P. 1877/87

10 1987

informatyka

Prof. Cliff B. Jones o specyfikacjach i programach

Rozproszone bazy danych

WYDAWNICTWO NOT SIGMA

Cena 150 zł

informatyka

10 1987

Nr 10
Miesięcznik Rok XXI
Październik 1987

Organ Komitetu
Naukowo-Technicznego NOT
ds. Informatyki

KOLEGIUM REDAKCYJNE:

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

PRZEWODNICZĄCY
RADY PROGRAMOWEJ:

Prof. dr hab. Juliusz Lech KULIKOWSKI

Materiałów nie zamówionych redakcja nie zwraca

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

Zakł. Graf. „Tamka". Zam. 0673-1300/87. Obj. 4,0 ark. druk. Nakład 8300 egz. K-81

ISSN 0542-9951. INDEKS 36124

Cena egzemplarza 150 zł
Prenumerata roczna 1800 zł

NACZELNA ORGANIZACJA TECHNICZNA
WYDAWNICTWO
SIGMA
CZASOPISM I KSIĄŻEK TECHNICZNYCH

00-950 Warszawa
skrytka pocztowa 1004
ul. Biała 4

W NUMERZE: Strona

W NAJBLIŻSZYCH NUMERACH:

- Kazimierz Bieńkowski o systemach przetwarzania tekstów
- Zdzisław Płoski o właściwościach języka awk
- Mieczysław Kłopotek o maszynowym tłumaczeniu tekstów
- Tadeusz Kuroczycki o mikrokomputerowym procesorze tekstów pod nazwą „Redaktor 2000"
- Zdzisław Płoski o właściwościach i możliwościach zastosowań procesora tekstu Chiwriter
- Władysław Gąsiorek o pewnych problemach konstrukcji i oprogramowania systemowego dla Mery 300

CLIFF B. JONES
Department of Computer Science
University of Manchester
Wielka Brytania

P. 1877/87

Pierwsza wersja artykułu została przedstawiona podczas Szkoły Jesiennej PTI „Współczesne kierunki rozwoju informatyki" — Mrągowo, 3—7 listopada 1986 r.

Specyfikacje a programy (I)

W artykule omówiono niektóre pojęcia związane z językami specyfikacji. Należy przy tym mieć na uwadze istotne rozróżnienie między specyfikacją a implementacją, nawet jeśli notacja użyta do zapisu projektu jest mieszaniną języka implementacji i języka specyfikacji. Rozważania w tym artykule koncentrują się na podejściu VDM (ang. Vienna Development Method), opartym na obowiązku przeprowadzenia dowodów w kolejnych etapach projektowania, tj. podczas ukonkretniania danych i dekomponowania operacji. Podejście takie przeciwstawia się innym, w których język implementacji i język specyfikacji są połączone w jedną całość.

Konstruowanie większych systemów komputerowych wiąże się z wieloma problemami. Niektóre z nich można rozwiązać przy użyciu metod formalnych. Termin „metody formalne" odnosi się tu do użycia notacji matematycznej w specyfikacjach oraz do użycia takich specyfikacji jako podstawy w budowie poprawnych projektów systemów komputerowych. Artykuł ten jest pod wieloma względami rozszerzonym streszczeniem książki [11], która przedstawia metodę VDM w zastosowaniu do konstruowania oprogramowania (metoda VDM jest też stosowana w pracach nad językami programowania [2]).

Nie powinno budzić wątpliwości, że metody formalne mogą wpłynąć na znaczną redukcję liczby błędów w tworzeniu oprogramowania. Problem efektywności procesu produkcyjnego przedstawiono schematycznie na rysunku. Widać na nim, jak gwałtownie wzrasta koszt usunięcia błędu w zależności od czasu, kiedy ten błąd zostanie wykryty. Ponadto, błędy wykryte w dalszych etapach procesu produkcyjnego wynikają na ogół z błędów popełnionych we wczesnej fazie projektowania. Przez dostatecznie wczesne zapewnienie w pełni formalnej kontroli poprawności podjętych decyzji projektowych, zapewnia się jednocześnie większą efektywność całego procesu produkcji oprogramowania [9].

Są trzy mniej lub bardziej rozłączne podejścia do formalnego konstruowania oprogramowania. Każde z nich wychodzi z formalnej specyfikacji żądanej funkcji programu i wykorzystuje formalnie dowodzone przejścia, ktore wiążą program z podaną specyfikacją.

- **Specyfikacja-projekt-weryfikacja.** Specyfikacje pisze się w odmiennym języku specyfikacji, projekt natomiast (i kod programu) — w zwykłym języku implementacji. Poprawność projektu jest zapewniona przez spełnienie określonych obowiązków przeprowadzenia dowodów. Podejście to jest stosowane w metodzie VDM.

- **Transformacja.** „Specyfikacja" jest wykonalną funkcją, w której główny nacisk jest położony na jej przejrzystość (wykonanie funkcji jest na ogół zupełnie nieefektywne). Bardziej efektywną implementację otrzymuje się przez ciąg syntaktycznych przekształceń (z niektórymi z nich wiąże się obowiązek dowodzenia poprawności ich zastosowania). Najbardziej znanym przykładem takiego podejścia jest projekt CIP [5].

- **Konstruktywna matematyka.** W tym podejściu specyfikację uważa się za tezę twierdzenia, a program uzyskuje się z konstruktywnego dowodu tego twierdzenia [6].

