojawienie się klasy modeli wspierania decyzji jest ściśle związane z rozwojem metod sztucznej inteligencji.

Praktyczne zastosowania teorii sieci Petriego objęły także tworzenie modeli biura z klasy modeli proceduralnych. Od początku lat osiemdziesiątych obserwuje się proces łączenia w jednym modelu kilkunastu funkcji biura. W związku z tym mówi się o zintegrowanych rozwiązaniach w tworzeniu modeli. Zazwyczaj rozwiązania te przedstawia się użytkownikowi w postaci „skrzynki z narzędziami" o zróżnicowanych właściwościach. Narzędzia są powiązane ze sobą przez jednolity sposób ich udostępnienia użytkownikowi. Podejście wiążące rozwiązania proceduralne z modelami czynnościowymi wydaje się najlepszą drogą tworzenia modelu funkcjonowania złożonego biura.

Tylko kilka modeli opisanych w literaturze zostało w pełni zaimplementowanych. Często przedstawia się jedynie koncepcję. Obecnie nie są dostępne na rynku systemy biurowe oparte na modelach łączących podejście proceduralne i czynnościowe. Biura zaś potrzebują takich właśnie narzędzi do zwiększenia produktywności.

ARCHITEKTURA BIUROWYCH SYSTEMÓW INFORMACYJNYCH

Implementację modelu biura w systemie komputerowym nazywa się biurowym systemem informacyjnym (BSI). Biurowy system informacyjny jest środowiskiem rozproszonym, w którym elementy wyposażenia technicznego są połączone przez sieć komputerową. Głównymi składnikami BSI są (rys. 2):
- indywidualne stanowiska robocze użytkowników, cechujące się przyjaznym sposobem współpracy z systemem,
- komputerowy system łączności,
- różne rodzaje systemów usługowych, w tym obowiązkowo system obsługi bazy danych.



Rys. 2. Architektura Biurowych Systemów Informacyjnych

Stanowiska robocze dostarczają skutecznych środków komunikacji z użytkownikiem, opartych na technice okien. Doświadczenia z relacyjnym językiem programowania QBE (ang. Query By Example) i biurowymi systemami informacyjnymi QBE i OPAS wskazują, że dwuwymiarowe i nieproceduralne języki programowania, przeznaczone do specyfikowania potrzeb użytkownika, potwierdzają swoją przydatność. Na użytek wielonośnikowego przekazywania informacji, stanowisko robocze musi być wyposażone w wielonośnikowy edytor. Typowe wymagania sprzętu na stanowisku roboczym to:
- komputer 16-bitowy z pamięcią operacyjną o pojemności co najmniej 512 KB (dla systemów wielonośnikowych do 2 MB).
- dysk stały o pojemności co najmnej 20 MB),
- monitor rastrowy do prezentacji graficznej,
- mysz, stosownie do wymagań użytkownika.

System łączności stanowi ośrodek wiążący wszystkie stanowiska robocze i podsystemy usługowe. Zazwyczaj jest to szybka lokalna sieć komputerowa (np. Ethernet).

Systemy usługowe zapewniają realizację usług niezbędnych do funkcjonowania biura. Jedną z głównych usług jest gromadzenie informacji. Każde biuro wytwarza i zarządza wieloma obiektami informacyjnymi. Nieefektywne byłoby przechowywanie ich wszystkich w stanowiskach roboczych. Składowa systemu, odpowiedzialna za gromadzenie informacji, musi odzwierciedlać typowe elementy pracy biurowej, jakimi są: dokumenty, teczki i kartoteki.

We współczesnych systemach projektanci i programiści zwiększyli nacisk na komunikację z użytkownikiem. Większość niedostatków dziesiejszych BSI wiąże się ze zbyt prostymi podsystemami obsługi gromadzenia informacji [3, 4]. Charakteryzując system Star, Gibbs pisze [4]: „Być może najpoważniejsze z punktu widzenia modelu jest to, że zdolność zarządzania danymi jest w istocie oparta[1)] na prostym systemie plików. Choć dane mogą być modyfikowane, udostępniane w odpowiedzi na zapytania, formatowane przy użyciu sprzęgu graficznego systemu Star, to jednak nie podjęto próby opisu logicznych wewnętrznych związków między tymi danymi. W konsekwencji ani specyfikacja związków i pytania ich dotyczące nie mogą być tak wyrafinowane, jak w systemach zarządzania bazami danych, ani też nie jest możliwa kontrola współdzielenia dostępu do danych przez wielu użytkowników jednocześnie".

Zastosowania biurowe tworzą zapotrzebowanie na nowe systemy zarządzania bazami danych, zrealizowane jako podsystemy usługowe. Wymagania strukturalne i operacyjne obiektów baz danych są kształtowane przez sposób komunikowania się (sprzęg) z użytkownikiem. Wymagania te, a także inne warunki stosowania systemu zarządzania bazą danych w BSI oraz architektura takiego systemu zostaną przedstawione w drugiej części artykułu.

[1)] W systemie Star (przyp. red.).

Tłumaczył i opracował:
A. RADOMSKI

LITERATURA

[1] Barbic F. et al.: Modeling and Integrating Procedures in Office Information Systems Design. Information Systems, Vol. 10, No. 2, pp.149 168, 1985
[2] Bracchi G., Pernici B.: The design requirements of office systems. ACM Trans. on Office Information Systems. Vol. 2, No. 2, pp. 151 170, April 1984
[3] Ellis C. A., Nutt G. J.: Office Information Systems and Computer Science. Computing Surveys, Vol. 12, No. 1, pp. 26 60, March 1980
[4] Gibbs S. J.: Conceptual Modeling and Office Information Systems. Tsichritzis D. (Ed.): Office Automation, Concepts and Tools. Springer-Verlag, Berlin, 1985.

GRZEGORZ KALETA
JACEK OWCZARCZYK
Instytut Podstaw Informatyki
Polska Akademia Nauk
Warszawa

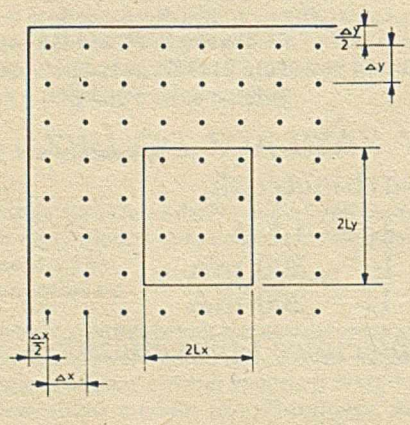
Wieloprocesorowa realizacja operacji przetwarzania obrazów

Celem prac prowadzonych w IPI PAN przez autorów tego artykułu (por. [6, 7]) jest stworzenie wieloprocesorowego systemu przetwarzania obrazów, którego architektura mogłaby być dynamicznie programowo zmieniana w zależności od struktury realizowanych obliczeń. Wymaga to jednak wcześniejszego przeanalizowania wszystkich operacji przetwarzania obrazów w celu określenia podstawowych typów architektury, w które system ten mógłby być rekonfigurowany.

Celem artykułu jest porównanie efektywności dwóch podstawowych struktur wieloprocesorowych, tj. struktury SIMD (ang. single instruction, multiple data) i struktury MIMD (ang. multiple instruction, multiple data), w realizacji operacji próbkowania wtórnego (ang. image resampling). W przedstawionych rozważaniach zakłada się, że struktury te są budowane z identycznych procesorów elementarnych, stanowiących systemy mikroprocesorowe z lokalną pamięcią o pojemności dostatecznej do prowadzenia obliczeń. Każdy z procesorów elementarnych ma cztery jednobitowe, dwukierunkowe magistrale umożliwiające międzyprocesorową transmisję danych, przy czym proces ten z założenia jest współbieżny z przetwarzaniem. Praktyczną implementacją modelu procesora elementarnego mógłby być na przykład układ VLSI typu Transputer [9].

ASPEKTY OBLICZENIOWE PRÓBKOWANIA WTÓRNEGO

Próbkowanie wtórne jest w istocie zagadnieniem interpolacji funkcji dwóch zmiennych w węzłach siatki prostokątnej, przy czym dane wejściowe stanowi zbiór wartości tej funkcji w węzłach innej siatki. Obydwie siatki różnią się zazwyczaj zarówno wzajemną orientacją, jak i wielkością podstawowego kroku. Konieczność wykonania takiej operacji pojawia się na przykład w wypadku obrotu obrazu cyfrowego w celu skorelowania go z innym obrazem. Jakkolwiek próbkowanie wtórne jest tu rozważane w kontekście przetwarzania obrazów, to zastosowania tej operacji są znacznie szersze, gdyż jest ona również używana w wielu dziedzinach, w których są przetwarzane dane przestrzenne.



Rys. 1. Reprezentacja graficzna próbkowania wtórnego

Przyjęto następujące oznaczenia:
OP – obraz pierwotny (wejściowy);
Δx, Δy – rozmiary siatki dla obrazu pierwotnego;
OW – obraz wtórny (wyjściowy);
Δx_n, Δy_n – rozmiary siatki dla obrazu wtórnego;
x_n, y_n – współrzędne punktów siatki.

Operacja próbkowania wtórnego sprowadza się do wykonania splotu funkcji obrazu, określonej w punktach **OP**, z wybraną funkcją współczynników wagowych, określoną w otoczeniu punktu **OW**. Funkcja ta jest nazywana jądrem splotu. Ogólna postać równania splotu dla próbkowania wtórnego jest następująca (rys. 1):

$$I_r(x,y) = \sum_{j_x = l_x}^{u_x} \sum_{j_y = l_y}^{u_y} I(j_x\Delta x + \Delta x/2;\ \ j_y\Delta y + \Delta y/2) \times$$
$$\times R(x_n - (j_x\Delta x + \Delta x/2);\ \ y_n - (j_y\Delta y + \Delta y/2)) \qquad (1)$$

gdzie: $\mathbf{I(x, y)}$ jest funkcją **OP**; $\mathbf{I_r(x, y)}$ jest funkcją **OW** po próbkowaniu; $\mathbf{R}(\alpha, \beta)$ jest wartością jądra splotu w (α, β). Jądro splotu jest określone w obszarze interpolacji tak, że:

$$R(\alpha,\beta)\begin{cases} \neq 0, \text{ jeśli } |\alpha| \leqslant L_x, |\beta| \leqslant L_y \\ = 0 \text{ w przeciwnym razie} \end{cases}$$

Sumowanie ważonych wartości funkcji **OP** odbywa się dla danego punktu **OW** tylko w tym obszarze, czyli:

$$|x_n - (j_x\Delta x + \Delta x/2)| \leqslant L_x;\ \ |y_n - (j_y\Delta y + \Delta y/2)| \leqslant L_y \qquad (2)$$

Na podstawie wyrażenia (2) można obliczyć granice sumowania w równaniu (1):

$$u_x = \left\lceil \frac{x_n + L_x - \Delta x/2}{x} \right\rceil; \qquad l_x = \left\lceil \frac{x_n - L_x - \Delta x/2}{x} \right\rceil$$
$$u_y = \left\lfloor \frac{y_n + L_y - \Delta y/2}{y} \right\rfloor; \qquad l_y = \left\lfloor \frac{y_n - L_y - \Delta y/2}{y} \right\rfloor \qquad (3)$$

Zapis górny oznacza najmniejszą liczbę całkowitą większą niż **A**, natomiast zapis dolny – największą liczbę całkowitą mniejszą niż **A**. Indeksy sumowania $\mathbf{j_x}$ i $\mathbf{j_y}$ w równaniu (1) mogą być traktowane jako indeksy punktów **OP**. Stąd współrzędne punktów w układzie związanym z **OP**:

$$x = j_x\Delta x + \Delta x/2; \qquad y = j_y\Delta y + \Delta y/2$$

Aby wyrazić współrzędne x_n i y_n punktów **OW** w układzie związanym z **OP** wprowadzono przesunięcie globalne O_x i O_y **OW** względem **OP** (rys. 2). Wówczas:

$$x_n = i_x\Delta x_n + O_x; \qquad y_n = i_y\Delta y_n + O_y \qquad (4)$$

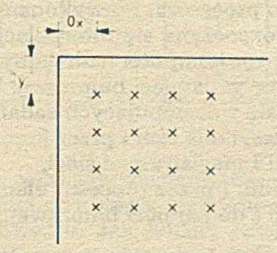
gdzie: $\mathbf{i_x}$, $\mathbf{i_y}$ są indeksami punktów **OW**.

Po podstawieniu zależności (4) do równania (1) otrzymuje się:

$$I_r(x,y) = \sum_{j_x = l_x}^{u_x} \sum_{j_y = l_y}^{u_y} I(j_x\Delta x + \Delta x/2;\ \ j_y\Delta y + \Delta y/2) \times$$

$\times R(i_x\Delta x_n + O_x - (j_x\Delta x + \Delta x/2);\ i_y\Delta y_n + O_y - (j_y\Delta y + \Delta y/2)$ (5)

W próbkowaniu wtórnym istotna jest relacja współrzędnych punktów i próbkowania względem danych wejściowych. Relacja ta jest stała jedynie w banalnych wypadkach, np. gdy $x=1$, $x_n=1/2$, $y=1$, $y_n=1/2$ (podwójne zagęszczenie siatki).



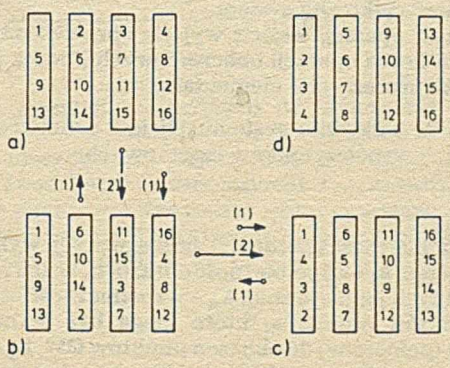
Rys. 2. Przesunięcie globalne siatki OW względem układu współrzędnych OP

Z reguły jednak relacja punktów **OW** i punktów **OP** zależy od współrzędnych x_x, y_n, np. gdy Δx_n lub Δy_n nie są całkowitymi dzielnikami Δx, Δy lub dla obrotu. W rezultacie jądro splotu dla kolejnych punktów **OW** może przyjmować różne wartości i musi być wyznaczone przed obliczeniem wartości danego punktu. Jest to bardzo ważna właściwość próbkowania wtórnego, która odróżnia je (pod względem obliczeniowym) np. od operacji wygładzania lub detekcji brzegów, gdzie jądro splotu jest takie samo dla wszystkich punktów obrazu, niezależnie od położenia. Dlatego też próbkowanie wtórne nie może być traktowane jak prosta operacja lokalna [6]. Inaczej mówiąc, współczynniki wagowe, przez które są mnożone wartości punktów **OP** z **n**-elementowego jądra splotu dla punktu **OW**, mogą być różne dla dwóch kolejnych punktów **OW** i wymagają wcześniejszego obliczenia.

PRÓBKOWANIE WTÓRNE W SYSTEMIE SIMD

Wieloprocesorowe systemy przetwarzania obrazów o strukturze SIMD bardzo efektywnie realizują operacje lokalne. Jak wynika z wcześniejszych rozważań, struktura obliczeń w próbkowaniu wtórnym różni się od tej klasy operacji, jednak w dalszej części artykułu przedstawiono możliwość realizacji tej operacji za pomocą systemu o strukturze SIMD.

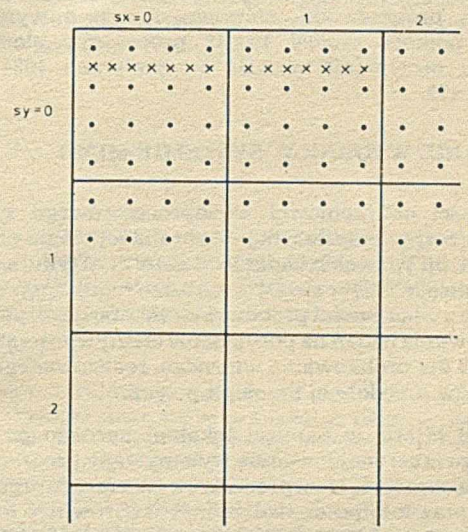
Zazwyczaj obraz wejściowy jest ładowany do takiego systemu za pomocą rejestru szeregowo-równoległego do skrajnego wiersza procesorów elementarnych, z których wypełnienie tablicy odbywa się przez cykliczną propagację danych do kolejnych wierszy. Ponieważ obrazy cyfrowe są przesyłane (pozyskiwane) z reguły kolejnymi liniami, więc użycie do ładowania obrazu rejestru szeregowo-równoległego spowoduje, że w pamięciach kolejnych procesów elementarnych punkty obrazu zostaną przemieszane (rys. 3a). Do wykonania operacji próbkowania wtórnego obraz pierwotny o wymiarach $M_x \times M_y$ powinien być uporządkowany w taki sposób, aby w pamięci każdego procesora elementarnego znalazł się fragment obrazu o wymiarach



Rys. 3. Ortogonalizacja obrazu

$m_x \times m_y$. Aby uzyskać takie uporządkowanie elementów obrazu trzeba wykonać tzw. ortogonalizację obrazu [3]. Jej zasada polega na przesyłaniu (a właściwie „obracaniu") danych wzdłuż wierszy i kolumn tablicy procesorów. Na rysunkach 3b i 3c przedstawiono kroki pośrednie przy ładowaniu i ortogonalizacji szesnastoelementowej linii w tablicy złożonej z czterech procesorów. Wykonując próbkowanie wtórne w systemie SIMD, do którego obraz pierwotny został załadowany w powyższy sposób, przyjmuje się, że każdy procesor elementarny przetwarza jeden przydzielony mu podobraz. Następstwa takiego podejścia są następujące.

Po pierwsze, regularny podział **OP** na obszary wcale nie pociąga za sobą regularnego podziału **OW**. Na przykład, mimo że obszary o indeksach $s_x = s_y = 0$ i $s_x = 1$, $s_y = 0$ (rys. 4) mają tę samą wielkość ($m_x = m_y = 4$), to w **i**-tej linii obszaru $s_x = s_y = 0$ znajduje się 7 punktów **OW**, a w tej samej linii obszaru $s_x = 1$, $s_y = 0$ jest 8 punktów **OW**. Oznacza to, że różne procesory elementarne będą wykonywać różną liczbę obliczeń. Fakt ten musi być uwzględniony przy projektowaniu algorytmów dla struktury SIMD: jednym z ich elementów będzie wtedy maskowanie niektórych procesów elementarnych, aby pozostałe mogły dokonać obliczeń dla swoich dodatkowych punktów.



Rys. 4. Różna liczba punktów OW w różnych obszarach

Po drugie, relacja punktów **OW** względem **OP** jest różna w różnych obszarach. W rezultacie dla każdego z procesorów elementarnych jest potrzeba inna wartość jądra splotu.

Po trzecie, w każdym obszarze istnieje pas brzegowych punktów **OW**, do których obliczenia są wymagane wartości punktów **OP** spoza obszaru załadowanego do pamięci danego procesora elementarnego. Granice sumowania z udziałem takich punktów są określone zależnościami:

$$l_x < 0 : \lfloor (L_x/\Delta x + 1/2) \rfloor \leqslant l_x \leqslant \lfloor (L_x - \Delta x_n)/\Delta x + 1/2 \rfloor$$
$$u_x \geqslant m_x : \lfloor (L_x/\Delta x + 1/2) \rfloor \leqslant u_x - m_x - 1 \leqslant \lfloor (L_x - \Delta x_n)/\Delta x + 1/2 \rfloor \quad (6)$$

Największym utrudnieniem w realizacji próbkowania wtórnego za pomocą systemu SIMD jest drugi z wymienionych aspektów. Konieczność obliczenia różnych jąder splotu dla różnych procesorów elementarnych można jednak wyeliminować stosując podaną wcześniej zależność związaną z przesunięciem **OW** względem **OP**. Wówczas przesunięcie globalne należy zapisać przesunięciem lokalnym dla każdego obszaru. Przesunięcie lokalne jest zdefiniowane następująco [8]:

$$o_x = \left\lfloor \frac{\Delta x s_x m_x - O_x}{\Delta x_n} \right\rfloor \Delta x_n - \Delta x s_x m_x + O_x$$
$$o_y = \left\lfloor \frac{\Delta y s_y m_y - O_y}{\Delta y_n} \right\rfloor \Delta y_n - \Delta y s_y m_y + O_y \quad (7)$$

gdzie s_x, s_y są indeksami obszaru.

Pierwszym krokiem w algorytmie SIMD próbkowania wtórnego będzie więc obliczenie dla każdego procesora elementarnego przesunięć lokalnych o_x i o_y. Wartości te należy następnie podstawić w miejsce przesunięcia globalnego O_x i O_y w ogólnym równaniu próbkowania [5]. Jak widać, wprowadzenie przesunięcia lokalnego do podanych wcześniej zależności na tyle ujednolica obliczenia wykonywane w każdym z obszarów, że możliwa staje się realizacja próbkowania wtórnego w systemie SIMD.

Pewną trudność stanowi także uwzględnienie przesłań danych między procesorami przy obliczeniu punktów brzegowych. Można obliczyć liczbę punktów, które muszą być przesłane do danego procesora elementarnego z procesorów sąsiednich:

$$T_{min} = 2m_x\lfloor(L_y-\Delta y_n)/\Delta y+1/2\rfloor+2m_y\lfloor(L_x-\Delta x_n)/\Delta x+1/2\rfloor+ +4\lfloor(L_y-\Delta y_n)/\Delta y+1/2\rfloor\cdot\lfloor(L_x-\Delta x_n)/\Delta x+1/2\rfloor \quad (8)$$

Pierwszy składnik sumy odnosi się do punktów w obszarach $s_x=s_xPE$ $s_y=s_yPE\pm1$ (a więc nad i pod danym procesorem elementarnym), drugi – do punktów obszarów na prawo i lewo od procesora, a trzeci – do punktów obszarów narożnych względem danego procesora elementarnego. Podstawiając do równania (8) wartości maksymalne l_x i u_x z (6) można obliczyć T_{max}. Na przykład, dla próbkowania wtórnego obrazu o wymiarach 512×512 za pomocą tablicy 16×16 procesorów elementarnych w celu otrzymania obrazu o wymiarach 1024×1024, $T_{min}=T_{max}=420$.

PRÓBKOWANIE WTÓRNE W SYSTEMIE MIMD

O wydajności obliczeniowej wieloprocesorowego systemu MIMD w znacznej mierze decyduje wybrana topologia połączeń procesorów [4, 5]. W swoich badaniach autorzy artykułu rozważają konfigurację MIMD w sieci o strukturze regularnej. Wynika to po pierwsze z właściwości procesora elementarnego, po drugie – z ukierunkowania badań na porównanie efektywności struktur SIMD i MIMD dla próbkowania wtórnego, realizowanego przez sieć procesorów o podobnej topologii połączeń.

W systemie MIMD w pamięci lokalnej każdego procesora elementarnego musi znajdować się wykonywany program. Sposób ładowania obrazu do sieci procesorów (w którego organizacji uczestniczy teraz komputer nadrzędny) jest również inny niż w wypadku systemu SIMD. Wydajność struktury MIMD przeanalizowano dla dwóch metod realizacji próbkowania wtórnego.

Pierwsza z tych metod wynika bezpośrednio z zależności podanych na początku artykułu i jest podobna do metody omówionej dla struktury SIMD. Można przyjąć, że linie obrazu pierwotnego o wymiarach $M_x \times M_y$ są kolejno przesyłane do procesorów elementarnych. Przesyłanie odbywa się w taki sposób, że w pamięci każdego procesora znajdzie się obszar $M_y/N \times M_{x_l}$ przy czym sąsiednie obszary znajdą się w sąsiednich procesorach elementarnych. Przesyłanie to organizuje komputer nadrzędny, tworząc bloki po M_y/N linii, z których każdy otrzymuje adres docelowy w postaci numeru kolumny i wiersza w tablicy procesorów oraz długości bloku. Dodatkowo komputer nadrzędny wybiera wiersz tablicy, od którego rozpoczyna przesyłanie bloku. Jakkolwiek taki sposób ładowania obrazu jest dosyć złożony (wymaga opracowania protokołu transmisji między procesorami elementarnymi), umożliwia on uzyskanie dużej współbieżności operacji wejścia-wyjścia z przetwarzaniem.

Każdy z procesorów elementarnych przetwarza obraz o wielkości **1/N OP**. Próbkowanie wtórne jest wykonywane według równania (5), przy czym przesunięcie lokalne o_x jest takie samo dla każdego procesora, zaś przesunięcie lokalne o_y trzeba obliczyć oddzielnie dla każdego obszaru według zależności (7). Jak łatwo zauważyć, omawiana metoda nie eliminuje zagadnienia przesyłania danych między procesami elementarnymi w celu obliczenia punktów brzegowych.

W każdym obszarze są teraz dwa pasy o szerokości $L_y-\Delta y/2$, takie że do obliczenia punktów **OW** są niezbędne dane z sąsiednich obszarów. Równanie (8) przyjmuje następującą postać:

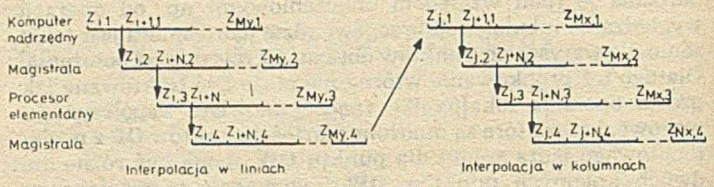
$$T_{min}=2M_x\lfloor(L_y-\Delta y_n)/\Delta y+1/2\rfloor$$

Jak wynika z tego równania, między sąsiednimi procesorami elementarnymi są przesyłane całe linie obrazu. Dla $M_x=512$, $\Delta y_n=1/2$, $\Delta y=1$, liczba przesłań T_{min} wynosi 2048.

Druga z metod realizacji próbkowania wtórnego za pomocą systemu MIMD polega na dwukrotnym wykonaniu interpolacji jednowymiarowej: najpierw w linii, a potem w kolumnie. Podczas ładowania **OP** do tablicy procesorów każda linia jest przesyłana do innego procesora. Po wykonaniu interpolacji we wszystkich liniach rozpoczyna się interpolacja kolumn. Transmisję danych organizuje komputer nadrzędny, a wyniki pośrednie są przechowywane w pamięci buforowej. Wykonanie interpolacji linii **i** składa się z następujących zadań:
Z_{i1} – inicjalizacja przesyłania linii **i** przez komputer nadrzędny,
Z_{i2} – przesyłanie linii **i** magistralami sieci,
Z_{i3} – przetwarzanie linii **i** przez procesor elementarny,
Z_{i4} – przesyłanie linii **i** do pamięci buforowej.

Wykonanie interpolacji kolumny **j** można podzielić następująco:
Z_{j1} – inicjalizacja przesyłania kolumny **j** przez komputer,
Z_{j2} – przesyłanie kolumny **j** magistralami sieci,
Z_{j3} – przetwarzanie kolumny **j** przez procesor,
Z_{j4} – przesyłanie kolumny **j** do pamięci buforowej.

Współbieżność wykonywania tych zadań w systemie MIMD zilustrowano na rys. 5. Jak widać, jedyne ograniczenie we współbieżnym wykonywaniu zadań polega na konieczności przetworzenia



Rys. 5. Podział zadań w metodzie MIMD II

wszystkich linii obrazu przez rozpoczęciem przetwarzania kolumn. W metodzie tej w większym stopniu jest obciążony komputer nadrzędny, który uczestniczy w inicjalizacji przesyłania każdej linii i każdej kolumny, a także w zapisywaniu ich do pamięci buforowej. Efektem ubocznym wykonywania próbkowania wtórnego w dwóch krokach jest obrócenie obrazu o 90°. Może ono jednak być uwzględnione przez odpowiednio wybrany algorytm operacji przetwarzania obrazów wykonywanej w następnej kolejności.

PORÓWNANIE EFEKTYWNOŚCI STRUKTUR SIMD I MIMD

W celu porównania efektywności struktur SIMD i MIMD przyjęto następujące założenia związane z implementacją sprzętową:
a) taka sama liczba procesorów elementarnych **N** w obu strukturach,
b) taki sam model procesora elementarnego,
c) możliwie podobna topologia połączeń.

Porównanie wykonano według kilku wybranych kryteriów. Są nimi:
- współbieżność całkowita systemu,
- współbieżność operacji wejścia-wyjścia i przetwarzania,
- szacunkowa liczba operacji obliczeniowych i liczba przesłań danych między procesorami elementarnymi.

Współbieżność całkowita systemu [2] jest definiowana jako iloczyn czterech współczynników cząstkowych:

$$K=k_ok_ik_nk_p$$

gdzie: k_o – współbieżność operacji, tj. liczba operacji wykonywanych jednocześnie; k_i – współbieżność obrazu, tj. liczba punktów **OW** generowanych jednocześnie; k_n – współbieżność otoczenia, tj. liczba punktów otoczenia, które mogą być jednocześnie wykorzystane (pobierane) do obliczeń punktów **OW**; k_p – współbieżność bitów, tj. długość słowa (w bitach) przetwarzanego jednocześnie. Wartości tych współczynników przedstawiono

w tabeli 1, w której MIMD I – oznacza pierwszą metodę dla MIMD, a MIMD II – drugą metodę.

Tabela 1. Wartość współczynników współbieżności

Współczynnik	SIMD	MIMD I	MIMD II
k_o	1	N	N
k_i	N	N	N
k_n	1	1	1
k_p	1	1	1
K	N	N^2	N^2

Współbieżność całkowita nie przesądza o efektywności systemu, stanowi jedynie wykładnik jego potencjalnych możliwości w danym zastosowaniu. Jak widać, struktura MIMD w obu przeanalizowanych metodach wykazuje N-razy większą współbieżność całkowitą.

Kolejnym kryterium jest współbieżność operacji wejścia-wyjścia i przetwarzania – **L**. Współczynnik ten wyraża procentowo czasową niezależność tych dwóch procesów (tab. 2).

Tabela 2. Współczynnik współbieżności operacji we-wy i przetwarzania

Współczynnik	SIMD	MIMD I	MIMD II
L	0	$\frac{N-1}{N+1}100\%$	$\frac{(M_y-1)+(M_x-1)}{(M_y+1)+(M_x+1)}100\%$

Wartość **L** dla MIMD II osiąga prawie 100% (w najkorzystniejszym wypadku tylko cztery zadania ładowania obrazu odbywają się w czasie, kiedy nie są wykonywane obliczenia: $Z_{1,1}$ – ładowanie pierwszej linii, $Z_{M_y,4}$ – przesyłanie ostatniej linii do pamięci buforowej, $Z_{j=1,1}$ – ładowanie pierwszej kolumny, $Z_{M_x,4}$ – przesyłanie ostatniej kolumny do pamięci buforowej. Ta korzystna cecha MIMD II wynika zarówno z właściwości samej metody, jak i z topologii połączeń procesorów, gwarantującej dużą przepustowość.

Do oszacowania liczby operacji obliczeniowych C_o można przyjąć, że jest ona proporcjonalna do liczby punktów **OW**, przy czym liczba operacji sumowania, mnożenia i liczenia wartości jądra splotu dla każdego punktu **OW** jest równa liczbie punktów **OP** zawartych w jądrze (tab. 3).

Tabela 3. Oszacowanie liczby operacji obliczeniowych

Współczynnik	SIMD	MIMD I	MIMD II
C_{omin}	$\lfloor 2L_x/\Delta x\rfloor\cdot\lfloor 2L_y/\Delta y\rfloor\cdot\lfloor M_x/\sqrt{N}\Delta x/\Delta x_n\rfloor\cdot\lfloor M_y/\sqrt{N}\Delta y/\Delta y_n\rfloor$	$\lfloor 2L_x/\Delta x\rfloor\cdot\lfloor 2L_y/\Delta y\rfloor\cdot\lfloor M_x\Delta x/\Delta x_n\rfloor\cdot\lfloor (M_y/N)(\Delta y/\Delta y_n)\rfloor$	$\lfloor 2L_x/\Delta x\rfloor\cdot\lfloor M_x\Delta x/\Delta x_n\rfloor + \lfloor 2L_y/\Delta y\rfloor\cdot\lfloor M_y\Delta y/\Delta y_n\rfloor$
T_{min}	$2m_x\lfloor(L_y-\Delta y_n)/\Delta y+1/2\rfloor + 2m_y\lfloor(L_x-\Delta x_n)/\Delta x_n+1/2\rfloor+4\lfloor(L_y-\Delta y_n)/\Delta y+1/2\rfloor\cdot\lfloor(L_x-\Delta x_n)/\Delta x+1/2\rfloor$	$2M_x\lfloor(L_y-\Delta y_n)/\Delta y+1/2\rfloor$	O

Liczba operacji niezbędnych do wykonania próbkowania wtórnego obrazu jest taka sama dla struktur SIMD i MIMD I. Jednak zdecydowanie najmniejsze jest C_{omin} dla MIMD II. W tabeli 3 podano także współczynnik T_{min} określający minimalną liczbę punktów, które muszą zostać przesłane do wybranego procesora elementarnego z obszarów sąsiednich. Ma on największą wartość dla MIMD I.

* * *

Jakkolwiek wymienione kryteria nie obejmują wszystkich czynników wpływających na efektywność systemu, to jednak z uwagi na ich podstawowe znaczenie jest możliwe sformułowanie wniosków dotyczących potencjalnych możliwości jednej i drugiej struktury.

- W systemie SIMD jest możliwe wykonanie globalnych operacji przetwarzania obrazów. Wymaga to jednak opracowania odpowiedniego aparatu matematycznego. Dla takich operacji system SIMD charakteryzuje się mniejszą współbieżnością całkowitą oraz mniejszą współbieżnością operacji wejścia-wyjścia i procesów obliczeniowych. Biorąc jednak po uwagę, że decydujący wpływ na efektywność przetwarzania ma liczba operacji arytmetycznych, efektywności struktury SIMD i MIMD są porównywalne.

- Dla operacji globalnych struktura MIMD charakteryzuje się większą współbieżnością. Jeśli algorytmy wykonywane w systemach SIMD i MIMD są podobne, to przewaga drugiej konfiguracji jest nieznaczna. Jednak właściwości systemów klasy MIMD umożliwiają dokładniejsze dopasowanie architektury do problemu, w szczególności wybór takiej metody rozwiązania zagadnienia, która zdecydowanie zwiększa efektywność systemu w porównaniu do SIMD.

Należy pamiętać, że dużą współbieżność operacji wejścia-wyjścia i przetwarzania charakterystyczną dla MIMD II można osiągnąć tylko dla odpowiednio dużej liczby procesorów elementarnych. Określa to równanie:

$$N\frac{C_o\, t_{śr}}{t_{pi}}+1$$

gdzie: $t_{śr}$ jest średnim czasem wykonania operacji arytmetycznej, a t_{pi} – średnim czasem przesyłania jednej linii (kolumny) do procesora. Wybór struktury SIMD lub MIMD jest zatem uzależniony od liczby procesorów elementarnych.

LITERATURA

[1] Cantoni V., Levialdi S.: Matching the task to an image processing architecture. Computer Vision, Graphics, and Image Processing, Vol. 22, pp. 301 309, 1983
[2] Danielsson P. E., Levialdi S.: Computer architecture for pictorial information systems. Computer, No. 11, pp. 53 67, 1981
[3] Danielsson P. E.: Vices and virtues of image parallel machines. Digital Image Analysis, S. Levialdi (Ed.), Pitman Publishing, London, 1984
[4] Iszkowski W.: Architektury systemów n-mikroprocesorowych, INFORMATYKA, nr 7 8, 1986
[5] Matthes W.: Multimikrorechensysteme. Radio-Fernsehen-Elektronik, Vol. 33, Nr 4, 1984
[6] Owczarczyk J., Stolarski M., Woźniak E.: Architektura współbieżnych systemów przetwarzania obrazów. INFORMATYKA, nr 7 8, 1986
[7] Owczarczyk J., Stolarski M., Woźniak E.: Procesory tablicowe współbieżne systemy przetwarzania obrazów. INFORMATYKA, nr 12, 1986
[8] Warpenburg M. R., Siegel L. J.: SIMD Image Resampling. IEEE Trans. on Computers, Vol. C-31, No. 10, pp. 208 218, 1982
[9] Whitby-Stevens C.: Transputer. Proc. 12th Annual Int. Symp. on Computer Architecture, 17 19 June 1985, pp. 292 300.

System KEE coraz popularniejszy

System inżynierii wiedzy KEE (ang. knowledge engineering environment) firmy IntelliCorp jest jednym z bardziej znanych (i drogich) narzędzi wspomagających budowę systemów z bazą wiedzy. Umożliwia on konstruowanie systemów z wykorzystaniem różnych reprezentacji wiedzy, takich jak reguły, ramy i obiekty. Departament Obrony USA złożył w firmie IntelliCorp zamówienie na budowę specjalnej wersji systemu KEE. Handlowa wersja systemu dostępna na stanowiskach komputerowych Sun-3 firmy Sun Microsystems, cieszy się dużą popularnością. Ostatnio firma McDonnell Aircraft Company zakupiły 30 kopii systemu KEE w ramach zakrojonego na szeroką skalę planu wykorzystania narzędzi sztucznej inteligencji. Piętnaście kopii systemu posłuży do budowy własnych zastosowań, pozostałe są przeznaczone do szkolenia pracowników. Podobnie angielska firma ICL zakupiła licencję na siedem egzemplarzy systemu KEE, co zwiększyło liczbę posiadanych przez nią kopii do 10. Firma zamierza budować handlowe systemy z bazą wiedzy, jak również tworzyć wewnętrzne opracowania wspomagające produkcję.

M. MACHURA

SŁAWOMIR BŁASZCZAK
Instytut Informatyki
Politechnika Warszawska

Język Occam (2)

Druga część artykułu zawiera opis dalszych mechanizmów języka Occam, a także przykładowy program, który przybliży Czytelnikowi programowanie w tym języku.

REPLIKATORY

Replikator jest mechanizmem umożliwiającym powielenie procesu podległego w konstrukcjach SEQ, PAR, ALT i IF. Powielenie to można sobie wyobrazić jako tekstowe powtórzenie procesu podaną liczbę razy. Replikator definuje się pisząc za słowem kluczowym rozpoczynającym konstrukcję, frazę:
zmienna = [baza FOR licznik]
Zmienna może być użyta (tylko do odczytu) w powielanym procesie, przy czym w kolejnych kopiach procesu jej wartość będzie się zwiększać o jeden, począwszy od wartości wyrażenia **baza**. Wyrażenie **licznik** określa liczbę powtórzeń procesu.

Chociaż wszystkie konstrukcje, z którymi może być związany replikator, wyglądają podobnie, jednak ich interpretacja jest w każdym wypadku inna. Replikator z konstruktorem SEQ umożliwia tworzenie pętli iteracyjnych. Na przykład proces:

```
VAR sum :
SEQ
  sum := 0
  SEQ i = [ 0 FOR 10 ]
    sum := sum + vec [i]
  out ! sum
```

sumuje pierwszych dziesięć elementów wektora **vec**. Zastosowanie replikatora w konstrukcji PAR umożliwia zapisanie procesów jednorodnych, tzn. procesów o takiej samej treści. Przy użyciu tablicy kanałów, mechanizm ten umożliwia tworzenie regularnych struktur procesów współbieżnych.

Proces realizujący przetwarzanie potokowe można zapisać następująco:

```
CHAN cp [ n-1 ] :
PAR
  WHILE TRUE
    VAR x:
    SEQ
      in ? x
      cp [0] ! x
  PAR i = [ 1 FOR n-2 ]
    WHILE TRUE
      VAR x:
      SEQ
        cp [i - 1] ? x
        x := x + x
        cp [i] ! x
  WHILE TRUE
    VAR x:
      SEQ
        cp [n - 2] ? x
        out ! x
```

Pierwszy i ostatni proces tego ciągu służą do komunikacji z otoczeniem (przez kanały **in** i **out**). Są one zapisane oddzielnie, w ramach zewnętrznej konstrukcji PAR.

Bardziej złożone struktury regularne można otrzymać zagnieżdżając kilka procesów z replikatorem. Na przykład, tablicę (o N wierszach i M kolumnach) procesów komunikujących się wzdłuż wierszy i kolumn można przedstawić następująco:

```
CHAN poz [N*M + N], pion [N*M + M]:
PAR i = [0 FOR N]
  PAR j = [0 FOR M]
VAR kol, wiersz:
SEQ
  kol := i*M + j
  wiersz := j*N + i
  WHILE TRUE
    VAR x,y:
    SEQ
      poz [kol] ? x
      pion [wiersz] ? y
      IF
        x>y
          poz [kol +1] ! x
          pion [wiersz + 1] ! y
        x<=y
          poz [kol + 1] ! y
          pion [wiersz + 1] ! x
```

Replikator zastosowany w konstrukcji **ALT** jest przydatny wówczas, gdy trzeba zapisać proces oczekiwania na komunikaty pochodzące z wielu kanałów połączonych w jedną tablicę i rozróżnianych indeksem. Przykładowo, następujący proces zbiera
informację z kanałów od c[0] do c[n-1] przesyłając je dalej wspólnym kanałem **ch**:

```
WHILE TRUE
  VAR x:
  ALT i = [0 FOR n]
    c[i] ? x
      ch ! x
```

Rzadziej stosuje się replikację w procesach warunkowych. W tym wypadku, aby konstrukcja miała sens, należy uzależnić warunek od zmiennej wprowadzonej przez replikator, na przykład, jako indeks tablicy. Poniżej przedstawiono proces, który przez kanał **out** przekazuje indeks elementu tablicy **tab** równy wprowadzonej wartości **x**:

```
WHILE TRUE
  VAR x:
  SEQ
    in ? x
    IF i = [0 FOR n]
      tab [i] =x
        out ! i
```

W sytuacjach, w których jest istotna tekstowa kolejność procesów podległych (jak powyższa), pierwszym procesem jest proces wykorzystujący najmniejszą wartość zmiennej replikatora. W podanym przykładzie zostanie znaleziony pierwszy równy element.