W metodzie VDM wykorzystuje się głównie specyfikacje ukierunkowane na modele (ang. model-oriented specifications). W następnym punkcie omówiono na ich tle specyfikacje ukierunkowane na właściwości (ang. property-oriented specifications), a także obszary zastosowania obu tych metod. Operacje specyfikuje się zarówno przez warunki wyjściowe (ang. postcondition) stanów początkowych i końcowych, jak i przez (odmienne) warunki wejściowe tych stanów (ang. precondition). W punkcie pn. „Modele abstrakcyjne" pokazano, w jakim stopniu specyfikacje ukierunkowane na modele mogą być pomocne w opanowaniu architektury systemu. W punkcie „Etapy rozwijania specyfikacji" (cz. 2) przytoczono szerszy przykład ilustrujący różne aspekty podejścia systematycznego do projektowania metodą VDM. Przykłady podano bez formalnych dowodów. Ostatni rozdział przedstawia różnice pomiędzy VDM a innymi metodami.

Cliff Jones zetknął się z techniką komputerową w 1960 roku podejmując pracę, bezpośrednio po szkole średniej, przy komputerach firmy Leo. W ciągu następnych pięciu lat kilkakrotnie zmieniał zajęcie, zdobywając doświadczenie w technice obliczeniowej, badaniach operacyjnych i programowaniu systemowym. W 1965 roku powrócił do pracy w filmie IBM, w Hursley Laboratories. Pracując w laboratorium testowania wyrobów i opracowując narzędzia testujące (jak np. automatyczne generatory testów) przekonał się o bezskuteczności testowania (!).

W 1968 roku został przeniesiony do wiedeńskiego laboratorium IBM, gdzie podjął badania nad opartą na języku VDL (ang. Vienna Development Language) definicją języka PL/I jako podstawą do projektu kompilatora. Choć skuteczne pod względem teoretycznym, opracowanie to zostało niepotrzebnie skomplikowane przez użycie semantyki operacyjnej.

W latach 1970—72 powrócił do Hursley, gdzie próbował wprowadzić nowy styl do semantyki i stosować metody formalne w innych dziedzinach niż tworzenie kompilatorów. Korzystając z możliwości utrwalenia związków z Wiedniem, miastem muzyki i metod formalnych, zrezygnował z dającego dużo satysfakcji kierowania grupą ds. nowoczesnej technologii. W ciągu następnych trzech lat miał udział w „wykuwaniu" metody VDM, powstającej z mieszanki teoretycznych idei i rzeczywistych potrzeb.

Ten okres prawdziwej ekscytacji w jego życiu minął z chwilą wycofania z użycia komputera docelowego dla kompilatora. Możliwość pisania kodu asemblerowego dla narzędzi użytkowych nie mogła zatrzymać dłużej Cliffa w Wiedniu. Następnym przystankiem w jego karierze był Instytut ESRI w Brukseli, gdzie wykładał metody formalne. Po spędzeniu tam trzech lat, napisaniu dwóch książek i późniejszym opublikowaniu kilku artykułów Cliff uznał, że nadszedł czas, aby wypełnić lukę w życiorysie. Przeniósł się do Oxfordu, gdzie pod opieką Tony Hoare'a rozpoczął pracę nad doktoratem na temat metod formalnych w programowaniu równoległym. W 1981 roku objął Katedrę Informatyki (Computing Science) na Uniwersytecie w Manchester. (Zal.)

TYPY DANYCH

Istnieją dwie różne szkoły specyfikacji typów danych. Są to podejścia: **ukierunkowane na właściwości** i **ukierunkowane na modele.** Oba podejścia są stosowane i — odpowiednio użyte — wzajemnie się uzupełniają. Przykładowo, w opisie prostego typu danych, np. skończonych zbiorów liczb naturalnych, sygnatura operatorów może być następująca:

$$\text{empty}: \rightarrow \text{Set}_N$$
$$\text{add}: N \times \text{Set}_N \rightarrow \text{Set}_N$$
$$\text{is-empty}: \text{Set}_N \rightarrow B$$
$$\text{is-memb}: N \times \text{Set}_N \rightarrow B$$

Powyższe operatory umożliwiają utworzenie termów postaci:

$$\text{add}\big(3, \text{add}(5, \text{empty}())\big)$$

lub twierdzeń takich jak:

$$\text{is-memb}\Big(5, \big(\text{add}(3, \text{add}(5, \text{empty}()))\big)\Big)$$

Oczywiście można by wprowadzić standardowy, infiksowy symbol $\in$ zamiast is-memb, co wymagałoby nieznacznej tylko zmiany w sposobie przedstawiania sygnatur. Ale sygnatura jest tylko częścią syntaktyczną opisu typów danych. Podstawowa różnica między opisem ukierunkowanym na modele a opisem ukierunkowanym na właściwości wynika ze sposobu przedstawienia semantyki. W opisie ukierunkowanym na właściwości znaczenie operatorów jest ustalone przez równości [1]). Kluczem do utworzenia tych równości jest — w powyższym przykładzie — fakt, że wszystkie zbiory skończone mogą być utworzone za pomocą operatorów empty i add. Łatwo więc scharakteryzować te operatory, które dostarczają wartości w typach **widocznych zewnętrznie** (np. liczby naturalne i wartości boolowskie).
Przykładowo, równości:

$$\text{is-empty}(\text{empty}()) = \text{true}$$
$$\text{is-empty}(\text{add}(i,s)) = \text{false}$$

obejmują poniekąd oczywiste właściwości operatora is-empty. Podobnie operator is-memb może być opisany równościami:

$$\text{is-memb}(i,\text{empty}()) = \text{false}$$
$$\text{is-memb}(i,\text{add}(j,s)) = (i=j \vee \text{is-memb}(i,s))$$