PROCESY NAZWANE

W Occamie można nadawać procesom nazwy. Nazwa umieszczona w innym miejscu programu jest równoważna treści procesu, z którym jest związana. Można również wprowadzić do definicji procesu parametry formalne, zastępowane parametrami aktualnymi w miejscu wystąpienia nazwy. Parametrami mogą być nazwy kanałów, zmiennych oraz wartości wyrażeń. Definicję procesu nazwanego umieszcza się przed procesem, w którym będzie on użyty. Definicja rozpoczyna się nagłówkiem,

w którym po słowie **PROC** podaje się nazwę oraz listę parametrów formalnych. Poniżej, z odpowiednim wcięciem, rozpoczyna się treść procesu wiązanego z nazwą. Ostatni wiersz definicji musi, tak jak w innych deklaracjach, na końcu zawierać dwukropek.

Przykład

```
PROC dodaj2 (CHAN c1, c2) =
  WHILE TRUE
    VAR x:
    SEQ
      c1 ? x
      c2 ! x + 2 :
CHAN c:
PAR
  dodaj2 (in, c)
  dodaj2 (c, out)
```

Każdy parametr formalny musi mieć określony typ. Uzyskuje się to przez poprzedzenie danej nazwy lub listy nazw słowem kluczowym **CHAN, VAR** lub **VALUE.** Dwa pierwsze informują, że odpowiedni parametr aktualny będzie kanałem lub zmienną i odwołanie do parametru formalnego jednego z tych typów będzie równoważne odwołaniu bezpośrednio do podanego kanału lub zmiennej. Parametry formalne poprzedzone słowem **VALUE** będą zastąpione wartością wyrażenia znajdującego się w odpowiednim miejscu listy parametrów aktualnych. Dwa nawiasy kwadratowe, otwierający i zamykający, umieszczone za nazwą parametru formalnego, oznaczają, że parametr aktualny będzie tablicą zmiennych lub kanałów. Rozmiar tej tablicy jest określany w chwili odwołania do procesu nazwanego.

Użycie procesów nazwanych do tworzenia struktury procesów można zilustrować następującym przykładem:

```
PROC mpx (CHAN in[], out, VALUE 1) =
  WHILE TRUE
    VAR x:
    ALT i = [0 FOR 1]
      in [i] ? x
        out ! x :
CHAN c3 [2]:
-- w otoczeniu procesu są zadeklarowane kanały:
-- c1[3], c2[2], out
PAR
  mpx (c1, c3[0], 3)
  mpx (c2, c3[1], 2)
  mpx (c3, out, 2)
```

W definicji procesu nazwanego można umieścić odwołania do innych procesów nazwanych, przy czym rekurencja jest niedozwolona. Nazywanie daje możliwość tworzenia programów w sposób bardziej systematyczny. Nazwa procesu umieszczona w pewnym miejscu programu może określać cały podsystem, bez konieczności wprowadzania zbędnych szczegółów. Sama treść może być zdefiniowana w innym miejscu.

DODATKOWE MECHANIZMY OCCAMU

Przedstawione dotychczas elementy Occamu określają język podstawowy. Prostota stosowanych konstrukcji i ubogie struktury danych mogą w wielu wypadkach utrudniać tworzenie programów. W tej sytuacji sami autorzy języka wprowadzają nowe mechanizmy ułatwiające programowanie w Occamie, przy czym rozszerzenia języka podstawowego mogą być pominięte w prostszych implementacjach.

Jedyną złożoną strukturą danych Occamu jest tablica. Dodatkowym mechanizmem jest pojęcie wycinka jako fragmentu tablicy. Wycinek definiuje się podając nazwę tablicy, a za nią, w nawiasach kwadratowych, indeks pierwszego elementu (licząc od zera) oraz liczbę elementów. Oba te wyrażenia rozdziela się słowem kluczowym **FOR.** Na przykład dla tablicy zadeklarowanej jako:

```
VAR vec [10]
```

można określić trzysłowowy wycinek:

```
vec [0 FOR 3]
```

Jeśli tablica jest bajtowa, to również powstałe z niej wycinki muszą być zaznaczone jako bajtowe:

```
bvec [BYTE 1 FOR 10]
```

Wycinki można też tworzyć na podstawie tablic stałych, np.:

```
DEF nazwa = "SYSTEM 1", ...
    :
  nazwa [BYTE 1 FOR 6]
```

Wycinki mogą być kopiowane w całości w procesach przypisania lub przez realizację procesów wejściowych i wyjściowych, np.:

```
vec [0 FOR 10] := vec1 [3 FOR 10]
```

co jest równoważne:

```
ch ! vec1 [3 FOR 10]
```

i

```
ch ? vec [0 FOR 10]
```

wykonanych w dwóch procesach współbieżnych. W obydwu wypadkach długości wycinków – źródłowego i docelowego – muszą być takie same i wycinek docelowy nie może być tworzony na podstawie tablicy stałych.

Stosowanie wycinków może w wielu wypadkach uprościć zapisywanie programów, zwłaszcza w miejscach, gdzie muszą być wykonane operacje na tekstach. Przykładowo, poniższy proces zamienia numer urządzenia na jego nazwę symboliczną:

```
PROC zamien (CHAN in, out) =
  DEF u1 = "KB", u2 = „DY", ...
WHILE TRUE
  VAR nr:
  SEQ
    in ? nr
    IF
      nr = 1
        out ! u1 [BYTE 0 FOR 3]
      nr = 2
        out ! u2 [BYTE 0 FOR 3]
      :
      :
```

Drugi dodatkowy mechanizm wprowadzony do Occamu pozwala w większym stopniu wiązać program z systemem, w którym ma pracować. Ogólnie przyjmuje się, że program w Occamie może być wykonany w dowolnym systemie, zarówno jednoprocesowym, jak i rozproszonym. Niekiedy jednak, zwłaszcza w wypadkach, gdy system jest złożony z różnych elementów, istnieje potrzeba określania zbioru procesów przeznaczonych dla pewnego konkretnego procesora. W ten sposób cały program można podzielić na procesy, z których każdy grupuje akcje związane z jednym procesorem. W zapisie taki program rozpoczyna się frazą **PLACED PAR**, za którą (w następnych wierszach) są podawane wszelkie informacje identyfikujące procesory oraz wiążące kanały z fizycznymi portami, przez które procesor komunikuje się z otoczeniem. Dla pewnego systemu, złożonego z dwóch procesorów, fragment jego definicji może mieć następującą postać:

```
PLACED PAR
  CHAN c1, c2, c3:
  PLACED PAR
    ALLOCATE 0          -- procesor nr 0
    PORT 0 :- c1        -- wiązanie portów z kanałami
    PORT 1 :- c2
    LOAD 0 :            -- oznaczenie portu,
    -- przez który będzie ładowany program
  proces0 (c1, c2)      -- proces pierwszego procesora
  ALLOCATE 1            -- procesor nr 1
    PORT 0 :- c2
    PORT 1 :- c3
    LOAD 0:
  proces1 (c2, c3)      -- proces drugiego procesora
```

W ramach jednego procesora procesy działają na zasadzie podziału czasu. Niekiedy jest istotne, w jakiej kolejności będzie następował przydział procesora. W Occamie programista może uszeregować procesy według priorytetów, korzystając z konstruktora **PRI PAR**. Tekstowa kolejność podległych tej konstrukcji wyznacza stopień ich pilności; najbardziej uprzywilejowany jest proces pierwszy. Na przykład, obsługa urządzeń umieszczona w procesie **wej.wyj.** będzie wykonywana z wyższym priorytetem niż procesy systemu plików:

```
PRI PAR
  wej.wyj
  system.plików
```

Dla jednego procesora konstrukcja **PRI PAR** może być użyta tylko raz.

Priorytety mogą być również określane w procesach **ALT**. Umożliwia to jednoznaczne zdefiniowanie kolejności przyjmowania komunikatów w sytuacji, gdy w tym samym czasie więcej niż jeden dozór jest gotowy. Poniższy proces modeluje układ przerwań procesora; przerwanie **int0** ma najwyższy priorytet, a **int3** najniższy:

```
PRI ALT
  int0 ? ANY
    -- obsługa 0
  int1 ? ANY
    obsługa 1
  int2 ? ANY
    -- obsługa 2
  int3 ? ANY
    -- obsługa 3
```

PRZYKŁAD PROGRAMU

Przedstawiony poniżej przykład jest ilustracją zastosowania Occamu do implementacji pewnego systemu, a także – metody tworzenia programów w tym języku. Programy w Occamie, zgodnie z nowoczesną metodologią programowania, powinny być dzielone na moduły. Każdy moduł udostępnia pewien zbiór operacji logicznie ze sobą powiązanych, np. implementujących struktury danych lub obsługujących urządzenia. W Occamie modułem jest proces. Dane do realizacji jego funkcji są mu przekazywane przez kanały; tą drogą są także udostępniane wyniki przetwarzania. Kanały służą również do określania, jaka operacja ma być wykonana. Same procesy działają cyklicznie, w pętli nieskończonej. Punktem „spoczynkowym" tej pętli jest z reguły konstrukcja **ALT** (lub w prostszym wypadku proces wejściowy), w której proces czeka na polecenia. Takie podejście do tworzenia programów niewątpliwie upraszcza projektowanie i uruchamianie systemów. Przedstawiona metoda umożliwia również opisywanie systemów w całości, tzn. łącznie z elementami sprzętowymi, których logika może być zapisana w modułach. Umożliwia to testowanie systemu jeszcze przed jego fizyczną realizacją (przy użyciu symulatora), a jednocześnie pozostawia swobodę wyboru, co będzie sprzętem, a co pozostanie programem.

Jako przykład przedstawiono program implementujący **stoper**, czyli urządzenie włączające alarm, gdy upłynie określony przez użytkownika czas. Czas podawany w minutach jest wprowadzany do systemu za pomocą klawiatury, złożonej z 10 klawiszy cyfr (numerowanych od 0 do 9) oraz klawiszy: **START** naciskanego po wprowadzeniu czasu i **ZERUJ** – służącego do zatrzymywania stopera i ustawienia go w stan początkowy. Wewnętrzny zegar układu zwiększa się o 1 co 20 ns.

Cały system można podzielić na trzy procesy działające współbieżnie:

- klawiatura – proces obsługujący klawiaturę,
- zegar – proces odmierzający podany czas,
- sterowanie – proces sterujący całym systemem.

Proces klawiatura odbiera sygnały z klawiatury przez kanały zgrupowane w tablicy **klawisz** i następnie przekazuje kanałem kod wciśniętego klawisza:

```
PROC klawiatura (CHAN klawisz[],kod)=
  WHILE TRUE
    ALT i = [0 FOR 12]  -- 12 klawiszy
      klawisz[i] ? ANY  -- gotowy,
                        -- gdy klawisz i wciśnięty
        kod ! i         -- wysłanie kodu
```

Proces zegar odmierza czas określony przez wartość przekazaną kanałem start. Gdy upłynie podany czas, proces wysyła sygnał kanałem stop.

```
PROC zegar (CHAN start, stop) =
  DEF interwał = 300:  -- liczba jednostek czasu w min
  VAR czas, licznik:
  SEQ
    licznik := 0
    WHILE TRUE
      ALT
        licznik>0 & WAIT NOW AFTER czas
        SEQ
        czas := czas + interwał          -- następna minuta
        licznik := licznik - 1
        IF
        licznik = 0
        stop ! ANY
        start ? licznik
        czas := NOW + interwał           -- włączenie zegara
```

Proces sterowania odbiera i rozpoznaje kod wprowadzony z klawiatury, steruje pracą zegara oraz włącza alarm, gdy upłynie podany czas.

```
PROC sterowanie (CHAN klaw, start, stop, alarm) =
  VAR czas, n:
  SEQ
    czas := 0
    WHILE TRUE
      ALT
        klaw ? n                 -- numer wciśniętego klawisza
          IF
            n>0 AND n<10                    -- klawisz cyfry
              czas := czas*10 + n
            n = 10                          -- klawisz START
              SEQ
                start ! czas                -- włączenie zegara
                czas := 0
            n= 11                           -- klawisz ZERUJ
              SEQ
                start ! 0                   -- zatrzymanie zegara
                czas := 0
        stop ? ANY               -- koniec odmierzania czasu
          alarm ! ANY :
```

Proces główny można zapisać następująco:

```
CHAN klawisz [12], klaw, start, stop, alarm :
PAR
  sterowanie (klaw, start, stop, alarm)
  klawiatura (klawisz, klaw)
  zegar (start, stop)
```

W procesach nazwanych można zrezygnować z parametrów, umieszczając wszystkie definicje procesów za wierszem deklaracji kanałów i odwołując się w nich bezpośrednio do tych kanałów.

Każdy z przedstawionych procesów może być wykonywany przez procesor, ale również może to być tylko opis jednego z układów, np. klawiatury lub zegara.

* * *

W obecnej chwili Occam należy raczej traktować bardziej jako eksperyment i propozycję niż profesjonalne narzędzie. Tworzenie złożonych programów utrudniają zwłaszcza ubogie struktury danych. Z tego względu język ten jest używany głównie tam, gdzie powstają i są badane eksperymentalne systemy o architek-

dokończenie na s. 16

ANDRZEJ TUROWICZ
Ośrodek Informatyki
Urzędu Wojewódzkiego
Wrocław

Lokalna sieć komputerowa D-Link (3)

Ostatnia część artykułu zawiera alfabetyczny wykaz poleceń usługowych sieci D-LINK. Przyjęto następujące oznaczenia:
[...] – zawartość nawiasów może, lecz nie musi, zostać użyta w poleceniu;
⟨...⟩ – składnik polecenia obowiązkowy;
nazwa l – logiczna nazwa urządzenia, tzn. **LPTn, A-Z, M;**
nazwa f – fizyczna nazwa rządzenia, tzn. **Pn, Hn, Mn** (przy czym **n** oznacza numer urządzenia).
Parametry polecenia rozdziela się kreskami ukośnymi "/". Parametr pominięty pomija się wraz z kreską.

ATTN

Przesyłanie informacji między komputerami w sieci lub powtórzenie ostatnio odebranej informacji.

Składnia:

ATTN – wywołanie menu przesyłania informacji,
ATTN/D – powtórzenie ostatniego komunikatu,
ATTN ALL, ⟨komunikat⟩ – przesłanie do wszystkich,
ATTN⟨[numer komputera] [nazwa zadania]⟩, ⟨komunikat⟩ – przesłanie do określonego komputera lub zadania.

Przykłady:

attn 001,Koniec – przesłanie do komputera 1 tekstu „Koniec",
attn all, 12.30 – przesłanie **"12.30"** do wszystkich,
attn /D – żądanie powtórzenia komunikatu.

CONNECT

Dołączenie lub odłączenie urządzenia innego komputera pracującego w sieci.

Składnia:

CONNECT – wywołanie menu dołączania,
CONNECT ⟨nazwa l⟩ – odłączenie urządzenia,
CONNECT ⟨nazwa l⟩ ⟨[nr komputera] [nazwa zadania]⟩,⟨nazwa f⟩, [/R],[/W],[/L],[/P[⟨hasło⟩]] – dołączenie urządzenia, z podaniem trybu dostępu: **/R** – odczyt, **/W** – zapis, **/L** – tylko dla siebie prawo zapisu.

Przykłady:

connect f – odłączenie logicznego dysku F,
connect lpt2/001/p1/W – dołączenie jako **LPT2** drukarki **p1** komputera nr **001** do zapisu.

INSTIMER

Pobieranie daty i czasu z sieci. Pierwszy komputer wykonujący to polecenie narzuca czas całej sieci. Służy to do synchronizacji zegarów komputerów w sieci. Komputer bez zegara czasu rzeczywistego może w ten sposób uzyskać czas i datę. Tylko jeden komputer w sieci powinien wykonać to polecenie. Aby zmienić czas sieci należy wydać polecenie ponownie. Zegary pracujących komputerów pozostaną w stanie niezmienionym.

LOCK

Zawłaszczenie urządzenia zdalnego, które jest niedostępne dla innych, pomimo zezwoleń. Urządzenie musi być najpierw przydzielone poleceniem **CONNECT**. Zwykle dotyczy to drukarek.

Składnia:

LOCK⟨nazwa l⟩ – urządzenie staje się niedostępne dla innych komputerów

Przykłady:

lock d – zawłaszczenie dysku **D:**
lock lpt2 – drukarki **LPT2.**

LOGOFF

Wyrejestrowanie komputera z sieci.

Składnia:

LOGOFF – wywołanie menu wyrejestrowania komputera z sieci,
LOGOFF ⟨nazwa zadania⟩ – wyrejestrowanie komputera.

LOGON

Zarejestrowanie komputera w sieci, rozpoczęcie sesji sieciowej. Zadania muszą mieć jednoznaczne nazwy.

Składnia:

LOGON – wywołanie menu rozpoczęcia sesji
LOGON ⟨nazwa zadania⟩ – rozpoczęcie zadania z podaną nazwą.

Jeżeli na dysku bieżącym znajduje się zbiór **Auto-Logon** utworzony programem **ALC**, to program pyta o hasło; po podaniu prawidłowego hasła uruchamia zadanie z numerem grupy i indywidualnym, takim jak zapamiętał **ALC**. Jeżeli nie ma pliku sterującego, to zadanie będzie miało numer grupy 255 i numer indywidualny 255.

Przykłady:

logon – wywołanie menu rozpoczęcia sesji,
logon Henryk – rozpoczęcie zadania o nazwie Henryk.

LPLOCK

Badanie czy urządzenie jest niezawłaszczone. Jeżeli inny komputer je zawłaszczył, to jest wyświetlany komunikat:

Device locked by another user Waiting ... (or press (ESC) to abort)

Można czekać aż urządzenie będzie wolne lub zrezygnować z niego naciskając klawisz **ESC**.

Składnia:

LPLOCK ⟨nazwa l⟩

Przykłady:

lplock d – sprawdzenie dostępności dysku **D**,
lplock LPT3 – drukarki numer **3**.

NETSAVE

Zapisanie do pliku **AUTOEXEC.BAT** stanu praw dostępu i przyporządkowań urządzeń innych komputerów. Komputer będzie

mógł automatycznie wykonywać procedury dołączania się do sieci.

NETVER

Podanie numeru wersji **D-Link**, a właściwie numeru wersji programu sterującego **D-LINK.DRV**.

SETDEV

Nadanie praw dostępu do urządzeń.

Składnia:

SETDEV⟨nazwa f⟩[/A⟨[/R][/W]⟩[/P[hasło]]
[/G⟨numer grupy⟩/S⟨numer indywidualny w grupie⟩]
[/X[R] [W]] [/Y[R][W]][/Z[R] [W]] [/L][U]

Prawo dostępu do urządzenia o nazwie **f**; **A** podaje jakie prawo dostępu przysługuje po podaniu hasła; **G** i **S** określają dla właściciela grupy numer i numer indywidualny w grupie; **X** podaje prawa dostępu dla właściciela urządzenia; **Y** podaje prawa dostępu dla członków grupy; **Z** podaje prawa dostępu innych; **L** zawłaszcza urządzenie, **U** zwalnia urządzenie.

Przykłady:

SETDEV H1/G001/S001/XW/YW/Zr – dysk **H1** staje się dostępny właścicielowi i grupie do zapisu, a innym do odczytu,
SETDEV P1 /XW/YW/ZW – drukarka **P1** jest ogólnodostępna.

SPL

Uaktywnienie gromadzenia wydruków na dysku stałym.

Składnia:

SPL – wywołanie menu
SPL [nazwa pliku buforowego],
[nazwa pliku strony oddzielającej] [/Sn] [/Um] [/Qk]
Domyślne nazwy plików to **SPL.BUF** i **SPL.SEP**;
n oznacza liczbę kilobajtów przeznaczonych na bufor [50–1024], wartość domyślna 200;
m oznacza liczbę użytkowników bufora [1 – 255], wartość domyślna 10;
k oznacza liczbę wydruków w kolejce [1 – 255], wartość domyślna 20.

Przykłady:

spl – wywołanie menu; może zapisać parametry do **AUTOEXEC.BAT**,
spl plik.drk, strona.sep /S1024 – bufor o wielkości 1 MB, o nazwie **plik.drk**; strona separująca – w pliku **strona.sep**.

SPLER

Gromadzenie kolejnych wydruków w jeden pakiet, do czasu sygnalizacji końca pakietu. Pakiet jest drukowany po zakończeniu całości.

Składnia:

SPLER [/M] [/E]
przy czym **M** oznacza początek pakietu, **E** – koniec pakietu.

SPL STAT

Wywołanie menu statystyki bufora drukarki oraz zmiany kolejności wydruku dokumentów.

UNLOCK

Przywrócene sieci zawłaszczonych urządzeń.

Składnia:

UNLOCK ⟨nazwa l⟩

Przykłady:

unlock d – odblokowanie dysku **D**:
unlock lpt1 – odblokowanie drukarki numer 1.

USER

Wywołanie menu pokazującego stan sieci: które komputery są dołączone do sieci, z jakimi numerami grup oraz numerami indywidualnymi w ramach grupy i jakie nazwy mają zadania uruchomione w sieci.

* * *

Na zakończenie wypada powiedzieć o wadach i zaletach sieci D-Link, przy czym należy rozróżnić wady i zalety sieci jako takiej od wad i zalet D-Link.

Do wad sieci należy zaliczyć:

- możliwość niepowołanego dostępu do danych,
- brak dostępu do części danych, jeżeli nie pracuje komputer zawierający ją,
- spowolnienie pracy urządzenia podczas jednoczesnego wykorzystywania przez kilka komputerów,
- możliwość kolizji w dostępie do informacji.

Zaletami sieci są natomiast:

- możliwość zdalnego i wielodostępnego przetwarzania,
- dostęp do urządzeń innych komputerów,
- możliwość przesyłania dokumentów między komórkami organizacyjnymi,
- poczta elektroniczna,
- możliwość pracy systemu informatycznego mimo awarii jednego z komputerów; brak jest jedynie dostępu do tej części informacji, którą przechowuje uszkodzony komputer (można jednak te same dane przechowywać w kilku komputerach na wypadek awarii lub wyłączenia).

Do wad i zalet sieci komputerowych D-Link dodaje swoje niedogodności. Należy o nich pamiętać projektując systemy wielokomputerowe i – o ile to możliwe – łagodzić ich skutki w swoich programach.

Wady D-Link są następujące:

- wymagana zgodność programu sterującego **D-LINK.DRV** z wersją systemu operacyjnego, która często się zmienia,
- niewielki zasięg terytorialny sieci,
- brak obsługi woluminów powyżej 360 KB, pomimo zapewnień w dokumentacji[1],
- konieczność dysponowania dodatkowym oprogramowaniem sieciowym PC-NET oraz programami synchronizacji czasu w sieci.

Do zalet D-Link należą natomiast:

- niewielkie wymagania dotyczące kabla magistrali,
- tanie karty sieciowe i kabel,
- stosunkowo duża szybkość transmisji,
- duży zestaw usług sieciowych,
- prosta w obsłudze i zastosowaniu, bardzo dobrze opracowana dokumentacja.

[1] Autor przyznaje, że być może nie umiał tego zrobić.

Język Occam

dokończenie ze s. 14

turze rozproszonej. Jest on również dobrym narzędziem do zapisywania i testowania algorytmów współbieżnych oraz może być pomocny w nauczaniu programowania współbieżnego. Z drugiej strony, Occam można traktować jako język maszynowy systemów wieloprocesorowych, a nawet transputerów. Tym samym może on być językiem docelowym w translacji innych języków wysokiego poziomu, np. Ady lub Moduli, choć oczywiście takie przekształcanie jest problemem niełatwym.

JAN BIELECKI
Warszawa

Turbo C

Zmienne ustalone, zmienne nietrwałe, punkty charakterystyczne

Każdy, kto programował w języku C, zna modyfikatory **short**, **long** i **unsigned**. Ich użycie umożliwia posługiwanie się typami dodatkowymi, takimi jak **(long float)**, **(usigned int)** itp. W języku Turbo C wprowadzono wiele dodatkowych modyfikatorów, a wśród nich **const** i **volatile**. Pierwszy z nich umożliwia deklarowanie zmiennych zachowujących się jak stałe (ale nie będących stałymi), a drugi umożliwia deklarowanie zmiennych, którym dane mogą być przypisywane w sposób niekontrolowany. Stosownie do tych właściwości, zmienne zadeklarowane z modyfikatorem **const** będą nazywane zmiennymi ustalonymi, a zmienne zadeklarowane z modyfikatorem **volatile** będą nazywane nietrwałymi[1].

ZMIENNE USTALONE

Zmienne ustalone zawdzięczają swoją nazwę temu, iż można im przypisywać dane, ale jedynie podczas oprącowywania deklaracji. Po wykonaniu tej czynności, zmienna ustalona zachowuje się jak stała.

Przykłady:

A. Zmienna ustalona **Pi**, której podczas opracowywania deklaracji zostanie przypisana dana reprezentowana przez liczbę 3,141593

```
const float Pi = 3.141593;
```

B. Zmienna ustalona **Min**, której podczas opracowywania deklaracji zostanie przypisana dana reprezentowana przez liczbę –32768

```
int const Min = −32768;
```

C. Zmienna ustalona **Str**, której podczas opracowywania deklaracji zostanie przypisane wskazanie pierwszego znaku danej reprezentowanej przez napis **"Hello, I am JanB"**

```
char * const Str = "Hello, I am JanB";
```

Ponieważ **Pi** jest zmienną ustaloną, niepoprawne jest wykonanie takiej instrukcji, jak:

```
Pi++;
```

Z analogicznych powodów jest niedozwolone wykonanie np. instrukcji:

```
Str = "Hello world";
```

mimo iż jest dozwolone wykonanie instrukcji:

```
*Str = '\0';
```

a to dlatego, że ***Str** nie jest zmienną ustaloną. Należy zauważyć, że ***Str** byłoby taką zmienną, gdyby deklaracji zmiennej **Str** nadano postać:

```
char const * const Str = "Hello, I am JanB";
```

W takim wypadku błędne byłoby nie tylko wykonanie instrukcji:

```
*Str = '\0';
```

ale także wykonanie instrukcji:

```
Str[i] = '\0';
```

i to dla dowolnego **i**.

Zasady ogólne

1. Wyrażenie, które reprezentuje zmienną ustaloną może być argumentem operatora wskazania (**&**), ale nie jest **l**-wyrażeniem.

```
# include <stdio.h>
main
{
  int const Num = 13:
  Fun(&Num);
  printf ("%d", Num);
}

Fun (par)
int*par
{

(*par)++
```

Program jest błędny, ponieważ nie tylko argument funkcji **Fun** nie jest zgodny z typem jej parametru, ale również dlatego, że wyrażenie *par reprezentuje zmienną ustaloną, a więc nie może być argumentem operatora inkrementacji.

2. Wskazanie zmiennej, która nie jest zmienną ustaloną, może zostać przypisane zmiennej wskazującej zmienne ustalone:

```
# include <stdio.h>
main()
{
  int Dec = 13;
  int *Ref = &Dec;
  int const *Ptr;

  Ptr = Ref;
  printf ("%d", *Ptr);
}
```

Program jest poprawny, a jego wykonanie powoduje wyprowadzenie liczby 13. Należy zauważyć, że mimo iż Ptr = = Ref poprawne byłoby wykonanie instrukcji:

```
(*Ref)++;
```

ale niepoprawne byłoby wykonanie instrukcji:

```
(*Ptr)++;
```

a to dlatego, że ***Ptr** reprezentuje zmienną ustaloną.

[1] Redakcja zdecydowała się utrzymać termin zmienna nietrwała, wbrew wyraźnemu stanowisku Autora, który preferuje termin zmienna ulotna. Czynimy tak dlatego, że informatyczny termin **nietrwały** jako odpowiednik ang. volatile występuje już od dawna w słownikach. To jest w „Słowniku informatyki", WNT, 1981 oraz – co ważniejsze – w znaczeniu informatycznym w „Słowniku naukowo-technicznym angielsko-polskim", WNT, 1983. W dwóch wydanych dotychczas książkach Autora o języku C („Język C – interpretacja standardu", WNT, 1987; „Sam na sam z językiem C", WKiŁ, 1988) termin volatile nie występuje. (przyp. red.)

3. Zabrania się, aby wskazanie zmiennej ustalonej zostało przypisane zmiennej wskazującej, która nie wskazuje zmiennych ustalonych:

```
#include <stdio.h>
main()
{
  const int Dec = 13;
  int *Ref = &Dec;

  (*Ref)++;
  printf("%d", *Ref);
}
```

Program jest błędny, ponieważ **&Dec** reprezentuje wskazanie zmiennej ustalonej, a **Ref** jest zmienną, której mogą być przypisywane jedynie wskazania takich zmiennych, które nie są zmiennymi ustalonymi.

Przykład

```
#include <stdio.h>
int const Arr[2] = { 12,13 },
      *Ptr = Arr;

main()
{
  const int * const Ref = Ptr;

  printf("%d", *Ref);
}
```

Wykonanie programu powoduje wyprowadzenie liczby 12. W programie zadeklarowano dwuelementową tablicę zmiennych ustalonych typu **(int)**, zmienną **Ptr** wskazującą zmienne ustalone typu **(int)** oraz zmienną **Ref** wskazującą zmienne ustalone typu **(int)**. Daną przypisaną zmiennej **Ref** dobrano w taki sposób, że wskazuje ona ten sam obiekt co **Ptr**, a więc pierwszy element tablicy **Arr**. Należy zwrócić uwagę, że w zasięgu przytoczonych deklaracji można posłużyć się, na przykład, instrukcją:

```
Ptr++;
```

ale nie można by posłużyć się instrukcją:

```
Ref++;
```

a to dlatego, że **Ref** jest zmienną ustaloną. Z analogicznych powodów nie byłoby dozwolone wykonanie istrukcji:

```
Arr[1]++;
```

ani instrukcji:

```
(*Ref)++;
```

ZMIENNE NIETRWAŁE

W odróżnieniu od zmiennych ustalonych, które w znacznym stopniu przypominają stałe, a więc nie podlegają modyfikowaniu, zmienne nietrwałe są nie tylko zmiennymi w pełnym tego słowa znaczeniu, ale co więcej, mają tę właściwość, że przypisywanie im danych może zachodzić w sposób niekontrolowany.

Pod względem składniowym, użycie modyfikatora **volatile** nie różni się od użycia modyfikatora **const**. Nic również nie stoi na przeszkodzie równoczesnego użycia tych modyfikatorów.

Modyfikator **volatile** jest używany najczęściej w tych wypadkach, gdy dotyczy zmiennej utożsamionej z portem wejścia-wyjścia, albo zmiennej podlegającej modyfikowaniu przez funkcję obsługującą przerwania asynchroniczne. Jeśli program służy do przetwarzania zmiennych ustalonych, które są nietrwałe, a którym przypisano dane reprezentowane przez wyrażenia stałe, to wszystkie takie zmienne mogą zostać umieszczone w pamięci typu ROM.

Zasady ogólne

1. Program zawierający wyrażenia reprezentujące zmienne nietrwałe musi być interpretowany w taki sposób, aby w każdym punkcie charakterystycznym, dana przypisana zmiennej nietrwałej była dokładnie taka sama jak dana, która zostałaby przypisana takiej zmiennej w wypadku, gdyby program był wykonywany ściśle według jego zapisu, tj. bez dokonywania jakichkolwiek optymalizacji. Tytułem wyjaśnienia należy dodać, że punktami charakterystycznymi programu są:
a) miejsce, w którym zakończono opracowywanie argumentów funkcji,
b) miejsce wystąpienia dowolnego operatora koniunkcji **(&&)**, alternatywy (||), warunku **(?)** i połączenia (,),
c) miejsce, w którym zakończono opracowywanie kompletnego wyrażenia, w tym wyrażenia inicjującego zmienną automatyczną, wyrażenia stanowiącego instrukcję wyrażeniową, wyrażenia występującego po słowie kluczowym **if** albo **switch** oraz dowolnego wyrażenia określającego przebieg wykonywania instrukcji iteracyjnej **(while, do for)**.

2. Wskazanie zmiennej, która nie jest zmienną nietrwałą, może zostać przypisane zmiennej wskazującej zmienne nietrwałe.

3. Zabrania się, aby wskazanie zmiennej nietrwałej zostało przypisane zmiennej wskazującej, która nie wskazuje zmiennych nietrwałych.

Przykład

```
#include <stdio.h>
#include <dos.h>
main()
{
  unsigned volatile far * const Adr = MK_FP(0x40,0x6C);
  register Fix = *Adr;
  printf ("%u", Fix);
  printf ("%u", *Adr);
}
```

Wykonanie programu powoduje dwukrotne wyprowadzenie wartości danej wskazywanej przez **Adr**. Ponieważ wskazywana zmienna ma charakter nietrwały, wyprowadzone wartości mogą być różne. Jest to nawet bardzo prawdopodobne, ponieważ w komputerze IBM PC, w słowie o adresie 40:6C, jest przechowywanych 16 mniej znaczących bitów zegara czasu rzeczywistego, aktualizowanego asynchronicznie z wykonywaniem programu. Z tego powodu, błędne byłoby zoptymalizowanie pary instrukcji **printf** do postaci:

```
printf("%u ", Fix);
printf("%u", Fix);
```

Kanoniczny krok procesu programowania

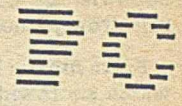
dokończenie ze s. 4

naturalne warunki (szczegóły, jak zwykle, w książce [10]), to **P'** okazuje się po prostu **I(D)** i nowe zadanie ma banalnie proste rozwiązanie, włączające bez zmiany wyniki uprzednio wykonanej pracy. Oczywiście, nie wszystkie zmiany **S** dają się równie łatwo zrealizować!

LITERATURA

[1] Cohen B., Harwood W. T., Jackson M. I.: The Specification of Complex Systems. Addison-Wesley, Reading (MA), 1986
[2] Gehani N., McGettrick A. D. (Eds.): Software Specification Techniques. Addison-Wesley, Reading (MA), 1986
[3] Goguen J. A., Meseguer J.: Rapid Prototyping in the OBJ Executable Specification Language. SRI Technical Report CSL-137, Menlo Park (CA), 1982
[4] Guttag J. V., Horning J. J., Wing J.: Some Notes on Putting Formal Specifications to Productive Use. Science of Computer Programming, Vol. 2, No. 1, 1982
[5] Lehman M. M., Stenning V., Turski W. M.: Another Look at Software Design Methodology. ACM Software Engineering Notes, Vol. 9, No. 2, pp. 38-53, April 1984
[6] Maibaum T. S. E., Turski W. M.: On What Exactly Goes On When Software is Developed Step-by-Step. Proc. 7th Intern. Conf. on Software Engineering, Orlando, (Fl), pp. 528-533, 1984
[7] Turski W. M.: Specification as a Theory with Models in the Computer World and in the Real World. INFOTECH State of Art Report, IX (6), pp. 363-377, 1981
[8] Turski W. M.: Design of Large Programs. Pp. 129-160, Software engineering Entwurf und Spezifikation (C. Floyd, H. Kopetz, Hrsg.), Teubner Verlag, 1981
[9] Turski W. M.: O informatycznym sposobie rozwiązywania zadań nie tylko matematycznych. Referat plenarny na Zjeździe PTM, Kielce, wrzesień 1985. Sprawozdania Instytutu Informatyki Uniwersytetu Warszawskiego, nr 148
[10] Turski W. M., Maibaum T. S.: The Specification of Computer Programs. Addison-Wesley, Reading (MA), 1987.

Użycie funkcji DOS-a w Turbo Pascalu



W artykule opisano wywołania funkcji systemu operacyjnego w programach pisanych w Turbo Pascalu i działających pod nadzorem systemu PC-DOS lub MS-DOS. Przedstawione metody można stosować w innych implementacjach Pascala lub innych językach wysokiego poziomu mających możliwości odwoływania się do systemu operacyjnego i sprawdzania wartości zapisanych w rejestrach jednostki centralnej.

Program **DirectoryDemo** wydruki (1–6) podaje zawartość skorowidza dysku (nazwy plików i ich wielkości) szukając zgodności z podanym wzorcem w sposób podobny do polecenia systemowego DIR. Procedury i funkcje tworzące ten program ilustrują wywołania funkcji DOS-a i mogą również służyć w innych programach do automatycznego przetwarzania grup plików lub przekazywania listy dostępnych plików wraz z informacjami o nich.

DOS i języki wysokiego poziomu

DOS udostępnia pewną liczbę funkcji do obsługi operacji we-wy z konsoli, urządzeń pamięci masowej i innych. Programista posługujący się językiem asemblera ma standardowe środki do ich wyrażania. Języki wysokiego poziomu również korzystają z tych funkcji, aczkolwiek w inny sposób. W programie napisanym w języku asemblera, przed wydaniem do DOS-a polecenia otwarcia pliku, trzeba pamiętać o licznych szczegółach, takich jak ustawienie bloku kontrolnego pliku (FCB) itp.

System operacyjny tak bogaty, jak MS-DOS lu PC-DOS, udostępnia dużo więcej funkcji obsługi we-wy i pamięci niż jest dostępnych w językach wysokiego poziomu. Wynika to z dwóch powodów. Po pierwsze, różne systemy operacyjne mają różne funkcje, a pisanie przenośnego oprogramowania łączy się ze stosowaniem funkcji wspólnych dla wszystkich systemów operacyjnych. Po drugie, muszą istnieć ograniczenia możliwości oferowanych przez język wysokiego poziomu, aby był on możliwy do opanowania. Oczywiście, przenośność i prostotę uzyskuje się kosztem pewnych, użytecznych możliwości.

Implementacje niektórych języków wysokiego poziomu są wyposażone w mechanizmy dostępu do funkcji systemu operacyjnego lub – wykonywania zadań nie wyrażalnych bezpośrednio w języku. Zwykle w tym celu stosuje się wplatanie kodu w języku asemblera lub konsoliduje moduły języka asemblera z programami skompilowanymi. Pełne wykorzystanie takich możliwości łączy się więc z programowaniem w języku asemblera.

Turbo Pascal także pozwala na wplatanie w programie kodu asemblerowego. Jednakże ta właściwość zasadniczo nie jest potrzebna, ponieważ język zawiera konstrukcje pozwalające odwołać się do systemu operacyjnego, wraz z zapisem i odczytem rejestrów jednostki centralnej. Tym samym, bez uciekania się do języka asemblera można pisać procedury i funkcje wykonujące zadania nie przewidziane bezpośrednio w języku. Cena, jaką się płaci za wykorzystanie tej możliwości, polega na utracie przenośności oprogramowania do innych systemów operacyjnych.

DOS zawiera zestawy funkcji zarządzania plikami. Zestaw tradycyjny jest ukierunkowany na operowanie blokiem FCB. Żeby wykorzystać te odwołania, program aplikacyjny musi utworzyć FCB i przesłać go do DOS-a. W wersji 2.0 systemu DOS wprowadzono nowy, rozszerzony zestaw funkcji łatwiejszych w użyciu. Szczegółowe omówienie funkcji DOS-a można znaleźć w literaturze.

Organizacja pamięci

Znajomość organizacji pamięci w komputerach wykorzystujących mikroprocesory firmy Intel rodziny 8088/8086 jako jednostki centralne jest niezbędna do zrozumienia działania funkcji systemu DOS.

Rejestry jednostki centralnej są 16-bitowe i mogą zawierać liczby całkowite bez znaku z przedziału od 0 do 65535. Jednostka centralna może jednak adresować więcej niż 64 KB dzięki wykorzystaniu 20-bitowych adresów segmentowych złożonych z adresu bazowego segmentu (wielokrotność 16) i wyrównania względnego do tej bazy. Za pomocą takiej techniki jest możliwy dostęp do każdego bajtu megabajtowej przestrzeni adresowej jednostki centralnej. Jednostka centralna dzieli ten obszar na cztery segmenty używane jednocześnie: segment kodu, danych, stosu i segment dodatkowy. Odwołanie do każdego z tych obszarów odbywa się przez parę rejestrów – jeden zawiera adres segmentu, a drugi wyrównanie.

DOS zarządza pamięcią w sposób dynamiczny. Aktualny adres rezydowania programu nie jest ustalany, aż do chwili jego wykonania. Podczas ładowania programu do pamięci DOS zapisuje do rejestru segmentu kodu jednostki centralnej odpowiedni adres segmentu. Jako część procesu ładowania odbywa się ustalenie liczby obszarów roboczych do wykorzystania przez DOS. Ich położenia są zmienne i zależą od pewnych czynników będących poza kontrolą programu. Aby można było korzystać z niektórych funkcji DOS-a, programy muszą mieć dostęp do tych obszarów roboczych. Jeśli wykorzystuje się obsługę plików i skorowidza, to w szczególności jest potrzebny obszar DTA (ang. data transfer area).

DTA jest miejscem, do którego funkcja przeszukiwania skorowidza wpisuje informacje o znalezionych plikach. DOS ustala domyślny adres DTA w prefiksie segmentu programu, inny obszar jest ustalany podczas inicjalizacji programu. Podczas wykonywania programy mogą również ustalać i zmieniać własne adresy DTA. DOS posiada funkcję ustalania bieżącego adresu DTA, lecz z punktu widzenia niniejszego artykułu ważniejsza jest funkcja udostępniająca adres aktywnego DTA.

Wywołanie funkcji DOS-a

Obydwie wspomniane metody wywoływania funkcji DOS-a wymagają zapisania w rejestrze **AH** numeru wywołanej funkcji. Niektóre funkcje wymagają zapisania dodatkowych wartości w innych rejestrach. Po wykonaniu wywołania, natomiast, funkcja bardzo często udostępnia różne informacje w tych rejestrach. Na przykład, po wystąpieniu błędu większość funkcji ustawia bit flagowy przeniesienia w rejestrze stanu, a w rejestrze **AX** podaje kod błędu.

Wywołanie funkcji DOS-a odbywa się w trzech etapach:

1. Przygotowanie wywołania polegające na zapisaniu w rejestrze **AH** numeru funkcji i zapisaniu we właściwych rejestrach innych danych.

2. Wykonanie wywołania.

3. Sprawdzenie bitu flagowego przeniesienia (jeśli jest ustawiony, to wystąpił błąd i potrzebna jest obsługa; w przeciwnym razie, błąd nie wystąpił i trzeba sprawdzić przekazane informacje zawarte w odpowiednich rejestrach).