Podany tu przykład jest bardzo prosty, ale umożliwia omówienie mocnych i słabszych stron specyfikacji ukierunkowanych na właściwości. Najbardziej oczywistą zaletą tej specyfikacji jest to, że nie odwołuje się ona do podstawowych, zdefiniowanych pierwotnie typów danych. W rzeczywistości, rolę modelu odgrywają tu poprawne termy **(algebry słów)** wyprowadzone z generatorów. Bardziej subtelna korzyść wypływa stąd, że cała koncepcja jest oparta na gałęzi matematyki (tzn. algebrze), zajmującej się pojęciami związanymi z typami danych. Niektórymi z tych pojęć są: **sygnatury, sorty, równości** i **modele.** Uogólnienia poszczególnych typów danych (jak np. Set_N na typ zbiorów skończonych parametryzowany typem elementów) dokonuje się stosując specyfikacje ukierunkowane na właściwości.

O wyborze między specyfikacją ukierunkowaną na właściwości a ukierunkowaną na modele powinny zadecydować przesłanki pragmatyczne. Należy jednak dostrzec pewne techniczne trudności związane z opisem ukierunkowanym na właściwości. W przykładzie ze zbiorem Set_N wszystkie operatory są **całkowite.** Mając zadanie zdefiniowania ciągów liczb naturalnych, operator sięgania po pierwszy element listy (hd) powinien być operatorem **częściowym.** Operatory częściowe pojawiają się bardzo często w obliczeniach i podstawowe znaczenie ma sposób, w jaki ich się używa. Związane z tym pierwsze znaczące podejście do posługiwania się algebrami błędów [8] było mniej niż satysfakcjonujące. Bardziej aktualne podejście (np. [4]) pełniej obejmuje problem operatorów częściowych.

Inna trudność jest związana z pytaniem o **interpretacje** takich równości [2]). Wybór interpretacji **początkowej, luźnej** czy **końcowej** jest problemem zbyt technicznym, aby go tu dokonać. Rozdział 9.2 książki [11] zawiera krótkie omówienie potrzeby wprowadzenia dodatkowych operatorów lub równości w podejściu końcowym i początkowym w celu zapewnienia odpowiednich utożsamień. Szerszy opis problemu znajduje się np. w [1] lub [7].

Dużo głębszy problem wynika stąd, że nie wszystko można wyrazić przez specyfikację ukierunkowaną na właściwości. Od dawna wiadomo, że pewnych typów danych nie można scharakteryzować skończonym zbiorem równości. Wypływa stąd konieczność sięgnięcia po tzw. **funkcje ukryte** (ang. hidden functions). Związek pomiędzy tymi funkcjami a modelem jest interesującym tematem badawczym. Obecność funkcji ukrytych osłabia główną zaletę specyfikacji ukierunkowanej na właściwości; ideał, w którym typ danych może być zrozumiany jedynie z użyciem jego operatorów (funkcji) i związków między nimi, staje się nieosiągalny, gdy trzeba wprowadzić nowe funkcje opisujące pewne związki wewnętrzne.

Warto teraz zająć się pytaniami praktycznymi, takimi jak decyzja o wyborze między opisem ukierunkowanym na właściwości a opisem ukierunkowanym na modele przy opisie typów danych. Można rozróżnić typy danych, takie jak Set_N, który nie ma żadnego oczywistego stanu, czy takie jak baza danych, mające operacje [3]), których wykonanie aktualizuje stan bazy. Rzeczywiście, biorąc na przykład stos (Stack) — niemal standardowy przykład specyfikacji typu danych — w sposób naturalny wprowadza się pojęcie stanu. Można zamaskować ten fakt przedstawiając sygnaturę w postaci:

$$\text{emtpy}: \rightarrow \text{Stack}$$
$$\text{push}: \leftarrow X \times \text{Stack} \rightarrow \text{Stack}$$
$$\text{top}: \text{Stack} \rightarrow X$$
$$\text{remove}: \text{Stack} \rightarrow \text{Stack}$$
$$\text{is-empty}: \text{Stack} \rightarrow B$$

Zostały tu jednak rozdzielone dwie operacje (top i remove) tworzące zazwyczaj pojedynczą operację POP, której efektem ubocznym jest zmiana stanu stosu, a wynikiem — żądana wartość. Nie ma szczególnych powodów, dla których nie można by rozszerzyć opisów właściwości tak, aby objęły sygnatury z więcej niż jednym wynikiem [4]) (np. [13]). Na pewno jednak nie da się wówczas równie elegancko przedstawić odpowiednich równości.

Alternatywny sposób specyfikowania ukierunkowany na modele traktuje każdą z operacji oddzielnie. Każdą operację charakteryzują dwa kierunki, wejściowy i wyjściowy, i nie ma żadnych trudności w operowaniu pojęciem stanu. Stosując specyfikację ukierunkowaną na modele napotyka się, jednak niebezpieczeństwo „przespecyfikowania". Problem ten opisano w [10] i [11] jako **orientację na realizację** (ang. implementation bias). Podano tam test ustalający, czy rozpatrywany stan nie wykazuje takiej orientacji.