W Turbo Pascalu funkcje DOS-a wywołuje się za pomocą pierwotnie zdefiniowanej procedury **MsDos**, której parametrem jest rekord **Registers**. Rekord składa się z pól typu całkowitoliczbowego odpowiadających większości rejestrów jednostki centralnej. Przygotowując wywołanie, do odpowiednich pól przypisuje się odpowiednie wartości. Każdy rejestr jest 16-bitowy. Ponieważ rejestr **AH** jest bardziej znaczącym bitem rejestru **AX**, przypisując mu numer funkcji trzeba tę wartość pomnożyć przez 100H (jest to notacja szesnastkowa, w programach używa się notacji Pascala – $100) i wynik wpisać do odpowiedniego pola

rekordu **Registers**. Po zakończeniu działania funkcji DOS-a, odpowiednie pola rekordu zawierają wartości przekazane do rejestrów jednostki centralnej.

Kilka funkcji DOS, łącznie z funkcją początkowego przeszukiwania skorowidza, potrzebuje informacji nie przekazywanych przez rejestry jednostki centralnej. Takie informacje podaje się w napisie **ASCIIZ**, tj. w ciągu znaków **ASCII** nie posiadającym początkowego bajtu określającego długość napisu, lecz zakończonym bajtem dwójkowych zer tj. znakiem **NUL** – o kodzie **ASCII – 00H**. Podczas przygotowania wywołania, do wskazania napisu wykorzystuje się parę rejestrów zawierających adres segmentu i wyrównanie.

Dobrym przykładem implementacji w Turbo Pascalu wywołania funkcji DOS-a jest użycie funkcji **4EH** (procedura **FindFirst**) i **4FH** (procedura **FindNext**). Funkcja **4EH** określa położenie pierwszego pliku spełniającego specyfikację wejściową. Funkcja **4FH** znajduje następne pliki. Ponieważ funkcje przeszukiwania skorowidza udostępniają informacje o znalezionych plikach przez obszar **DTA**, to trzeba jeszcze użyć funkcji **GetDTA** do określenia położenia bieżącego obszaru **DTA**.

Program przykładowy

Program przykładowy nazywa się **DirectoryDemo**. Zawiera funkcję **SizeOfFile** i trzy procedury **PoitDTA, FindFirst** i **FindNext**. Jako dane podaje się obowiązującą w DOS-ie specyfikację pliku, mogącą zawierać nazwę napędu, ścieżki i nazwy plików. W specyfikacji można też użyć zmiennej nazwy zawierającej znaki „*" i „?". W wyniku działania programu, w trzech kolumnach na ekranie pojawia się spis nazw plików i skorowidzów spełniających specyfikację wejściową. Nazwy skorowidzów są wyświetlane z mniejszą jaskrawością, a wielkość pliku w tym wypadku jest równa zero. Spis kończy się podaniem liczby znalezionych plików i skorowidzów.

Program **DirectoryDemo** kompiluje się bezbłędnie dla wersji 2 i 3 Turbo Pascala. W tej wersji pracował poprawnie pod kontrolą systemu PC-DOS 2.0 i 2.10, jak również na komputerze Compaq pod kontrolą MS-DOS 2.11. Pod kontrolą PC-DOS 3.0 program będzie pracował w pewnych warunkach niepoprawnie, ponieważ nie uwzględniono w nim, że DOS 3.0 podaje dodatkowe kody błędów. W programie wykorzystano kilka nowych funkcji DOS-a i dlatego nie będzie on działał pod nadzorem systemu DOS w wersji 1.0 i 1.1.

Typy danych, stałe i zmienne

W programie **DirectoryDemo** bardzo ważne są trzy zdefiniowane typy danych (wydruk 1) i dwie procedury wykonujące przeszukwanie skorowidza. Typ **Registers** określa rekordowy typ danych zawierający pola reprezetujące rejestry jednostki centralnej. Każdorazowe wywołanie funkcji DOS powoduje przesłanie zmiennej tego typu do zdefiniowanej pierwotnie procedury **MsDos**. Pola rekordu zawierają wartości, które mają być zapisane w rejestrach jednostki centralnej. Po powrocie z wywołania te same pola zawierają wartości udostępniane przez funkcję DOS-a.

```
program DirectoryDemo;
type
  UserSpec     = string[64];
  Registers    = record
                   AX,BX,CX,DX,BP,SI,DI,DS,ES,Flags : integer
                 end;
  FileName     = string[13];
  DTAPointer   = ^DTARecord;
  DTARecord    = record
                   DOSReserved  : array[1..21] of byte;
                   Attribute    : byte;
                   FileTime,    (postać specjalna  )
                   FileDate,    (  "        "      )
                   SizeLow,
                   SizeHigh     : integer;
                   FoundName    : array[1..13] of char;
                 end;

const
  NUL    = ^@;
  SeekAttrib = $10;  (pliki i podskorowidze)

var
  TransferRec  : DTAPointer;
  MatchPtrn    : UserSpec;
  RetName      : FileName;
  FilSize      : real;
  Count        : integer;
  Nofind, LastFile,
    SubDirec   : boolean;
```

Wydruk 1

Pozostałe dwa typy danych – **DTARecord** i **DTAPointer** – są powiązane wzajemnie:

- **DTARecord** jest rekordem, którego pola odpowiadają rozmieszczeniu danych udostępnianych przez DOS w odpowiedniej postaci o każdym pliku znalezionym w DTA.
- **DTAPointer** określa wskaźnik rekordu typu **DTARecord**. Gdy wskaźnik wskazuje na DTA, to można bezpośrednio z pól **DTARecord** uzyskiwać informacje podawane przez funkcje przeszukiwania skorowidza.

Dla wygody zdefiniowano dwie stałe globalne. Stała **NUL** ma wartość znaku **ASCII 00H** i służy jako ogranicznik napisu **ASCIIZ**, przekazywanego do funkcji rozpoczynającej przeszukiwanie skorowidza. Inna stała, o nazwie **SeekAttrib**, ma wartość **10H** i jest używana podczas przeszukiwania skorowidza do określania atrybutów szukanych plików.

Trzy zmienne logiczne, **NoFind, LastFile** i **SubDirec**, po każdym wywołaniu informują o stanie przeszukiwania i udostępnieniu nazwy skorowidza, co ma wpływ na sposób jej wyświetlania.

Odczytanie adresu DTA

Pierwszą procedurę programu **PointDTA** (wydruk 2) wykorzystuje się do zlokalizowania aktualnego adresu DTA i ustawienia wskaźnika **TransferRec** wskazującego na ten adres. W celu odczytania adresu DTA wywołuje się funkcję DOS o numerze **2FH**. W programie przestrzegano zasady, aby procedury lub funkcje wywoływały tylko jedną funkcję DOS-a. Dodatkowo, funkcje DOS-a zadeklarowano jako stałe mnożone przez **100H** aby można je zapisywać do rejestru **AH** przez proste przypisanie wartości do rejestru **AX**. Adres DTA udostępniany przez funkcję **2FH** jest zapisany w rejestrach **ES** (adres segmentu) i **BX** (wyrównanie). Wartość wskaźnika ustawia się za pomocą zdefiniowanej funkcji **Ptr**.

```
procedure PointDTA(var DTARec : DTAPointer);
const
  GetDTA = $2F00;    (numer funkcji)

var
  Regs : Registers;

begin
  Regs.AX := GetDTA;
  MsDos(Regs);
  DTARec := Ptr(Regs.ES, Regs.BX);
end;
```

Wydruk 2

Obliczanie wielkości pliku

Funkcja **SizeOfFile** (wydruk 3) oblicza wielkość każdego znalezionego pliku. Ponieważ pliki w systemie DOS mogą przekroczyć wielkość 32767 bajtów (**MaxInt** w Turbo Pascalu), to funkcja udostępnia wartość typu rzeczywistego. Wielkość każdego pliku jest zapisana w parze słów, **HiWord** i **LoWord**, przekazywanych jako paramentry do funkcji **SizeOfFile**. Stwarza to pewien kłopot, ponieważ wartość każdego słowa jest wyrażona jako całkowita liczba 16-bitowa bez znaku, a więc w postaci nieznanej w Turbo Pascalu. Dlatego wartości większe od **MaxInt** traktowane jako całkowite ze znakiem wydają się być ujemne.

Problem rozwiązano za pomocą zmiennej rzeczywistej **BigNo** zawierającej wartość 2^{16}, od której odejmuje się wartość ujemną w celu znalezienia prawdziwej wartości słowa. Wielkość pliku jest sumą wartości słowa bardziej znaczącego, pomnożonego przez **BigNo**, i słowa mniej znaczącego.

```
function SizeOfFile(HiWord, LoWord : integer) : real;

var
  BigNo, Size : real;

begin
  BigNo := (MaxInt * 2.0) + 2;
  if HiWord < 0 then Size := (BigNo + HiWord) * BigNo
    else Size := HiWord * BigNo;
  if LoWord >= 0 then Size := Size + LoWord
    else Size := Size + (BigNo + LoWord);
  SizeofFile := Size;
end;
```

Wydruk 3

Znajdowanie pierwszego pliku

Procedura **FindFirst** (wydruk 4) wykonuje przeszukiwanie aż do pierwszego wystąpienia nazwy pliku spełniającej specyfikację wejściową. Najpierw następuje wywołanie funkcji **4EH** rozpoczynającej przeszukiwanie i przygotowuje się kolejne wywołania funkcji **4EH** podającej nazwy następnych plików. Procedura **FindFirst** zapisuje odpowiednie rejestry, wykonuje wywołanie i sprawdza błędy. Po stwierdzeniu zgodności nazw podaje pełną nazwę pliku, jego wielkość i znacznik sygnalizujący czy plik jest skorowidzem. Jeśli przeszukiwanie jest nieudane, to **FindFirst** sygnalizuje ten fakt przez parę znaczników.

Funkcja **4EH** wymaga podania napisu **ASCIIZ** zawierającego porównywany wzorzec. Napis jest zapisywany do zmiennej lokalnej **FileSpec**. Za pomocą zdefiniowanych pierwotnie funkcji **Seg** i **Ofs** zapisuje się adresy segmentu i wyrównania zmiennej **FileSpec** do pól **DS** i **DX** zmiennej rekordowej **Regs**.

Atrybut poszukiwanych plików jest dodatkowym parametrem funkcji **4EH**. Przed wykonaniem wywołania zapisuje się go do rejestru **CX**. Ponieważ w tym wypadku należy zlokalizować wszystkie pliki i podskorowidze spełniające specyfikację, do **CX** zapisuje się stałą **SeekAttrib**. Po wywołaniu funkcji **4EH** procedura **FindFirst** bada bit flagowy przeniesienia w polu **Flags** zmiennej **Regs**, aby sprawdzić, czy funkcja nie udostępnia kodu błędu.

Kod błędu informuje o stanie przeszukiwania, jeśli przeszukiwanie jest bezskuteczne. DOS począwszy od wersji 2.0 podaje dwa możliwe stany: błąd 2 wskazuje, że nie znaleziono żadnego pliku odpowiadającego specyfikacji, a błąd 18 wskazuje, że znaleziono ostatni plik spełniający specyfikację wejściową. Procedura **FindFirst** została tak zaprojektowana, że dowolny błąd różny od 2 lub 18 powoduje zatrzymanie programu i wyświetlenie komunikatu „Niezrozumiały kod błędu". W poważniejszych programach można przewidzieć bardziej elegancki sposób obsługi tej sytuacji. Procedura **FindNext** działa w ten sam sposób.

Jeśli bit flagowy przeniesienia nie jest ustawiony, to nie trzeba sprawdzać kodu błędu, gdyż znaleziono plik, którego nazwa spełnia specyfikację wejściową. W procedurze **FindFirst** odbywa się przekazanie informacji o pliku z rekordu DTA do zmiennych widocznych w programie głównym. W tym wypadku podawana informacja o pliku składa się z trzech części: nazwy i rozszerzenia, atrybutu pliku i jego wielkości.

Funkcja **4EH** udostępnia nazwę i rozszerzenie pliku rozdzielone kropką w jednym polu zakończonym znakiem **NUL**. Jeśli nazwa i rozszerzenie nie wypełniają całego pola, to w końcowej części pola mogą pojawić się przypadkowe znaki. Jeśli znalezionym plikiem jest skorowidz, to zmienna logiczna **SubDir** ma ustawioną wartość **True**, a wielkość pliku jest równa zero. W przeciwnym razie wielkość pliku jest obliczana przez funkcję **SizeOfFile**. Z obszaru DTA można odczytać również datę i godzinę ostatniego uaktualnienia pliku; dla uproszczenia programu dekodowanie tych pól pozostawiono Czytelnikom. Zarówno data, jak i czas dzienny są zapisywane w specjalnej, upakowanej postaci.

Znajdowanie dodatkowych plików

Procedura **FindNext** (wydruk 5) poszukuje dodatkowych plików spełniających specyfikację ustaloną przez **FindFirst**.

```
procedure FindFirst(Pattern : UserSpec;
  var Found : FileName; var Size : real;
  var NoMatch : boolean; var LastOne : boolean;
  var SubDir : boolean);

const   FindFirst = $4E00;  {numer funkcji}
type
  ASCIIZ   = array[1..64] of char;

var
  FileSpec : ASCIIZ;
  Regs     : Registers;
  PosInStr,
  Count    : integer;
  FoundLen : byte absolute Found;

begin
  for PosInStr := 1 to length(Pattern) do
    FileSpec[PosInStr] := Pattern[PosInStr];
  FileSpec[length(Pattern) + 1] := NUL;
  with Regs do
  begin
    DS := Seg(FileSpec);
    DX := Ofs(FileSpec);
    CX := SeekAttrib;
    AX := FindFirst;
    MsDos(Regs);
    if (Flags and 1) > 0 then
    begin
      case AX of
        2: begin    {Nie ma zgodności}
             NoMatch := true;
             LastOne := true;
           end;
       18: begin    {Nie ma innych plików}
             NoMatch := false;
             LastOne := true;
           end;
      else
        writeln(^G'Niezrozumiały kod błędu');
        Halt;
      end;  {instrukcji case}
    end
    else
      begin         {Nie ma kodu błędu}
        NoMatch := false;
        LastOne := false;
      end;          {instrukcji wiazacej with}
  end;
  if (not NoMatch) then


  with Transferrec ^ do
  begin
    Found := FoundName;
    Count := 0;
    while Found[Count] <> NUL do Count := Count + 1;
    FoundLen := Count;
    for Count := length(Found) + 1 to 13
      do Found := Found + ' ';
    if (Attribute and SeekAttrib) > 0
      then SubDir := true
      else SubDir := false;
    if not SubDir
      then Size := SizeOfFile(SizeHigh, SizeLow)
      else Size := 0.0;
  end;        {instrukcji wiążącej with TransferRec}
end;
```

Wydruk 4

Działanie tej procedury polega na zapisaniu w polu **AX** rekordu **Regs** numeru funkcji **4FH** i wykonaniu wywołania. Tak jak w procedurze **FindFirst** sprawdza się bit flagowy przeniesienia. Funkcja **4FH** podaje tylko jeden kod błędu – 18, gdy znaleziono ostatni plik spełniający specyfikację. Jeśli nie ma błędu, to proces odczytywania i przekazywania informacji z DTA jest taki sam jak w procedurze **FindFirst**.

Program główny

Program główny (wydruk 6) składa się z trzech podstawowych części: przygotowawczej, pętli służącej do poszukiwania plików spełniających specyfikację wejściową, i końcowej. W czę-

ści przygotowawczej podaje się specyfikację pliku, procedura **PointDTA** ustawia wskaźnik na DTA i wywołuje się procedurę **FindFirst** sprawdzającą pierwszą zgodność nazw. Ponieważ jest to tylko program przykładowy, nie sprawdza się prawidłowości specyfikacji podanej przez użytkownika.

```
procedure FindNext(var Found : FileName; var Size : real;
                   var LastOne : boolean; var SubDir : boolean);

const   FindNext = $4F00;      {numer funkcji}
var
  Regs      : Registers;
  Count     : integer;
  FoundLen  : byte absolute Found;

begin
  with Regs do
  begin
    AX := FindNext;
    MsDos(Regs);
    if (Flags and 1) > 0 then
      if AX = 18 then Lastone := True
     else
       begin
         writeln('G'Niezrozumiały kod błędu');
         Halt;
       end
    else LastOne := false;
  end;          {instrukcji wiążącej with Regs}

  with TransferRec^ do
  begin
    Found := FoundName;
    Count := 0;
    while Found[Count] <> NUL do Count := Count + 1;
    FoundLen := Count;
    for Count := length(Found) + 1 to 13
      do Found := Found + ' ';
    if (Attribute and SeekAttrib) > 0 then SubDir := true
      else SubDir := false;
    if not SubDir then
      Size := SizeOFFile(SizeHigh, SizeLow)
    else Size := 0.0;
  end;     {instrukcji wiążącej with TransferRec }
end;
```

Wydruk 5

Należy pamiętać, że przy wywołaniu funkcji DOS-a nazwy ścieżek dostępu można kończyć ukośnikiem lub znakiem dzielenia.

```
BEGIN
  ClrScr;
  writeln(' --- Demonstracja przeszukiwania skorowidza ---');
  write('  Co znalezc?');
  readln(MatchPtrn);
  writeln;
  Count := 0;
  PointDTA(TransferRec);
  FindFirst(MatchPtrn, RetName, FilSize, NoFind, LastFile,
    SubDirec);
  if NoFind or LastFile then writeln('Nie znaleziono pliku.')
    else
    begin
      while (not LastFile) do
      begin
        if SubDirec then LowVideo;
        write(RetName,' ',FilSize:8:0,'    ');
        NormVideo;
        Count := Count + 1;
        if (Count mod 3) = 0 then writeln;
        FindNext(RetName, FilSize, LastFile, SubDirec);
      end;
    end;

  if (Count mod 3) <> 0 then writeln;
  writeln;
  write('*** znaleziono ');
  write(Count,' Pliki lub ');
  LowVideo;
  writeln('Podskorowidze ***');
  NormVideo;
END.
```

Wydruk 6

Jeśli procedura **FindFirst** znajdzie pierwszy plik, to program główny wchodzi w pętlę wyświetlającą informacje o znalezionych plikach i wywołującą procedurę **FindNext**, dopóki są znajdowane pliki spełniające specyfikację wejściową. Skorowidze są podświetlane z osłabioną, a wszystkie inne pliki z normalną jaskrawością. Po znalezieniu ostatniego pliku spełniającego specyfikację program podaje komunikat końcowy i kończy swoje działanie.

* * *

Program **DirectoryDemo** służy dwom celom. Ilustruje, jak w programach pisanych w Turbo Pascalu można wykorzystać funkcje systemu DOS, i obajśnia, jak można wywołać procedury przeszukiwania skorowidza.

Sposoby przedstawione w artykule można wykorzystać do wywołania procedur DOS-a i BIOS-a przez przerwania, używając w tym celu zdefiniowanej pierwotnie procedury **Intr**. W tym wypadku, oprócz zapisania odpowiednich pól w rekordzie **Registers**, trzeba podać w programie numer odpowiedniego przerwania.

Za pomocą procedur **MsDos**, **Intr** i podanych „wytrychów" do systemu operacyjnego zawartych w językach wysokiego poziomu można uzyskać pełny dostęp do usług wykonywanych przez funkcje i przerwania DOS-a. Najważniejsze, że nie trzeba przy tym znać języka asemblera, a programy mogą wykonywać prawie każde zadanie wykonalne przez sprzęt. Ceną, jaką płaci się za te udogodnienia, jest nieprzenośność tak powstałych programów do innych systemów operacyjnych (bez znacznych modyfikacji).

Opracował: MARIUSZ KUC
na podst. BYTE, December 1986

Cztery kompilatory Ady na mikrokomputery PC (1)

Ada pretenduje do miana uniwersalnego języka programowania – nie tylko z powodu udostępnienia funkcji charakteryzujących nowoczesne języki, takich jak struktura blokowa, ścisła typizacja, rekordowe i wskaźnikowe struktury danych, ale również ze względu na właściwości języków zaawansowanych, tj. wielozadaniowość, obudowane typy danych i obsługa wyjątków. Te cechy czynią Adę bardzo atrakcyjną dla programistów użytkowych, ale równocześnie utrudniają implementowanie i rozumienie skomplikowanych kompilatorów.

Mimo, że obecnie istnieje już siedem kompilatorów Ady na mikrokomputery, w artyule omówiono tylko cztery: **Alsys Ada** wersja 1.2 (obecnie jest już dostępna wersja 3.0), **Artek Ada** wersja 1.25, **AdaVantage** wersja 1.0 firmy Meridian Software Systems (istnieje już wersja 2.0) i **Janus/Ada** wersja 1.61 firmy RR Software. Kompilator Alsys Ada jest jedynym spośród wymienionych posiadającym atest Departamentu Obrony USA. Pozostałe ewoluują w stronę normy ANSI, ponieważ nie sankcjonuje się implementacji częściowych.

Trzy kompilatory wymagają 512 KB pamięci RAM. Artek Ada potrzebuje 384 KB. Alsys Ada wymaga mikrokomputera IBM PC/AT lub zgodnego z nim, pracującego pod kontrolą systemu PC-DOS lub MS-DOS wersji 3.0 lub z numerem wyższym, pozostałe trzy pracują na mikrokomputerach IBM PC/XT/AT z systemem operacyjnym o numerze nie niższym niż 2.0. Kompilator Alsys Ada jest sprzedawany z płytą pamięci o pojemności 4 MB, ale dodatkowo wymaga podstawowych 512 KB pamięci mikrokomputera PC/AT. Ponadto, Alsys Ada wymaga 6 MB przestrzeni na dyku stałym, Janus/Ada i Artek Ada po 2 MB, a AdaVantage 1 MB. Kompilatory firm Alsys i Meridian Software Systems potrzebują koprocesora 8087 lub 80287 do wykonywania operacji zmiennoprzecinkowych.

Zaimplementowane właściwości Ady

Ada jest językiem o strukturze blokowej z możliwością zagnieżdżania procedur. Ma struktury sterujące i typy wyliczeniowe podobne do zaimplementowanych w Pascalu. Kontrola typów jest nawet silniejsza, gdyż oddzielnie zdefiniowane dwa typy całkowitoliczbowe nie są zgodne. Ada przerasta Pascala pod względem właściwości prowadzących do tworzenia elastyczniejszych i łatwiejszych w pielęgnowaniu programów. Służą do tego zadania, pakiety, typy rodzajowe i wyjątki. Wielozadaniowość polega na jednoczesnym wykonywaniu więcej niż jednego procesu. Pakiety powodują, że dane i procedury dostępu do tych danych są obudowane. Ułatwia to pielęgnowanie oprogramowania, ponieważ można zmieniać implementację procedury bez zmiany jej sprzężenia. Typy rodzajowe są szablonami dla podprogramów lub pakietów, pomocnymi przy pisaniu programów różniących się między sobą w ściśle określony sposób. Przykładem podprogramu rodzajowego jest szablon procedury **Sort**, z którego można konkretyzować procedury **Sort** dla różnych typów danych. Wyjątki stanowią mechanizm pozwalający na podjęcie działania w wypadku wystąpienia błędu. Programista może tworzyć procedury obsługi wyjątków, które dynamicznie określają konieczne działanie.

W tabeli zestawiono niektóre ważniejsze właściwości Ady zaimplementowane w omawianych kompilatorach. Wszystkie kompilatory udostępniają operacje we-wy sekwencyjne lub bezpośrednie na plikach tekstowych, znakach, napisach, liczbach całkowitych i rzeczywistych. We wszystkich kompilatorach – z wyjątkiem Janus Ady – dopuszczono operacje we-wy na typach wyliczeniowych.

Alsys Ada

System Alsys Ada jest dostarczany na sześciu dyskietkach o wysokiej gęstości. W cenę 2995 dolarów jest wliczona płyta pamięci 4 MB. Kompilator firmy Alsys nie działa bez tej płyty lub z jakąś inną. W odróżnieniu od innych płyt pamięci z wstawianymi mikroukładami, płyta Alsys ma je wlutowane. Jeśli jeden z mikroukładów zostanie uszkodzony, to jego wymiana może być kłopotliwa.

Alsys Ada ma środowisko programowe nadbudowane na systemie operacyjnym DOS. Użytkownik musi znać wszystkie dostępne polecenia, ponieważ na ekranie nie pojawia się żadne menu. Kompilator ma polecenie przywołania samouczka, z którego korzysta się interakcyjnie. Z dowolnego poziomu można dowiedzieć się, jakie są domyślne parametry poszczególnych poleceń, lub wywołać tzw. skrypty (podobne do plików wsadowych systemu DOS) lub polecenia systemowe, takie jak wykonanie skompilowanego programu bez opuszczenia środowiska Ady.

Polecenia Alsys Ada są podobne do wywołań procedur z parametrami skojarzonymi przez pozycję, nazwę lub ich dowolną kombinację. Kompilator może wykonywać opcjonalnie: tylko analizę składniową, analizę składniową i semantyczną, lub pełną kompilację łącznie z generowaniem kodu. Można również ustawić parametry wpływające na wydruk skompilowanych jednostek, wyświetlanie komunikatów ostrzegawczych i wydruk postaci źródłowej programu.

Drugi krok przy tworzeniu plików wykonywalnych polega na wykonaniu konsolidacji za pomocą standardowego programu LINK. Programista ma możliwość pisania programów pracujących w trybie rzeczywistym, ochronnym lub rozszerzonym. W trybie rzeczywistym wielkość programów nie może przekraczać 640 KB. W programach wykonywanych w ochronnym trybie adresowym można alokować zmienne dynamiczne na obszarze do 16 MB, ale w dalszym ciągu wielkość programu nie może przekraczać 640 KB. Barierę 640 KB przekracza się wykorzystując program wiążący (ang. binder) do tworzenia rozszerzonych plików programowych używających dysku elektronicznego. Programy pracujące w trybie rozszerzonym są wolniejsze od programów trybu ochronnego.

Zarządca bibliotek pozwala utworzyć nową bibliotekę oraz kopiować, przemianowywać, usuwać lub modyfikować już istniejące. Alsys Ada ma również zarządcę jednostek, który służy do manipulowania indywidualnymi jednostkami w pojedynczej bibliotece programu.

System firmy Alsys udostępnia także wiele procedur sprzężenia z systemem operacyjnym PC-DOS i sprzętem. Wykorzystuje standardowy program **ANSI.SYS** i wykonuje funkcje, takie jak dodatkowe operacje we-wy, drukowanie, dostęp do rejestracji czasu i daty, sprawdzenie numeru systemu operacyjnego, zarządzanie plikami i plikowe operacje we-wy.

Ceny pakietu firmy Alsys i obowiązek korzystania ze specjalnej płyty pamięci wskazują, że ta pełna implementacja Ady nie jest przeznaczona dla nowicjuszy.

Artek Ada

W skład środowiska programowego Ady, tzw. **APSE**, firmy Artek wchodzi edytor, kompilator, konsolidator, program tłumaczący i uruchomieniowy oraz disasembler. Konsolidator ma szczególną właściwość polegającą na tym, że wytwarza pośrednie pliki **A-kodu** wykonywane przez interpreter. Program tłumaczący przekształca pliki **A-kodu** na samodzielne pliki wykonywalne. Program uruchomieniowy i disasembler wykorzystuje mnemoniki **A-kodu.** Dostępne są również programy uruchomieniowe: źródłowy i post-mortem. Środowisko **APSE** pozwala wykonywać polecenia systemu PS-DOS lub MS-DOS oraz skompilowane programy Ady.

System Artek Ada jest dostarczany na trzech dyskietkach o normalnej gęstości, do których dołączono trzytomowy podręcznik użytkowania. Dwa tomy zawierają normę Ady, a trzeci – informacje o samym produkcie. Dwie trzecie tomu dotyczącego kompilatora stanowi spis mnemoników **A-kodu;** informacje o implementacji są stosunkowo

Zestawienie zaimplementowanych konstrukcji Ady

Rodzaje konstrukcji	Alsys	Artek	AdaVantage	Janus
Typy				
Podtypy	T	T	T	T
Typy pochodne	T	T	T	T
Typy wyliczeniowe	T	T	T	T
Typy stałoprzecinkowe	T	N	T	N
Typy napisowe	T	T	T	T
Rekordy (wyróżniki i pola wariantowe)	T	T	T	Część
Typy wskaźnikowe	T	T	T	Część
Tablice dynamiczne	T	T	T	T
Agregaty dynamiczne	T	T	T	N
Niepełna deklaracja typu	T	T	T	T
Podprogamy				
Parametry domyślne	T	T	T	T
Kojarzenie parametrów	T	T	T	T
Przeciążenie podprogramów	T	T	T	T
Przeciążenie operatorów	T	T	T	T
Pakiety	T	T	T	T
Zadania				
Typ zadaniowy	T	N	T	N
Obiekt zadaniowy	T	N	T	N
Aktywacja zadania	T	N	T	N
Wykonanie zadania	T	T	T	N
Priorytety	T	N	T	N
Atrybut wejścia	T	N	T	N
Jednostki rodzajowe				
Obiekty formalne	T	N	T	N
Typy formalne	T	T	T	N
Podprogramy formalne	T	T	T	T
Wyjątki				
Deklaracje	T	T	T	T
Programy obsługi	T	T	T	T
Zgłaszanie wyjątku (podczas wykonywania zadań i podczas opracowania deklaracji)	T/T	N/T	T/T	N/T
Wykonanie instrukcji	T	T	T	T

skąpe. Ze względu na częste modyfikacje oprogramowania niezbędne jest wydrukowanie – przed użyciem – pliku **README**, zawierającego informacje o dokonanych zmianach.

Artek Ada ma najbardziej przyjazne dla użytkownika środowisko, co czyni go odpowiednim do nauczania Ady. W czasie pracy, na dole ekranu wyświetlane są wszystkie dostępne opcje. Kompilator, oprócz funkcji typowo edycyjnych, udostępnia 10 okien. Edytor ma polecenia charakterystyczne dla Ady, takie jak dodanie lub usunięcie komentarzy, wcięcie lub dosunięcie bloków programu. Można też wywołać specjalne opcje kompilatora, np. wytwarzanie specjalnych plików wynikowych, wykorzystywanych przy uruchomianiu na poziomie źródłowym. Inne opcje obejmują: wydruk postaci źródłowej programu na standardowym urządzeniu wyjściowym (taki sam efekt ma dyrektywa lub pragma **LIST(ON)**), wyłączenie kontroli czasu i zrzut tablic symboli. Konsolidator współpracuje z zarządcą biblioteki przy kontroli wersji. Informacje przechowywane przez zarządcę biblioteki są używane przez program usługowy do automatycznej rekompilacji (ang. automatic recompiling facility), dostępny jako opcja środowiska programowego Ady.

Można też nie wykorzystywać środowiska programowego Ady i pisać program w wybranym edytorze, a później go kompilować, konsolidować i tłumaczyć z poziomu systemu operacyjnego. W tym celu można wykorzystać pliki wsadowe, ponieważ kompilator, konsolidator i program tłumaczący znajdują się na oddzielnych plikach.

Firma Artek pracuje nad pełną implementacją Ady. Obecnie jeszcze nie zaimplementowano wielozadaniowości, chociaż kompilator może sprawdzać składnię niezbyt skomplikowanych zadań. Jednostki rodzajowe zaimplementowano tylko częściowo; obiekty rodzajowe są niedostępne, a podprogramy lub pakiety rodzajowe trzeba kompilować przyd użyciem.

Z drugiej strony, tablice dynamiczne, operatory, operacje we-wy na typach wyliczeniowych i ograniczone jednostki rodzajowe pozwalają pisać w omawianej implementacji całkiem niezłe programy. System Artek Ada zawiera uniwersalne biblioteki wykonujące wiele standardowych zadań, np. działania na macierzach, sortowanie, przeszukiwanie i zarządzanie dynamicznymi strukturami danych. Jeśli chodzi o operacje zmiennoprzecinkowe, to Artek Ada może korzystać z koprocesora 8087 lub 80287 lub wykorzywać emulatory.

Opracował: M. KUC
na podstawie „Byte"

Centrum Szkolenia Informatycznego ZETO w Łodzi informuje o kursach i seminariach zaplanowanych w listopadzie i grudniu 1988 r. Szczegółowe informacje można uzyskać pod adresem: ul. Cz. Hutora 69, 90-558 Łódź, tel.: 36-47-70, 32-98-98, 32-50-70 w. 13.

Komputery Jednolitego Systemu RIAD i komputery ODRA

Programowanie w języku PL/1 pod systemem operacyjnym OS
21 listopada – 2 grudnia, cena 23 800 zł

System operacyjny OS dla operatorów
5-14 grudnia, cena 18 600 zł

Cobol – organizacja zbiorów dyskowych
21-25 listopada, cena 12 600 zł

Użytkowanie i obsługa mini- i mikrokomputerów

System operacyjny MS DOS – podstawy użytkowania
19-22 grudnia, cena 14 800 zł

System operacyjny MS DOS – zasady funkcjonowania
5-9 grudnia, cena 15 800 zł

Podstawy programowania mikrokomputerów
28 listopada – 2 grudnia, cena 14 800 zł

PASCAL TURBO – kurs podstawowy
21 listopada – 2 grudnia, cena 29 600 zł

PASCAL TURBO – rozszerzenie (kurs dla zaawansowanych)
12-16 grudnia, cena 15 800 zł

ASSEMBLER Z-80
14-25 Listopada, cena 27 600 zł

ASSEMBLER 8086
28 listopada – 9 grudnia, cena 29 600 zł

Przegląd oprogramowania narzędziowego mikrokomputerów 16-bitowych
28 listopada – 2 grudnia, cena 15 800 zł

DBASE-3
12-22 grudnia, cena 29 600 zł

SYMPHONY – pakiet zintegrowany
5-16 grudnia, cena 29 600 zł

Podstawy użytkowania mikrokomputera IBM PC/XT
14-18 listopada, cena 14 800 zł

Obsługa operatorska i superoperatorska urządzenia MERA 9150 w systemie MT
14-25 listopada, cena 29 600 zł

Metody obliczeń inżynierskich
12-16 grudnia, cena 14 200 zł

Seminaria i konferencje

VI Seminarium dla projektantów i programistów: „Oprogramowanie narzędziowe i użytkowe mikrokomputerów"
21-24 listopada, cena 24 800 zł

V Szkoła Mikrokomputerowa: „Mikrokomputery 16-bitowe i 32-bitowe; Mikrokomputerowe systemy baz danych"
5-7 grudnia, cena 24 800 zł.

Niektóre pakiety do sterowania dla IBM PC

Od kilku lat coraz więcej komputerów osobistych (PC) pojawia się w halach amerykańskich fabryk i zakładów produkcyjnych. W rezultacie na rynku pojawiła się pewna liczba gotowych pakietów oprogramowania na PC do przemysłowego zbierania danych i sterowania. Wiele z nich stanowi istotnie alternatywę w stosunku do programów pisanych na zlecenie. Poniżej przedstawiono krótko argumenty za i przeciw programom pisanym na zamówienie oraz omówiono cechy, na podstawie których ocenia się oferowane programy gotowe.

Programy pisane na zamówienie

Programy napisane na zlecenie są zwykle szybkie w działaniu i mają dokładnie takie sprzężenia jakich potrzebuje użytkownik. Jednakże im więcej szczegółowych wymagań dostarczy się piszącemu, tym lepiej program będzie spełniać zadanie i tym mniejsze będzie prawdopodobieństwo pojawienia się kłopotów w przyszłości.

Kiedy specyfikacja jest gotowa, dobrze jest uzyskać kilka ofert. Możliwe, że jakaś firma pośrednicząca w sprzedaży oprogramowania lub kompletująca je zaproponuje przy tym sprzęt, do którego jest przyzwyczajona. Jeśli przedstawiona oferta jest nie do przyjęcia, to należy sporządzić również specyfikację sprzętu.

Gdy zakończy się już tworzenie specyfikacji, składanie i ocenę ofert oraz zostanie zawarta umowa, to następuje okres tworzenia programu – liczony często w tygodniach lub miesiącach. W tym czasie można instalować nowy sprzęt, albo uzupełniać już posiadany, aby skrócić następną fazę – uruchamianie. Po zainstalowaniu oprogramowania, można rozpocząć usuwanie błędów, po czym następuje gruntowne testowanie systemu. Ostatecznie otrzymujemy dostosowane do potrzeb oprogramowanie, realizujące sformułowane wcześniej zadania.

Programy gotowe

Z gotowymi programami rzecz ma się trochę inaczej i dużo prościej. Programy te są zaprojektowane do realizacji wielu różnych zadań sterowania i zbierania danych. Mogą ich nie realizować dokładnie tak, jak to sobie wyobraża użytkownik i nie zawsze muszą być najlepszym rozwiązaniem, ale za to są dostępne od razu i zwykle kosztują kilkakrotnie mniej lub najwyżej połowę tego, co program napisany na zamówienie.

Instalowanie oprogramowania gotowego przebiega łatwiej i szybciej, ponieważ usunięto już jego defekty w zastosowaniach, do których jest przeznaczone. Błędy w programie pojawiają się rzadko, chyba że jest on używany do celów, do których nie został zaprojektowany. Na przykład, program zaprojektowany do działania w rafinerii może nie sterować dobrze aparatami przy fabrycznej taśmie montażowej. Funkcje sterowania w rafinerii są natomiast podobne do wymaganych w fabryce szkła lub papieru, dlatego też program mógłby działać równie dobrze w każdej z nich.

Większość pakietów gotowych można połączyć z różnorodnym sprzętem do zbierania danych ze sterowników i podawania parametrów do sterowników programowalnych takich firm, jak: Allen-Bradley, Gould lub General Electric. Pakiety służą równocześnie do sterowania oraz gromadzenia danych przez karty wejścia-wyjścia, dołączone do magistrali IBM PC. Niektóre programy umożliwiają kilkunastu komputerom PC, połączonym w sieć, współużytkowanie zebranych danych przez szybkie łącze do transmisji, w celu wykonywania bardziej złożonych funkcji sterowania.

Kryteria porównawcze

Dobrą metodą do porównania gotowych pakietów oprogramowania jest przyjrzenie się funkcjom, które realizują i znaczeniu tych funkcji w różnych zastosowaniach (tab.). Większość programów jest prosta w obsłudze, ma komunikatywne zestawy poleceń, „przyjazne" bazy danych i nie wymaga znajomości języka programowania. Kryteria służące do porównywania gotowych programów są następujące:

- maksymalna liczba punktów wejścia-wyjścia,
- minimalny okres próbkowania,
- maksymalna liczba pętli PID (nie licząc ograniczeń sprzętowych),
- funkcje sterujące (włącz-wyłącz, PID, regulacja kaskadowa itp.),
- możliwości graficzne sprzężenia operatorskiego,
- gromadzenie danych o procesie,
- format i zgodność baz danych,
- sprzęgi do innych urządzeń,
- możliwość pracy w sieci,
- łatwość poznania i obsługi,
- cena.

Poniżej omówiono każde z tych kryteriów trochę dokładniej.

Maksymalna liczba punktów wejścia-wyjścia.

W pierwszej chwili wydaje się, że w większości pakietów oprogramowania przewidziano współpracę ze zbyt wieloma czujnikami. Jednakże takie pakiety mogą grupować punkty rzeczywiste i sztuczne, tj. wyniki obliczenia wykonanego na podstawie pomiaru rzeczywistego, a nie tylko odczyty z czujnika. Na przykład, punktem rzeczywistym może być kanał mierzący temperaturę wylotową. Punktem sztucznym może być taki, który odejmuje tę temperaturę od temperatury wlotowej. Wynik jest temperaturą różnicową.

Niektóre pakiety obsługują punkty rzeczywiste, inne pracują z punktami sztucznymi, a jeszcze inne posługują się jednymi i drugimi. W pewnych pakietach rodzaj punktów zależy od użytej aparatury.

Minimalny okres próbkowania

Okres próbkowania to czas między kolejnymi odczytami wartości pomiarowych wraz z ich aktualizacją w bazie danych. W większości pakietów, na aktualizację wartości mierzonych jest przeznaczony pewien procent czasu działania. Reszta czasu jest przeznaczona na takie zadania jak kontakt z operatorem i działania sterujące. Pakiety aktualizują dane przeważnie co sekundę (tj. wszystkie wartości są odświeżane raz na sekundę), choć niektóre są uzależnione od sprzętu i mogą próbkować wolniej lub szybciej. Jeśli sprzęt jest szczególnie powolny, to oprogramowanie może nie zwolnić do takiej szybkości. Jeśli natomiast sprzęt jest szybki, to dane nie będą uaktualniane częściej. Im więcej czasu zużywa się na aktualizację danych, tym mniej pozostaje go na funkcje sterowania i komunikacji z operatorem.

Maksymalna liczba pętli regulacji

Mimo, że nie we wszystkich zastosowaniach jest potrzebna funkcja **regulacja**, to większość pakietów może ją realizować. Jednakże liczba pętli regulacji jest sprzętowo i programowo ograniczona. Ze względu na wielką rozmaitość ograniczeń sprzętowych warto rozważyć tylko ograniczenia programowe, którymi pakiety różnią się między sobą.

Liczba obsługiwanych pętli może być różna (od czterech do około tysiąca). Przynajmniej jeden popularny pakiet oprogramowania nie realizuje regulacji i radzi sobie z tym problemem sprzętowo. W zależności od sterownika, ten sam pakiet może również wysyłać polecenia zmiany parametrów regulacji. W innym jeszcze pakiecie regulacja PID jest traktowana jako opcja.

Funkcje sterujące

Możliwość adresowania wystarczającej liczby urządzeń i przypisania wystarczającej liczby punktów do wykonania funkcji sterujących, to ponad połowa sukcesu. Jednakże często wymaga się czegoś więcej niż prostego sterowania włącz-wyłącz. W większości pakietów znajdują się algorytmy sterowania PID. Większość pakietów dopuszcza też zmiany parametrów regulacji dokonywane na bieżąco, bez zakłócania procesu sterowania. Ogólnie, operator może zmieniać nastawy z konsoli. Ponieważ typowy okres próbkowania dla większości pakietów wynosi jedną sekundę, komputerowi pozostaje dość czasu na obliczenia oparte na najnowszych odczytach i spowodowanie zmiany na wyjściu, jeśli jest potrzebna.