Wiele operacji definiuje się w odniesieniu do stanu. Stan konstruuje się jako kombinację znanych typów. Wracając do przykładu stosu, podstawowy stan można zdefiniować jako ciąg elementów X. Wówczas operację POP można wyspecyfikować jako:

$$\text{POP}\ ()\ r : X$$
$$\text{ext wr st} : \text{seq of } X$$
$$\text{pre st} \neq [\,]$$
$$\text{post } r = \text{hd } \overline{\text{st}} \wedge \text{st} = \text{tl } \overline{\text{st}}$$

Klauzula ext identyfikuje obiekty, do których operacja ma dostęp. W tym wypadku rozważa się tylko jedną zmienną, gdyż stan jest bardzo prosty. W większych przykładach samo wypisanie potrzebnych zmiennych prowadzi do tzw. „problemu budowy" (ang. „frame problem"). Lepszą charakterystykę potencjalnych wyników operacji uzyskuje się przez odróżnienie dostępu z prawem odczytu (rd) od do-

---

[1]) Z użyciem takich równości algebraicznych wiążą się szerzej stosowane nazwy tego podejścia: specyfikacje równościowe, prezentacje algebraiczne lub nawet specyfikacje algebraiczne. Ta ostatnia nazwa jest nieco niewłaściwa, gdyż obejmuje zarówno prezentowanie modelu liczb wymiernych, jak i podanie w tekście algebraicznym aksjomatów liczb naturalnych (por. [12]).

[2]) Wiąże się z tym pytanie: co się stanie, gdy nie ma dogodnego zbioru operatorów generatorów?

[3]) Użycie terminu operacje zamiast operatory ma na celu podkreślenie roli efektów ubocznych.

[4]) W wypadku występowania niedeterminizmu próba rozdzielenia operacji na funkcje dające pojedyncze wartości jest niepoprawna.

stępu z prawem odczytu i zapisu (wr). Warunek wejściowy jest predykatem stanu i ogranicza liczbę wypadków, w których można zastosować operację (implementując POP należy pominąć stany, w których st jest ciągiem pustym). Warunek wyjściowy jest predykatem dwóch stanów — opisuje związek stanu przed wykonaniem operacji ze stanem po jej wykonaniu. Należy tu rozróżnić dwie wartości tej samej (zewnętrznej) zmiennej. Można przyjąć różne konwencje; w [11] wartości starego stanu są wyróżnione kreską z haczykiem umieszczoną nad nazwą stanu[5] $(\overline{st})$.

Taką specyfikację operacji można przedstawić w bardziej funkcjonalnej postaci:

$$POP : \text{seq of } X \to \text{seq of } X \times X$$

$$\forall \overline{st} \in \text{seq of } X \cdot \text{pre-OP}(\overline{st}) \Rightarrow$$

$$\exists st \in \text{seq of } X,\ r \in X \cdot OP(\overline{st}) = (st, r) \wedge$$

$$\forall st \in \text{seq of } X,\ r \in X \cdot$$

$$OP(\overline{st}) = (st, r) \Rightarrow r = \text{hd } \overline{st} \wedge st = \text{tl } \overline{st}$$

Z powyższej specyfikacji wynika, że operacja kończy wykonanie dla każdego stanu spełniającego warunek wejściowy (oznacza to **całkowitą poprawność** specyfikacji). W tym artykule posłużono się bardziej schematycznym stylem specyfikacji niż w metodzie VDM. Wydzielenie warunku wejściowego w specyfikacji wynika z przesłanek pragmatycznych. Operacje częściowe bardzo często występują podczas konstruowania oprogramowania i warunki wejściowe określają założenia dotyczące dziedzin tych operacji. Bardziej złożone przykłady wykażą pełną przydatność warunków wyjściowych. Do zalet specyfikacji ukierunkowanej na modele można zaliczyć m.in. to, że:

- istnieje możliwość specyfikowania operacji niedeterministycznych;
- często wygodnie specyfikuje się wynik przez koniunkcję różnych właściwości — dzięki temu dużo łatwiej jest opisać wynik, niż go wprost otrzymać;
- podobny wynik daje zastosowanie zaprzeczenia;
- często łatwiej specyfikuje się operację przez operację odwrotną.

Zarówno częściowość jak i niedeterminizm[6] stwarzają problemy przy stosowaniu technik specyfikacji ukierunkowanych na właściwości (por. [13]).

Wadą specyfikacji ukierunkowanej na modele przez warunki wejściowe i wyjściowe jest możliwość specyfikowania operacji nieimplementowalnych (np. obliczenie parzystej liczby pierwszej większej od 10). Pojawia się tu pierwszy z wielu **obowiązków dowodzenia** (ang. proof obligations), stanowiących nieodłączną część metody VDM. Operacja (tj. POP) jest **implementowalna** tylko wtedy, gdy:

$$\forall \overline{st} \in \text{seq of } X \cdot \text{pre-POP}(\overline{st}) \Rightarrow$$

$$\exists st \in \text{seq of } X,\ r \in X \cdot \text{post-POP}(\overline{st}, st, r)$$

Obowiązki dowodzenia nie są normalnie przedmiotem formalnego dowodu, ale przypominają, że informacja o typie, warunek wejściowy i warunek wyjściowy wspólnie decydują o tym, czy operacja jest implementowalna.