Grafika

Mówiąc popularnie, grafika jest czymś, co sprawia, że pakiety programów są atrakcyjne dla użytkowników. Praktycznie we wszystkich pakietach znajdują się plansze graficzne w swobodnym formacie. Jeden z pakietów może wyświetlać na bieżąco przebieg wejś-

Porównanie pakietów przeznaczonych do sterowania

	CAMM	On-Spec Superintendent	Notebook	Loopworks	THE FIX	CIM-PAC
Największa liczba punktów	6000 punktów analogowych lub cyfrowych	128; zawiera system ekspertowy z tysiącem reguł, działający na bieżąco	800 (każdy punkt wejściowy jest kanałem, niektóre kanały mogą być wtórne)	100, 300 i 3000; mogą być punktami rzeczywistymi lub funkcjami, w ilości zależnej od rozmiaru pakietu	3000 bloków (bloki mogą być analogowe, cyfrowe, obliczeń PID, przyrostów funkcji, okresu funkcji, związane z urządzeniami itp.)	1200 (do 250 punktów obliczeniowych i osobno realizowanych funkcji na węzeł)
Minimalny okres próbkowania	1 s	1 s	Do 1300/s bez bieżącej obsługi grafiki lub do 0,1 s w zwykłym trybie	0,3 s	0,25–1 s	1 s
Liczba pętli PID	Programowo nie realizuje obliczeń PID, przesyła nastawy tylko do PLC	128 płyt pomiarowych; sterowanie PID jako funkcja dodatkowa	400 (potrzebne są dwa kanały na pętlę)	33 (najwyżej 4 wyjścia) lub 75	1000 pętli PID wymaga minimum trzech bloków: wejście, algorytm i wyjście	400 na system; w połączeniu z punktami zmiennymi do 100 na węzeł
Rodzaj sterowania	Pewne funckje nadzorcze, tylko przekazywanie parametrów, programy nie wykonują obliczeń	Monitorowanie, sterowanie nadrzędne na bieżąco, analogowe i cyfrowe, możliwość automatyzacji	Sterowanie ciągłe i dyskretne, pełne algorytmy PID sterowania w pętli zamkniętej, sterowanie w otwartej pętli, kaskadowe itd.	Sterowanie ciągłe, PID dla kaskady, pojedynczego kanału, sprzężenia do przodu itd., przetwarzanie zdarzeń	Sterowanie ciągłe, i dyskretne, wsadowe sterowanie sekwencyjne, pełne algorytmy PID dla kaskady, sprzężenie do przodu itd.	Sterowanie ciągłe, pełny algorytm PID dla pętli zamkniętej, przetwarzanie zdarzeń
Grafika	Punktowa lub punktowa o wysokiej rozdzielczości grafika EGA; można użyć punktowo rysującego zestawu o wysokiej rozdzielczości SCION i Aydtin, również grafika HALO	CGA, punktowa grafika EGA	CGA, EGA i Hercules	10 różnych, punktowych, bieżąco aktualizowanych obrazów do trzech wykresów na obraz, rozdzielczość pionowa 240 punktów	Znaki dla kart graficznych CGA i EGA, można przesyłać specjalne znaki punktowe EGA, grafika HALO	Punktowa EGA, grafika HALO o średniej rozdzielczości 600×350 punktów
Typ rysunków	Określany przez użytkownika, rysunki w dowolnej formie, tworzone kursorem lub digitizerem, animacja na bieżąco	Określany przez użytkownika, rysunek poglądowy w dowolnej formie, animacja na bieżąco	Wstępnie definiowany, wykresy i histogramy oraz wskazania mierników cyfrowych, animacja na bieżąco w trybie zwykłym	Tekst mieszany z grafiką w formacie znakowym; histogramy wielkości analogowych i wyliczonych; przełączniki suwakowe do wskazywania wartości cyfrowych i logicznych	Definiowane przez użytkownika, rysowanie w dowolnej postaci w opcji punktowej EGA, specjalny zestaw znaków do drukowania w dowolnej formie, animacja na bieżąco	Definiowane przez użytkownika, rysunki poglądowe w dowolnej formie, animacja na bieżąco
Bazy danych	Format nietypowy, dodatkowy moduł do przekształcania formatu zapisanego pliku na ASCII lub DIF	Format nietypowy, wykorzystując funckje dodatkowe można gromadzić dane na dysku, lub czytać w kodzie ASCII, a także selektywnie przepisywać do plików odczytywanych przez program Lotus 1-2-3	Definiowana przez użytkownika, sformatowane dane są zapamiętane na dysku w kodzie ASCII lub binarnie; pliki mogą być odczytane przez program Lotus, współpraca na bieżąco z arkuszami obliczeniowymi	Stosowana baza danych może być skonfigurowana on-line lub off-line; użytkownik określa konfigurację wypełniając puste miejsca i wybierając z menu	Format nietypowy, dostęp przez moduły języków Basic i C, dodatkowe moduły do bieżącej obsługi arkusza	Format nietypowy, zbiory danych mogą być przekształcane na format DIF, dane można drukować w kodzie ASCII
Sprzęgi szeregowe	30 programów obsługi w USA i 20 w Europie do PLC takich, jak: A-B, GMM, Westinghouse, Honeywell, TSC, TI; karta wspomagająca ARTIC	Dostępnych jest ponad 50 sprzęgów w tym Allen-Bradley, General Electric, Honeywell, Foxboro i Fisher Controls	Można zdefiniować wejście RS-232C; zapewnia również sprzęg do aparatury naukowej	RS-232C, RS-422, Acrosystems, Omega, Data Translation, Intel	Action, Gould, A-B, Foxboro. Analog Devices i inne programy pomocnicze do obsługi innych sterowników	Programy obsługi są dołączane za dodatkową opłatą
Sprzężenia do magistrali	Standardowa magistrala IBM	Data Translation, Digitronics Sixnet, Opto 22, Intel Bitbus	27 wytwórców, takich jak Action, Metrabyte, Data Translation i in.; można zamówić uniwersalny zestaw programów obsługi	Burr-Brown, ICS Computer Products, Analog Devices, Metra Byte, Omega	Computer-Products, Siemens-Allis, Digitronics Sixnet i inne	Action, inne dołączane stosownie do potrzeb
System operacyjny	PC-DOS, wielozadaniowy	Równoległy DOS-ONSPEC, wielozadaniowy i wielodostępny z czterema konsolami, wieloma programami w tle, zgodny z DOS-em	PC-DOS, pewne funkcje działają w tle, system jednozadaniowy	PC-DOS zawiera wbudowane szeregowanie zadań	PC-DOS, dodatkowa powłoka wielozadaniowa działa w DOS-ie i programy użytkowe mogą być wykonywane z niższym priorytetem	PC-DOS, CIM-PAC działa w języku Forth i zastępuje DOS, Forth jest wielozadaniowy
Cena (w dolarach)	2000–7500	895–7000	895 podstawy pakiet, 295 dodatkowo za funkcje czasu rzeczywistego	100 – 995 300 –2995 3000–4995	2495–8595	950

ciowy w postaci plansz w formacie liniowym. Można wybrać kilkanaście plansz, na których będą pokazane aktualne przebiegi wielu parametrów.

Przynajmniej dwa dostępne pakiety zawierają zestawy znaków graficznych, służące do samodzielnego projektowania obrazu. Zestawy znaków są zawarte w kostkach ROM, które umieszcza się w gniazdkach w module adaptera do standardowego monitora kolorowego. Te znaki zastępują niektóre standardowe zestawy znaków obcojęzycznych w komputerach IBM, kształtami przydatnymi do wyświetlania symboli zaworów, pojemników, połączeń instalacji rurowych itp. Kształty mogą zmieniać kolor lub rozmiary, proporcjonalnie do stanu lub wartości związanych z nimi parametrów. Wadą grafiki znakowej jest ograniczenie do wstępnie zdefiniowanego zestawu znaków. Niektóre pakiety mogą współpracować z kartą graficzną EGA. Użytkownik może definiować potrzebne kształty na ekranie o rozdzielczości 640×350 punktów. Typowe elementy są gotowe (np. prostokąt, łuk, okrąg, linia). Rozmiar i położenie elementu określa się klawiszami kursora lub przyciskiem myszki.

Gromadzenie danych

Jedną z podstawowych funkcji większości omawianych pakietów jest zbieranie danych. Niektóre pakiety mogą zapisywać dane w typowym formacie wymiany danych (**DIF**), a więc można opracowywać je programem Lotus 1-2-3 lub przy użyciu innego popularnego pakietu do analizy. Dane zarejestrowane można odtworzyć w formie rysunków z wykresami. Wielkość pliku sięga często pojemności standardowo sformatowanej dyskietki lub 360 KB. W kilku pakietach ilość danych jest ograniczona pojemnością dysku lub ograniczony jest okres przechowywania danych. Jeden z pakietów tworzy pliki jedynie z ostatnich dwudziestu czterech godzin. Dane mogą być odtwarzane w formie graficznej jako wykresy. Inny pakiet może zawierać moduł, który przekształca pliki na format ASCII.

Sprzęgi do urządzeń zewnętrznych

Nie wszystkie dane wprowadzą się do komputera przez karty wejścia-wyjścia połączone z magistralą. W kilku pakietach dane są przesyłane przez sprzęgi szeregowe i bezpośrednio z magistrali. W wielu zastosowaniach dołącza się komputer PC do urządzenia zewnętrznego, takiego jak sterownik programowany (**PLC**), rejestrator danych lub komputer mający własne karty wejścia-wyjścia (ze względu na odległość).

Tworzenie sieci komputerowej

Pakiety dla komputerów PC różnią się pod względem możliwości łączenia w sieci. Standardowy pakiet pracujący w sieci może współpracować z dwustu pięćdziesięcioma pięcioma węzłami, odległymi co najwyżej o 300 metrów (jeśli nie wzmacnia się sygnałów). Cała sieć może mieć około 1200 metrów długości i każdy węzeł może działać niezależnie. W innym pakiecie wykorzystano sieć PC-NET firmy IBM. Jeszcze inne, popularne pakiety współpracują z czujnikami połączonymi szeregowo z PC (**RS-232C**) lub dołączonymi do magistrali BITBUS.

Łatwość obsługi

Jeśli wybrany pakiet ma wszystkie potrzebne funkcje, ale nie można ich łatwo użyć z powodu ubogiej albo zbyt złożonej dokumentacji, to jego możliwości przestają mieć znaczenie. Dokumentacja dołączona do niektórych pakietów jest przejrzysta, a do innych – bardzo zawiła. Na przykład, jeden z pakietów mieści się na pojedynczej dyskietce, do której jest dołączony jeden podręcznik. Inny pakiet można kupić z podręcznikiem Pascala, wprowadzeniem do programowania w Pascalu, spisem wyświetlanych obrazów, spisem terminów, podręcznikiem dla użytkownika bloku sterowania, podręcznikiem obsługi urządzeń zewnętrznych i tzw. „elementarzem". Inne popularne pakiety mieszczą się pomiędzy tymi dwoma skrajnościami.

Opracował
JERZY STRYCHARCZYK

W skrócie ● W skrócie

□ Urządzenie o nazwie Qualiscope, które można zamocować na wózku do zakupów, służy klientowi do bezpośredniego wprowadzanie swej oceny nabywanych produktów, co zostaje następnie wykorzystane do uzyskania wskaźników jakości tego produktu, ogłaszanych regularnie konsumentom. Urządzenie jest wyposażone w system odbiorczy pracujący w zakresie fal podczerwonych, który odbiera sygnały wysyłane przez 32 terminale. Na ekranach terminali są wyświetlane pytania dla klientów (najdłuższe pytanie nie przekracza 160 znaków). Klient wybiera ocenę od 9 (najlepsza) do 1 (najgorsza). Przetwarzanie danych przez Qualiscope zapewnia wiarygodną ekspertyzę jakości. Wykorzystuje te urządzenia czterdzieści punktów sprzedaży firmy Casino.

• • •

□ W połowie 1987 roku nadwyżka eksportu nad importem w dziedzinie przemysłu komputerowego, biurowego i telekomunikacyjnego Stanów Zjednoczonych wyniosła tylko 52,1 mln dolarów, co oznacza spadek o 95,8% w stosunku do analogicznego okresu poprzedniego roku. W styczniu, kwietniu i lipcu import przewyższał eksport. Przemysł komputerowy osiągnął nadwyżkę 2,406 mld dolarów, lecz była ona o 62,6% niższa w stosunku do 1986 roku. W dziedzinie sprzętu biurowego deficyt udało się nieco zmniejszyć (o 9,8%) osiągając 1,232 mld dolarów.

• • •

Firma Vitesse z Camarillo w Kalifornii planuje budowę minisuperkomputera z użyciem układów z arsenku galu, o szybkości obliczeniowej 150 milionów operacji zmiennoprzecinkowych na sekundę, a więc 20 razy większej niż VAX 8600 (przy tym samym koszcie). Prace mają potrwać około dwóch lat.

• • •

Firma Force Computer z Los Gatos w Kalifornii zademonstrowała nowy jednopakietowy komputer CBS pracujący z częstotliwością zegara 25 MHz, bez stanu oczekiwania. Jest on wyposażony w statyczną pamięć półprzewodnikową o pojemności 1 MB i czasie dostępu 25 ns.

W skrócie ● W skrócie

Kto jest kim w IFIP

Herve Gallaire

Herve Gallaire był członkiem Międzynarodowego Komitetu Programowego Kongresu IFIP'86, a obecnie przewodzi Komitetowi Programowemu Kongresu IFIP'89. Ukończył Ecole Nationale Supérieure des Arts et Métiers we Francji, a stopnie magistra i doktora filozofii w zakresie nauk komputerowych otrzymał na Uniwersytecie Kalifornijskim w Berkeley. W 1970 roku został profesorem matematyki i nauk komputerowych Ecole Nationale Supérieure de l'Aéronautique et de l'Espace i pracował na wydziale Nauk Komputerowych Centre d'Etudes et de Recherches de Toulouse (CERT). W 1972 r. dr Gallaire został dyrektorem Wydziału Nauk Komputerowych CERT. Piastował obydwie funkcje do 1980 r., kiedy przeszedł do laboratoriów badawczych firmy Générale d'Electricité, aby stworzyć grupę zajmującą się naukami komputerowymi. W 1984 r. został mianowany dyrektorem zarządzającym Europejskiego Centrum Badawczego Przemysłu Komputerowego w Monachium – wspólnym centrum badawczym firm Bull, ICL i Siemens. Stanowisko to piastuje do dnia dzisiejszego.

Prace dra Gallaire obejmują wiele dyscyplin, od teoretycznych zagadnień komputerowych po sztuczną inteligencję. Są związane z programowaniem logicznym i bazami danych. Opublikował książkę o technikach kompilacji i pracował w komitecie redakcyjnym serii Advances in Databases Theory (Postępy w Teorii Baz Danych). Jest również członkiem wielu francuskich komitetów doradczych w dziedzinie nauk komputerowych.

Oprac. **MK**
na podstawie IFIP Newsletter

Nowe stanowiska komputerowe dla sztucznej inteligencji

Najpopularniejszymi stanowiskami komputerowymi (ang. workstation) przeznaczonymi do tworzenia programów sztucznej inteligencji są obecnie stanowiska przetwarzania symbolicznego firmy Symbolics. Komputerom Symbolics przybyli jednak ostatnio poważni konkurenci, w postaci dwóch nowych stanowisk przetwarzania symbolicznego firmy Texas Instruments: Explorer II oraz Explorer II LX. Obydwa stanowiska są oparte na Lispie i wykorzystują specjalny mikroprocesor lispowy, opracowany przez TI w ramach zamówienia złożonego przez agencję DARPA (Defence Advanced Research Projects Agency). W oparciu o testy Gabriela, stosowane powszechnie do porównywania wydajności systemów lispowych, stwierdzono, że nowe stanowiska działają pięciokrotnie szybciej od poprzednich modeli rodziny Explorer. Stanowisko Explorer II stanowi z punktu widzenia użytkownika środowisko programowania w Lispie, natomiast Explorer II LX jest w rzeczywistości zintegrowanym środowiskiem Lispu i Unixa, łączącym w jednym systemie procesor Explorera II i procesor oparty na mikroprocesorze 68020. Tym samym użytkownik ma do dyspozycji bogatą bibliotekę programów działających pod kontrolą Unixa, jak również może dołączać do tych programów moduły sztucznej inteligencji opracowane na stanowisku Explorer II LX.

Rodzina stanowisk przetwarzania symbolicznego Explorer jest wyposażona w oprogramowanie podstawowe, służące do budowy systemów z bazami wiedzy z przeznaczeniem do nadzorowania procesów proukcyjnych, planowania produkcji, diagnostyki i naprawy sprzętu oraz planowania finansowego. Oprogramowanie to umożliwia również szybkie konstruowanie prototypowych systemów w takich dziedzinach sztucznej inteligencji, jak: przetwarzanie sygnałów, mowa, widzenie i robotyka. Oprogramowanie nowych stanowisk zapewnia – w porównaniu z poprzednimi wersjami – większą wydajność, możliwość komunikacji z sieciami komputerowymi IBM SNA, DECnet, TCP/IP i NFS, jak również dostęp do zmodyfikowanych i nowych programów wspomagających budowę systemów z bazami wiedzy.

Pojemność pamięci centralnej stanowisk Explorer II i Explorer II LX wynosi 8 MB w wersji podstawowej i może być zwiększona do 128 MB. Stanowiska mogą być wyposażone w dyski stałe Winchester o pojemności 140 MB, dyski SMD (ang. storage module drive) 516 MB, pamięci kasetowe 250 KB oraz pamięci taśmowe. Siedemnastocalowy ekran monitora charakteryzuje się dużą rozdzielczością (1024×808 pikseli). Do obsługi stanowisk służy specjalna (ang. low profile) klawiatura oraz mysz.

M. MACHURA

Nowe czasopismo poświęcone systemom z bazami wiedzy

Knowledge-Based Systems

Angielskie wydawnictwo Butterworths, znane z edycji takich pism komputerowych, jak: „Computer-Aided Design", „Computer Communications", „Computer-Integrated Manufacturing Systems", Computer Systems Science and Engineering", „Displays, Image and Vision Computing", „Information Age", „Information and Software Technology", „Microprocessors and Microsystems" oraz „International Journal of Project Management", 1 grudnia 1987 r. opublikowało pierwszy numer kwartalnika „Knowledge-Based Systems". Nowe czasopismo jest poświęcone systemom z bazami wiedzy służącym do usprawniania procesów podejmowania decyzji, uczenia się i innych rodzajów działalności człowieka. Czasopismo koncentruje się przede wszystkim na zastosowaniach systemów z bazami wiedzy, z silnym podkreśleniem aspektu komunikacji człowiek-komputer. Redakcja czasopisma jest zainteresowana wszelkimi zastosowaniami i technikami, które przynoszą praktyczne korzyści użytkownikom oraz wspomagają budowę systemów z bazami wiedzy na etapie określania potrzeb, gromadzenia wiedzy, projektowania, implementowania oraz pielęgnowania oprogramowania.

Pismo „Knowledge-Based Systems" publikuje prace z następujących dziedzin: systemy ekspertowe, zastosowania baz wiedzy, narzędzia budowy systemów, mechanizmy podejmowania decyzji, interakcja z użytkownikiem, zagadnienia organizacyjne, gromadzenie wiedzy, reprezentowanie wiedzy, języki i środowiska programowania, metody budowy baz wiedzy, architektura systemów. Dziedziny zastosowań systemów z bazami wiedzy obejmują obronność, mechanikę, badania kosmiczne, planowanie strategiczne, medycynę, badania geologiczne, nawigację, finansowość, marketing, produkcję, robotykę, doradztwo i prowadzenie szkoleń. Oprócz artykułów technicznych nowe pismo publikuje sprawozdania z koneferencji, przeglądy książek i artykułów, omówienia narodowych programów badawczych, wiadomości z przemysłu i kalendarz wydarzeń związanych z systemami baz wiedzy. Redaktorem naczelnym jest prof. E. A. Edmonds z University of Technology, Loughborough, Wielka Brytania.

Adres redakcji: „Knowledge-Based Systems." Butterworth Scientific Ltd., P. O. Box 63 Guildford, Surrey GU2 5BH, Wielka Brytania.

Z przykrością informujemy Czytelników, że począwszy od numeru lipcowego br. cena INFORMATYKI została ustalona na 250 zł za egzemplarz.

Nowa cena obowiązuje nowych prenumeratorów oraz sprzedaż odręczną. Jednak nie wyklucza się ewentualnej dopłaty różnicy (pomiędzy nową a starą ceną) do już opłaconej prenumeraty na drugie półrocze br. O ewentualnej dopłacie prenumeratorzy zostaną powiadomieni pisemnie przez Dział Kolportażu Wydawnictwa SIGMA. Dotychczasowa cena ulgowa – 50 zł – zostaje utrzymana w prenumeracie do końca 1988 roku.

Jednocześnie informujemy, że od stycznia 1989 roku cena INFORMATYKI będzie wynosić:
300 zł za egzemplarz – cena normalna,
60 zł za egzemplarz – cena ulgowa.