MODELE ABSTRAKCYJNE

Jak już wcześniej wspomniano, w specyfikacjach ukierunkowanych na modele każda z operacji może być rozważana oddzielnie. W punkcie tym wykazano, że jeszcze przed rozważaniem operacji można wykorzystać strukturę stanu do badania architektury systemu.

Niech zadanie polega na zbudowaniu projektu i specyfikacji systemu plików. Można tu zaniedbać wewnętrzną strukturę pliku File (może to być ciąg bajtów). Pliki mają swoje nazwy (Name) umożliwiające dostęp. Najprostszy system plików można zdefiniować jako odwzorowanie:

Trivial = map Name to File

[5]) Ze względów technicznych znanych Czytelnikom Informatyki nie można wprowadzić tej notacji w tekście. Dlatego odpowiednie stany oznaczono zwykłą kreską (przyp. red.).

[6]) Interesujące podejście do interpretacji częściowych opisano w artykule [4]. Operuje się w nim pojęciem funkcji poddeterministycznej (ang. under-determined), ale nie niedeterministycznej. Potrzeba tej ostatniej — nawet przy implementacji deterministycznej — wynika ze zmiany równości na różnych poziomach abstrakcji.

Tworzenie odwzorowań, tak jak tworzenie zbiorów czy ciągów, jest jednym ze sposobów budowania obiektów złożonych w metodzie VDM. Teraz można zdefiniować operacje plikowe: CRATE, DELETE i COPY. Trzeba też zauważyć, co jest niemożliwe do zrobienia. Z właściwości odwzorowania wynika, że dwa różne pliki nie mogą mieć tej samej nazwy. Zatem w najprostszym systemie plików różni użytkownicy nie mogą mieć zapewnionych oddzielnych zbiorów nazw. System nie jest dostatecznie bogaty, co daje się zauważyć jeszcze przed specyfikowaniem operacji.

Oddzielne zbiory nazw można utworzyć przez zagnieżdżone katalogi (ang. nested directories). Stan tak wzbogaconego systemu plików może być określony następująco:

Nestedfs = Directory
Directory = map Name to Node
Node = Directory ∪ File

Taki system umożliwia różnym użytkownikom korzystanie z tych samych nazw. Struktura katalogów pod wieloma względami przypomina system Unix i jemu podobne. Koncepcja węzła Node dopuszcza występowanie plików i katalogów w tym samym katalogu.

Teraz już można określić operacje na Nestedfs. Rozsądnie będzie jednak sprawdzić, czego w tym stanie zrobić nie można. Nie można dotrzeć do tego samego pliku File przez różne nazwy ścieżek (ciągów nazw). Dopuszczenie do tego, aby różne nazwy ścieżek udostępniały ten sam plik, wymaga ponownego rozszerzenia pojęcia stanu.

Istnieje względnie standardowa metoda wprowadzenia do specyfikacji takiego mechanizmu współdzielenia. Jeśli wprowadzi się wiązanie pośrednie[7]) Fid :

Sharedfs :: root : Directory
filem : map Fid to File
Directory = map Name to Node
Node = Directory ∪ Fid

to plik może być udostępniony tak, jak poniżej:

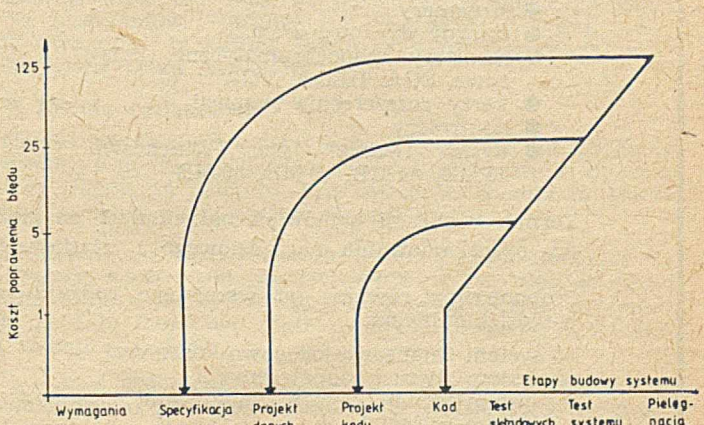
$$\text{mk-Sharedfs}(\{id_1 \mapsto fid_1,$$
$$id_2 \mapsto \{id_1 \mapsto fid_2, id_2 \mapsto fid_1\}\},$$
$$\{fid_1 \mapsto file_a, fid_2 \mapsto file_b\})$$

Można zdefiniować operacje. Pierwsza operacja udostępnia zawartość katalogu:

Dirstatus = map Name to {FILE, DIR}
SHOW () r : Dirstatus
ext rd d : Directory

post r = {nm ↦ (if d(nm) ∈ Directory then DIR
else FILE) | nm ∈ dom d}

Inna operacja dodaje nowy katalog do zawartości już istniejącego:



Koszty i rodzaje błędów

[7]) Notacja obiektów złożonych przypomina w VDM rekordy w Pascalu. W tym wypadku:
Sharedfs =
{mk-Sharedfs(root, filem) | root ∈ Directory ∧
filem ∈ map Fid to File}

MKDIR (n : Name)
ext wr d : Directory
pre n $\notin$ dom d
post $d = \overline{d} \cup \{n \mapsto \{\}\}$

Operacja wstawiania nowego pliku ma następującą postać:

MKFILE (n : Name, f : File)
ext wr d : Directory
wr fm : map Fid to File
post $\exists fid \in Fid \cdot$
$fid \notin dom\ \overline{fm} \wedge d = \overline{d} \cup \{n \mapsto fid\}$
$\wedge\ fm = \overline{fm} \cup \{fid \mapsto f\}$

Zainteresowany Czytelnik może zdefiniować inne operacje (np. usuwania) na poziomie katalogu Directory.