Basic po polsku (2)

Pełny wykaz słów kluczowych Basica podano w tabeli przyjmując kolejność: słowo angielskie, słowo polskie (według wersji TACT POLBASIC), proponowana zmiana w wersji polskiej. O ile głównym kryterium spolszczania nazw funkcji powinno być ich upowszechnienie międzynarodowe (zmieniać czy nie zmieniać), to dobierając nazwy instrukcji należy pamiętać o użyciu (o ile możliwe) trybu rozkazującego, gdy mają one postać poleceń.

Większość instrukcji ma właśnie taką formę, a dobór niektórych nazw nie sprawia trudności, np.:

BEEP – GRAJ,
CLOSE – ZAMKNIJ,
CLS – ZMAŻ,
CONTINUE – KONTYNUUJ,
COPY – KOPIUJ,
ERASE – USUŃ,
FORMAT – SFORMATUJ,
GO TO – SKOCZ DO,
LET – NIECH,
LIST – LISTUJ,
MOVE – PRZESUŃ,
OPEN – OTWÓRZ,
READ – ODCZYTAJ,
RUN – WYKONAJ,
STOP – STÓJ,
VERIFY – SPRAWDŹ.

Rzeczownikową postać poleceń należy jednak zamienić na czasownikową. Rzeczownik mianuje rzecz, w naszym wypadku funkcję lub operator, co omówiono w numerze 4, 1988, natomiast polecenia lub instrukcje – nadające nazwy czynnościom – muszą mieć postać czasownikową. Zgodnie z tym zaleceniem następujące nazwy instrukcji powinny być zmienione na podane przykładowo:

BORDER – OBMALUJ zamiast RAMKA,
CAT – SKATALOGUJ zamiast KATALOG,
RANDOMIZE – LOSUJ zamiast LOSOWANIE.

Choć większość nazw instrukcji ma zalecaną formę rozkazującą, uważam, że niektóre można dobrać lepiej, np. uściślając znaczenie lub dokładniej tłumacząc odpowiedni wyraz angielski.

Pierwszej systuacji dotyczą następujące nazwy:
GOSUB – WYWOŁAJ zamiast SKOCZ POD (w rzeczywistości następuje właśnie wywołanie podprogramu; używając tej formy unika się też zbędnego podobieństwa nazwy do instrukcji SKOCZ DO);
PAUSE – CZEKAJ zamiast PAUZUJ (taka nazwa odpowiada znaczeniu instrukcji);
RESTORE – COFNIJ zamiast PRZYWRÓĆ (nowa nazwa lepiej precyzuje znaczenie instrukcji).

Wydaje mi się też, że nieco lepiej brzmią następujące nazwy:
POKE – WSTAW DO zamiast WRZUĆ DO (ponieważ lepiej brzmi językowo, nie wulgaryzując samej czynności);
RETURN – POWRÓĆ zamiast WRACAJ (chodzi o pojedynczy powrót, a nie ciągłe wracanie).

Dokładniejsze tłumaczenie nazw angielskich dotyczy następujących instrukcji:
CLEAR – KASUJ zamiast CZYŚĆ (tak przyjęto w informatyce tłumaczyć termin **clear**);
INPUT – WPROWADŹ zamiast PRZYJMIJ (angielski termin **input** znaczy wprowadzać i tak należy go tłumaczyć);
LOAD – ŁADUJ zamiast WCZYTAJ (tylko takie tłumaczenie jest poprawne, a ponadto nie jest mylące z polskim odpowiednikiem terminu **read**);
MERGE – SCAL zamiast POŁĄCZ (łączenie znaczy w informatyce co innego, natomiast angielski termin **merging** jest tłumaczony właśnie jako scalanie);
SAVE – ZACHOWAJ zamiast NAGRAJ (bardziej odpowiada angielskiemu terminowi **save** używanemu nie tylko na oznaczenie zarejestrowania na nośniku magnetycznym, lecz – trwałego zapamiętania w ogóle).

Pewien kłopot sprawia przetłumaczenie nazwy instrukcji **PRINT**, ponieważ nie dotyczy ona drukowania w ścisłym sensie. Proponowana polska nazwa **PISZ** nie jest dobra, ponieważ zapisywanie odpowiada angielskiemu terminowi **writing**. Jeśli nie można użyć nazwy **WYŚWIETL** (ma tę samą wadę jako odpowiednik terminu display) lub **DRUKUJ** (chodzi o „drukowanie" na ekranie, a nie na papierze), to może lepiej byłoby przyjąć nazwę **WYPISZ**.

Ciekawą ilustracją trudności związanych z tłumaczeniem nazw instrukcji na język polski są instrukcje graficzne: **CIRCLE, DRAW** i **PLOT**. W POLBASICU każdej z tych instrukcji odpowiadają dwa wyrazy – czasownik z dopełnieniem. Czasownik ten powinien być jednak jednolity, a dopełnienie powinno precyzyjnie określać czynność, tzn.:
CIRCLE – RYSUJ KOŁO,
DRAW – RYSUJ PROSTĄ zamiast RYSUJ LINIĘ,
PLOT – RYSUJ PUNKT zamiast POSTAW PUNKT.
Gdyby komuś bardzo zależało na skróceniu wymienionych nazw do jednowyrazowych, to proponowałby przyjąć odpowiednio: **ZAKREŚL, RYSUJ** i **KROPKUJ**.

Niektórym instrukcjom wejścia-wyjścia towarzyszą dodatkowe parametry, mogące też wystąpić jako samodzielne instrukcje, określające właściwości obrazu. W tych wypadkach nie przyjęto nazw w trybie rozkazującym, uznając prowdopodobnie, że ich działanie jest analogiczne do nadawania wartości określonym zmiennym i wystarczy nadać im nazwy rzeczownikowe z domyślnym wyrazem **USTAW**, tzn.:

Nazwy POLBASICA

Kod dziesiętny	Nazwa angielska	Nazwa polska	Proponowana zmiana
165	RND	LOS	RND
166	INKEY$	KLAWISZ$	–
167	PI	Π	–
168	FN	FN	–
169	POINT	PUNKT	–
170	SCREEN$	EKRAN$	–
171	ATTR	ATRYBUT	–
172	AT	NA	–
173	TAB	W KOLUMNIE	–
174	VAL$	WARTOSC$	–
175	CODE	KOD	–
176	VAL	WARTOSC	–
177	LEN	DŁUGOŚĆ	–
178	SIN	SIN	–
179	COS	COS	–
180	TAN	TG	–
181	ASN	ASIN	ARCSIN
182	ACS	ACOS	ARCCOS
183	ATN	CTG	–
184	LN	LN	–
185	EXP	EXP	–
186	INT	INT	–
187	SQR	√	SQR
188	SGN	ZNAK	SGN
189	ABS	MODUŁ	ABS
190	PEEK	ZAWARTOŚĆ	–
191	IN	WEJŚCIE	–
192	USR	ADRES	–
193	STR$	ŁAŃCUCH$	NAPIS$
194	CHR$	ZNAK$	–
195	NOT	NIE	NOT
196	BIN	DZIESIĘTNE	–
197	OR	LUB	OR
198	AND	I	AND
199	<=	<=	–
200	>=	>=	–
201	<>	<>	–
202	LINE	OD LINI	–
203	THEN	TO WTEDY	TO
204	TO	DO	–
205	STEP	KROK	–
206	DEF FN	DEF FN	–
207	CAT	KATALOG	SKATALOGUJ
208	FORMAT	FORMATUJ	–
209	MOVE	PRZESUŃ	–
210	ERASE	USUŃ	–

INVERSE – NEGATYW,
OVER – NADRUK,
INK – TUSZ,
PAPER – PAPIER.

Przy tej okazji warto zwrócić uwagę, że w dwóch ostatnich wypadkach funkcjonalnie identyczne instrukcje w PTI Logo odnoszą się nie do kolorów tuszu i papieru, lecz do piórka i tła, co jest bardziej logiczne. Nazwy dwóch innych instrukcji z tej grupy są utworzone niepoprawnie; powinny brzmieć:

BRIGHT – JASKRAWOŚĆ a nie JASNOŚĆ (to jest właściwa nazwa odpowiedniego pojęcia fizycznego);
FLASH – MIGANIE, a nie BŁYSK (takie jest znaczenie odpowiedniego pojęcia; por. Informatyka, nr 3, 1985).

Niektóre instrukcje odgrywają rolę deklaracji, dlatego ich nazwy nie muszą występować w trybie rozkazującym:
DATA – DANE,
DIM – TABLICA zamiast WYMIAR (takie jest znaczenie tej deklaracji),
REM – KOMENTARZ zamiast OPIS (ponieważ taki jest termin informatyczny i nie wolno go zmieniać).

Jedna z instrukcji ma poprawną postać równoważnika zdania w postaci rozkazu:
OUT – NA WYJŚCIE.

Proponowałby utworzyć w ten sposób jeszcze jedną nazwę:
NEW – OD NOWA zamiast KASUJ,
ponieważ taka jest właściwa rola tej instrukcji, a wyraz KASUJ służy do oznaczenia czego innego.

Jeżeli instrukcje złożone, takie jak **FOR** i **IF**, muszą mieć nazwy polskie, bo spolszczamy wszystko, to ich elementy składowe są nazwane poprawnie, tzn.:
FOR – DLA,
STEP – KROK,
NEXT – NASTĘPNIE,
IF – JEŚLI,
THEN – TO a nie TO WTEDY, bo to jest postać nadmiarowa.
Zwróćmy uwagę, że budowa instrukcji złożonych jest inna niż prostych, dlatego nie muszą one mieć formy trybu rozkazującego.

Niektóre nazwy, takie jak **STEP** powyżej, **AT, TAB, TO** i **LINE**, pełnią rolę pomocniczą nie tworząc samodzielnych instrukcji i ich nazwy polskie, jak zresztą przyjęto, powinny odpowiadać roli tych elementów w składni języka.

JANUSZ ZALEWSKI

Kod dziesiętny	Nazwa angielska	Nazwa polska	Proponowana zmiana
211	OPEN #	OTWÓRZ #	–
212	CLOSE #	ZAMKNIJ #	–
213	MERGE	POŁĄCZ	SCAL
214	VERIFY	SPRAWDŹ	–
215	BEEP	GRAJ	–
216	CIRCLE	RYSUJ KOŁO	–
217	INK	TUSZ	–
218	PAPER	PAPIER	–
219	FLASH	BŁYSK	MIGANIE
220	BRIGHT	JASNOŚĆ	JASKRAWOŚĆ
221	INVERSE	NEGATYW	–
222	OVER	NADRUK	–
223	OUT	NA WYJŚCIE	–
224	LPRINT	WYDRUKUJ	–
225	LLIST	WYLISTUJ	–
226	STOP	STÓJ	–
227	READ	ODCZYTAJ	–
228	DATA	DANE	–
229	RESTORE	PRZYWRÓĆ	COFNIJ
230	NEW	KASUJ	OD NOWA
231	BRODER	RAMKA	OBMALUJ
232	CONTINUE	KONTYNUUJ	–
233	DIM	WYMIAR	TABLICA
234	REM	OPIS	KOMENTARZ
235	FOR	DLA	–
236	GO TO	SKOCZ DO	–
237	GO SUB	SKOCZ POD	WYWOŁAJ
238	INPUT	PRZYJMIJ	WYPROWADŹ
239	LOAD	WCZYTAJ	ŁADUJ
240	LIST	LISTUJ	–
241	LET	NIECH	–
242	PAUSE	PAUZUJ	CZEKAJ
243	NEXT	NASTĘPNIE	–
244	POKE	WRZUĆ DO	WSTAW DO
245	PRINT	PISZ	WYPISZ
246	PLOT	POSTAW PUNKT	RYSUJ PUNKT
247	RUN	WYKONAJ	–
248	SAVE	NAGRAJ	ZACHOWAJ
249	RANDOMIZE	LOSOWANIE	LOSUJ
250	IF	JEŚLI	–
251	CLS	ZMAŻ	–
252	DRAW	RYSUJ LINIE	RYSUJ PROSTĄ
253	CLEAR	CZYŚĆ	KASUJ
254	RETURN	WRACAJ	POWRÓĆ
255	COPY	KOPIUJ	–

Turski W. M.: Kanoniczny krok procesu programowania (1)

INFORMATYKA 1988, nr 8, s.1

Pierwsza część charakterystyki metody kroków kanonicznych, usprawniającej proces opracowywania i pielęgnacji programów użytkowych

Turski W. M.: Canonical step of programming process (1)

INFORMATYKA 1988, No. 8, p.1

Characteristics of the canonical steps method improving the development and maintenance of application program (part one).

Turski W. M.: Kanonischer Schritt eines Programmierungsprozesses (1)

INFORMATYKA 1988, Nr. 8, S.1

Erster Teil einer Charakteristik von Methode der kanonischen Schritte, die den Prozess der Erarbeitung und Pflege von Anwendungsprogrammen rationalisiert.

Foersterling F.: Wymagania stawiane bazom danych w biurach (1)

INFORMATYKA 1988, nr 8, s.5

Pierwsza część omówienia problemu baz danych w biurach, zawierająca charakterystykę biura i niektórych jego modeli oraz architektury biurowych systemów informacyjnych.

Foersterling F.: Requirements towards office data bases (1)

INFORMATYKA 1988, No. 8, p.5

Discussion of data bases in offices, which includes characteristics of office and some office models, as well as architecture of office information systems (part one).

Foersterling F.: Anforderungen an Datenbanken in Büros (1)

INFORMATYKA 1988, Nr. 8, S.5

Erster Teil einer Besprechung von Problemen, die mit Datenbanken in Büros verbunden sind. Eine Charakteristik von Büro und manchen Büromodellen, sowie von Architektur der Büroinformationssysteme wurde angegeben.

Kaleta G., Owczarczyk J.: Wieloprocesorowa realizacja operacji przetwarzania obrazów

INFORMATYKA 1988, nr 8, s.8

Porównanie efektywności dwóch podstawowych struktur wieloprocesorowych stosowanych do budowy wieloprocesorowego systemu przetwarzania obrazów.

Kaleta G., Owczarczyk J.: Multiprocessor realization of image processing operations

INFORMATYKA 1988, No. 8, p.8

Effectiveness comparison of the two basic multiprocessor structures applied for building multiprocessor image processing system.

Kaleta G., Owczarczyk J.: Mehrprozessorrealisation von Bildverarbeitungsoperationen

INFORMATYKA 1988, Nr. 8, S.8

Eine Vergleichung der Effektivität von zwei grundlegenden Mehrprozessorstrukturen, die zum Bau eines Mehrprozessorbildverarbeitungssystems verwendet werden.

Błaszczak S.: Język Occam (2)

INFORMATYKA 1988, nr 8, s.12

Druga część charakterystyki języka Occam do programowania transputerów, zawierająca opis dalszych mechanizmów tego języka oraz przykładowy program.

Błaszczak S.: Occam language (2)

INFORMATYKA 1988, No. 8, p.12

Characteristics of the Occam language for transputer programming, which contains description of further language mechanisms and an exemplary program (part two).

Błaszczak S.: Occam-Sprache (2)

INFORMATYKA 1988, Nr. 8, S.12

Zweiter Teil einer Charakteristik von der Occam-Sprache für Transputer-Programmierung, der eine Beschreibung der weiteren Mechanismen dieser Sprache und ein Beispielprogramm umfasst.

Turowicz A.: Lokalna sieć komputerowa D-Link (3

INFORMATYKA 1988, nr 8, s.15

Dokończenie charakterystyki sieci D-Link, zawierające alfabetyczny wykaz poleceń usługowych tej sieci.

Turowicz A.: D-Link local area network (3)

INFORMATYKA 1988, No. 8, p.15

Termination of the D-Link network characteristics, which contains alphabetic list of the network service commands.

Turowicz A.: D-Link-Lokalnetz (3)

INFORMATYKA 1988, Nr. 8, S.15

Beendigung einer Charakteristik von dem D-Link-Computernetz, die eine alphabetische Liste von Dienstbefehlen dieses Netzes umfasst.

Bielecki J.: Turbo C. Zmienne ustalone, zmienne nietrwałe, punkty charakterystyczne

INFORMATYKA 1988, nr 8, s.17

Omówienie i przykłady stosowania dodatkowych modyfikatorów języka Turbo C, umożliwiających deklarowanie zmiennych ustalonych oraz zmiennych nietrwałych. Wskazano punkty charakterystyczne programów zawierających takie zmienne.

Bielecki J.: Turbo C. Constant variables, volatile variables, characteristic points

INFORMATYKA 1988, No. 8, p.17

Discussion and examples of applying additional Turbo C language modificators, which enable declaring of constant and volatile variables. Characteristic points of program with such variables are shown.

Bielecki J.: Turbo C. Feste Variablen, nichtfeste Variablen, charakteristische Punkte

INFORMATYKA 1988, Nr. 8, S.17

Besprechung und Beispiele der Anwendung von Turbo C-Sprache zusätzlichen Modifikatoren, die Deklarierung von festen und nichtfesten Variablen ermöglichen. Charakteristische Punkte der Programme mit solchen Variablen werden gezeigt.

Konkurs PTI na prace magisterskie z informatyki rozstrzygnięty!

Zakończył się IV Ogólnopolski Konkurs Polskiego Towarzystwa Informatycznego na najlepsze prace magisterskie z informatyki, zorganizowany przez Dolnośląski Oddział PTI. Na swym ostatnim posiedzeniu w dniu 21 grudnia 1987, które się odbyło w Centrum Obliczeniowym Politechniki Wrocławskiej, jury konkursu w składzie: prof. dr hab. Andrzej Blikle, doc. dr inż. Czesław Daniłowicz (przewodniczący), dr inż. Zbigniew Huzar, doc. dr hab. Maciej Sysło, dr inż. Zbigniew Szpunar (sekretarz), prof. dr hab. Władysław M. Turski i doc. dr hab. inż. Jan Zabrodzki powzięło następujące decyzje:

Pierwszej i drugiej nagrody nie przyznano.

Trzecią nagrodę ex aequo po 25 tys. zł otrzymali:

Krzysztof Jasieński, Jacek Lebiedz za pracę pt. „Uniwersalny makroprocesor tekstu dla potrzeb generacji translatorów MC²" (Politechnika Gdańska, Wydział Elektroniki, Instytut Informatyki, opiekun doc. dr Tadeusz Bartkowski, konsultant mgr inż. Piotr Cofta),

Henryk A. Kowalski za pracę pt. „Rozpoznanie przedwczesnych pobudzeń komorowych w systemach intensywnego nadzoru kardiologicznego" (Politechnika Warszawska, Wydział Elektroniki, Instytut Informatyki, opiekun doc. dr hab. Krzysztof Sapiecha, konsultant mgr inż. A. Banaszek) oraz

Maciej Tadeusz Kukawka za pracę pt. „Optymalne algorytmy sortowania równoległego w modelu siatki procesorów" (Uniwersytet Warszawski, Wydział Matematyki, Informatyki i Mechaniki, Instytut Informatyki, opiekun dr Bogdan S. Chlebus).

Jury konkursu przyznało ponadto trzy **wyróżnienia** po 10 tys. Otrzymali je:

Michał Kirpluk i Piotr Sobolewski za pracę pt. „Projekt i implementacja mechanizmów dedukcji w systemie bazy danych Holmes" (Politechnika Warszawska, Wydział Elektroniki, Instytut Informatyki, opiekun dr Henryk Rybiński),

Grzegorz Mazur za pracę pt. „Szesnastobitowy mikroprocesorowy system laboratoryjny" (Politechnika Warszawska, Wydział Elektroniki, Instytut Informatyki, opiekun dr inż. Cezary Stępień) oraz

Jerzy Stefanowski za pracę pt. „Zastosowanie teorii zbiorów przybliżonych i rozmytych do analizy niepełnych systemów informacyjnych" (Politechnika Poznańska, Wydział Elektryczny, Instytut Automatyki, opiekun doc. dr hab. Roman Słowiński).

Laureatom konkursu serdecznie gratulujemy!

Rzecznik prasowy PTI
BARBARA OSUCHOWSKA

NIE DAJEMY RECEPT
SPRZEDAJEMY NARZĘDZIA

ELEKTRONIKA – FILM – KOMPUTER

ul. Barska 2/20, 02-315 Warszawa

tel. 23-67-57, teleks 816955 efcom pl

OFERUJE:

* bogate profesjonalne oprogramowanie systemowe i narzędziowe umożliwiające tworzenie systemów użytkowych:
  - zarządzanie magazynami,
  - gospodarka materiałowa,
  - planowanie i kalkulacja,
  - lista płac,
  - finanse i księgowość,
  - symulowanie pracy przedsiębiorstwa,
  - przetwarzanie i edycja tekstów,
  - grafika dla celów zarządzania,
  - prowadzenie korespondencji

* wdrażanie systemów, szkolenie personelu oraz dostawy niezbędnego sprzętu mikrokomputerowego wraz z gwarancją i serwisem:
  - mikrokomputery IBM PC/XT/AT kompatybilne,
  - urządzenia peryferyjne,
  - wyposażenie

Sprzedajemy profesjonalne narzędzia
dla profesjonalistów

KONSULTACJE GRATIS

EO 592 88