Znaczenie prac nad specyfikacjami ukierunkowanymi na właściwości wzrosło m.in. dzięki badaniom nad dogodnymi metodami budowania specyfikacji o złożonej strukturze. W rozdziale 7.4 książki [11] opisano wykorzystanie techniki **uogólniania** operacji z jednego typu danych na inne typy przez **usyntaktycznienie operacji** (ang. operation quotation). Technika ta umożliwia zastosowanie operacji na pojedynczym katalogu do pełnej struktury katalogu. W tym celu stan mógłby być rozszerzony o nowe składniki zawierające — na przykład — bieżącą ścieżkę.

W przykładzie z katalogiem zilustrowano metodę badania architektury systemu przez jego stan. W innych przykładach w [11] pokazano, jak na różnym poziomie abstrakcji badać na przykład system pamięci wirtualnej i jakie stąd, w zależności od poziomu rozważań, wypływają wnioski. W rozdziale 8.3 pokazano, jak zgodnie z tą samą ideą opisać operacje wejścia-wyjścia. W artykule [3] podane idee zastosowano do języków programowania (por. też [2]).

Tłumaczył i opracował:
WOJCIECH PACHOCKI

LITERATURA

[1] Bauer F. L., Wössner H.: Algorithmic Language and Program Development. Springer-Verlag, 1982
[2] Bjørner D., Jones C. B.: Formal Specification and Software Development. Prentice-Hall International, 1982
[3] Bjørner D., Prehn S., Oest O.: Software Engineering Aspects of VDM. Materiały Szkoły Jesiennej PTI „Współczesne kierunki rozwoju informatyki", Mrągowo, 4—8 listopada 1985
[4] Broy M.: Partial Interpretations of Higher Order Algebraic Types. Marktoberdorf Summer School, 1986 (w druku)
[5] CIP Language Group: The Munich Project CIP. Volume 1. The Wide Spectrum Language CIP-L. Lecture Notes in Computer Science, Vol. 183, Springer-Verlag, 1985.
[6] Constable R.L. et al.: Implementing Mathematics with the Nuprl Proof Development System. Prentice-Hall International, 1986
[7] Ehrig H., Mahr B.: Fundamentals of Algebraic Specification 1 — Equations and Initial Semantics. EATCS Monographs on Theoretical Computer Science, Springer-Verlag, 1985
[8] Goguen J. A.: Abstract Errors for Abstract Data Types. Formal Descriptions of Programming Concepts, Neuhold E. J. (ed.), North-Holland, 1978
[9] Herzog O.: Formal Development Methods in Industrial Environments. NATO Conference, Nyborg, Denmark, May 1984
[10] Jones C. B.: Implementation Bias in Constructive Specifications of Abstract Objects. 1977
[11] Jones C. B.: Systematic Software Development using VDM. Prentice-Hall International, 1986
[12] MacLane S., Birkoff G.: Algebra. Second Edition, Collier Macmillan International, 1979
[13] Nipkow T.: Nou-Deterministic Data Types — Models and Implementations. Acta Informatica, Vol. 22, pp. 629—661, 1986.

MIĘDZYWOJEWÓDZKA SPÓŁDZIELNIA PRACY „SIÓDEMKA"

ŁÓDŹ, AL. KOŚCIUSZKI 93,

ZAKŁAD INFORMATYKI I SYSTEMÓW KOMPUTEROWYCH
poleca po **najniższych cenach** w kraju:

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

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

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

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

Proponujemy również na wszelkiego rodzaju mikrokomputery programy wspomagające zarządzanie przedsiębiorstwem:

— system finansowo-księgowo-kosztowy
— system zbytu i zaopatrzenia
— system technicznego przygotowania produkcji
— system materiałowy
— system kadrowy i kadrowo-płacowy (również dla pracowników akordowych)
Wymienione systemy pracują w wersjach sieciowych i wielodostępnych.

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

EO/1005/87

WITOLD STANISZKIS
Centrum Projektowania i Zastosowań Informatyki
Warszawa

Zarządzanie rozproszonymi danymi — przegląd problematyki (I)

W ostatnich dziesięciu latach zarządzania rozproszonymi danymi było przedmiotem intensywnych badań. Istnieje obszerna literatura dotycząca tej tematyki, lecz tylko w nielicznych pracach (np. [2, 7]) podjęto próbę scharakteryzowania dziedziny zarządzania rozproszonymi danymi w sposób zwięzły i pełny.

Celem artykułu jest ogólne wprowadzenie do tematyki zarządzania rozproszonymi danymi. Wprowadzenie to obejmuje większość zagadnień badawczych oraz prac rozwojowych. Główny nacisk położono na tematykę systemów geograficznie rozproszonych, lecz omówiono również pewne zagadnienia odnoszące się do lokalnych sieci komputerowych, uważane za istotne dla obu klas systemów.

PODSTAWOWE POJĘCIA

Warto najpierw przypomnieć niektóre podstawowe, ogólnie przyjęte pojęcia i definicje.

Rozproszona baza danych (RBD, rozproszona BD) jest to kolekcja danych, które logicznie należą do tego samego systemu, lecz są rozmieszczone w różnych węzłach sieci komputerowej. W definicji tej kładzie się nacisk na dwa jednakowo ważne aspekty RBD:

- *rozproszenie*, tj. fakt, że dane nie znajdują się w tym samym miejscu (procesorze); dzięki temu rozproszoną BD można odróżnić od scentralizowanej;
- *logiczną korelację*, tj. fakt, że dane mają pewne właściwości, które je ze sobą wiążą; dzięki temu rozproszoną BD można odróżnić od zbioru lokalnych BD lub plików, które znajdują się w różnych węzłach sieci komputerowej.

Jak widać, nie podkreśla się faktu, że poziom tej integracji jest zdeterminowany przez schemat logiczny bazy danych. Fakt ten staje się jednak ważnym czynnikiem umożliwiającym odróżnienie od siebie dwóch odmiennych klas systemów zarządzania rozproszonymi danymi, tj. systemów zarządzania rozproszonymi bazami danych (systemów ZRBD) od *systemów z wieloma bazami danych* (ang. multidatabase systems).

Schemat globalny definiuje wszystkie dane, które są zawarte w rozproszonej BD. W wypadku systemów ZRBD, schemat globalny zbiega się z logicznym schematem bazy danych. W systemach z wieloma bazami danych wprowadza się pojęcie schematu globalnego jako środka integracji schematów logicznych poszczególnych lokalnych scentralizowanych systemów zarządzania bazami danych.

Systemy ZRBD realizują nową formę niezależności danych, tzw. *przezroczystość miejsca* (ang. distribution transparency). Przezroczystość miejsca oznacza, że pytania i programy użytkowe mogą być pisane tak, jak gdyby baza danych nie była rozproszona. Zatem przenoszenie danych z jednego węzła do innego nie ma wpływu na poprawność pytań i programów. Może mieć ono jednak ogromne znaczenie dla szybkości przetwarzania. Pojęcie przezroczystości miejsca będzie omawiane w kontekście relacyjnej struktury danych, ponieważ w systemach ZRBD relacyjna struktura danych występuje najczęściej.

Jest to — dokonane za zgodą Autora — tłumaczenie referatu opublikowanego w Proc. 8th Intern. Seminar on Database Management Systems, Pieštany, Czechosłowacja, 1985.

Schemat globalny zawiera definicje wszystkich *globalnych relacji*. W wypadku systemów ZRBD są nimi relacje schematu logicznego lub tzw. relacje bazowe. Każda relacja globalna może być podzielona na kilka nie nakładających się porcji, nazywanych *fragmentami*. Podział taki (czyli fragmentację) można zrealizować kilkoma różnymi sposobami.

Schemat fragmentów (ang. fragmentation scheme) określa odwzorowanie między relacjami globalnymi a fragmentami. Jest to odwzorowanie jeden-wiele, tj. kilka fragmentów odpowiada jednej relacji globalnej, lecz tylko jedna relacja globalna odpowiada jednemu fragmentowi. Fragmenty są porcjami logicznymi, które są fizycznie rozmieszczone w jednym lub kilku węzłach sieci komputerowej.

Schemat alokacji definiuje rozmieszczenie poszczególnych fragmentów na stanowiskach. Sposób odwzorowania określony na schemacie alokacji przesądza, czy rozproszona baza danych jest redundantna czy nieredundantna. W pierwszym przypadku odwzorowanie jest typu jeden-wiele, w drugim natomiast jest jeden-jeden.

Dekompozycja relacji globalnych na fragmenty może być dokonana przy zastosowaniu *fragmentacji poziomej* lub *fragmentacji pionowej*. Poszczególne rodzaje fragmentacji można stosować oddzielnie lub łącznie. Przy dekomponowaniu relacji globalnych muszą być uwzględnione następujące reguły:

- **Reguła zupełności**

We fragmentach muszą być odwzorowane wszystkie dane należące do relacji globalnej, tj. nie może zdarzyć się taka sytuacja, że jakaś jednostka danych należy do relacji globalnej, a nie należy do żadnego fragmentu.

- **Reguła rekonstrukcji**

Zawsze musi być możliwe odtworzenie relacji globalnej z jej fragmentów. Warunek ten jest oczywisty, jeśli wziąć pod uwagę, że w każdej chwili w bazie danych są przechowywane tylko fragmenty.

- **Reguła rozłączności**

Często jest pożądane jawne kontrolowanie na poziomie programów aplikacyjnych procesu tworzenia replik danych, zatem fragmenty powinny być rozłączne. Reguła ta stosuje się głównie do fragmentacji poziomej.

Fragmentacja pozioma dzieli krotki relacji globalnej na podzbiory. Każdy taki podzbiór może na przykład zawierać krotki mające jakąś wspólną cechę geograficzną. Ten sposób podziału może być zdefiniowany jako operacja selekcji na relacji globalnej.

Pochodna fragmentacja pozioma (ang. drived horizontal fragmentation) jest wyprowadzana z fragmentacji poziomej dokonanej na jakiejś innej relacji. Przy takim podziale określenie, które krotki mają trafić do których fragmentów, wymaga operacji półpołączenia (ang. semi-join).

Fragmentacja pionowa dzieli atrybuty globalnej relacji na grupy, a fragmenty są uzyskiwane przez dokonanie projekcji relacji globalnej względem poszczególnych grup atrybutów.

Fragmentacja mieszana dzieli globalną relację na fragmenty uzyskiwane w wyniku superpozycji powyższych operacji.

System ZRBD może realizować wiele poziomów przezroczystości miejsca. Na każdym poziomie mogą być ukryte przed użytkownikiem różne aspekty rzeczywistej dystrybucji danych. Na poziomie najwyższym, zwanym *przezroczystością fragmentacji*, modyfikacje rozmieszczenia danych nie mają wpływu na zastosowania, tj. na pytania i programy użytkowe. Ogólnie, zapewnienie przezroczystości miejsca dla zastosowań aktualizujących jest znacznie trudniejsze niż dla zastosowań tylko odczytujących. Poziom przezroczystości miejsca jest jednym z najważniejszych cech systemu ZRBD.

Obecnie realizuje się wiele projektów, obejmujących jeden lub kilka obszarów zarządzania rozproszonymi danymi. Kilka prototypowych systemów jest już gotowych, bądź w trakcie opracowywania na uniwersytetach i w przemysłowych laboratoriach badawczych. Opisy prototypów są w literaturze dobrze udokumentowane, jednak dostępnych informacji na temat doświadczeń z użytkowania takich systemów jest bardzo mało. Wynika to prawdopodobnie stąd, że tylko nieliczne systemy są wykorzystywane przez „prawdziwych" użytkowników. Z artykułów opisujących systemy prototypowe często trudno jest wywnioskować, które z właściwości tych systemów są tylko zaprojektowane, a które są rzeczywiście zaimplementowane i przetestowane.

W ostatnich latach, w dziedzinie zarządzania rozproszonymi danymi pojawił się nowy kierunek badawczy, zmierzający do zintegrowania już istniejących baz danych i plików, tj. tworzenia systemów z wieloma bazami danych (systemów multibaz danych). Taka integracja jest uważana za jedną z najważniejszych potrzeb użytkownika. Systemy multibaz mają właściwości, które istotnie różnią je od systemów ZRBD.

Systemy multibaz danych oraz systemy ZRBD są nazywane *systemami o współdzielonych rozproszonych danych*.

ARCHITEKTURY SYSTEMÓW O WSPÓŁDZIELONYCH ROZPROSZONYCH DANYCH

Architektury systemów rozproszonych można odróżnić od siebie na podstawie następujących głównych właściwości:

- **Definicja fragmentu.** Jest to zdolność do jawnego definiowania schematu fragmentów. Oznacza, że system stosuje fragmentacje danych w celu zapewnienia integralności danych oraz optymalizowania realizacji pytań.

- **Definicja replik.** Jest to możliwość definiowania takiego schematu alokacji, w którym te same fragmenty można umieszczać w różnych węzłach sieci komputerowej. Implikuje możliwość użycia kontrolowanej redundancji danych w optymalizacji pytań, jak również — istnienie algorytmu sterowania aktualizowaniem, który pozwala utrzymywać integralność redundantnych danych.

- **Funkcja celu optymalizacji.** Funkcja celu w optymalizacji pytań jest zazwyczaj zdefiniowana jako liniowa kombinacja głównych czynników kosztu, tzn.: kosztu transmisji komunikatów, kosztu wejścia-wyjścia oraz kosztu działania jednostki centralnej (CPU). Zasadnicze różnice między strategiami optymalizacji pytań wynikają z rozmaitości opinii dotyczących relatywnej ważności powyższych czynników kosztów.

- **Czas wiązania pytań** (ang. query binding time). Dwa ekstremalne punkty wiązania odpowiadają pełnej kompilacji pytań w okresie ich planowania oraz w pełni interpretacyjnemu wykonywaniu pytań. Zalety i wady wczesnego i późnego wiązania są oczywiste i zostały już wystarczająco rozpatrzone w pracach z dziedziny zarządzania scentralizowanymi bazami danych. Celowe wydaje się kompilowanie (przynajmniej częściowo) pytań zanurzonych w takie programy apiikacyjne, których wykonywanie jest oczekiwane często. W odniesieniu do zarządzania danymi rozproszonymi staje się to tym ważniejsze, że w tym wypadku algorytmy optymalizacji pytań są znacznie bardziej złożone.

- **Planowanie pytań.** Jeżeli pytanie wymaga dostępu do danych przechowywanych w wielu węzłach, to węzły te mogą w różny sposób wzajemnie na siebie oddziaływać w celu określenia strategii wykonania tego pytania. Możliwe warianty są następujące:

— planowanie pytania w miejscu jego przedłożenia (podejście scentralizowane),
— podejmowanie mniej ważnych decyzji w innych węzłach sieci (podejście półscentralizowane),
— konstruowanie planu we wzajemnej współpracy we wszystkich węzłach (podejście rozproszone).

Podejście scentralizowane do planowania pytań wyklucza możliwość modyfikowania planu na podstawie danych ze sprzężenia zwrotnego.

- **Sterowanie współbieżnością** (ang. concurrency control). Trzy główne podejścia do sterowania współbieżnością są oparte na blokowaniu (ang. locking), etykietach czasowych, datownikach (ang. locking), etykietach czasowych, datownikach (ang. time stamps) oraz optymizmie. W większości systemów ZRBD przyjęto podejście oparte na blokowaniu w formie protokołu blokowania dwufazowego. Problemy sterowania współbieżnością występujące w wypadku systemów z wieloma bazami danych, przedstawiono w [4].

- **Model danych.** Z uwagi na inherentne właściwości systemów rozproszonych wszystkie modele danych są ukierunkowane na zbiory. W większości systemów rozproszonych przyjmuje się relacyjny model danych. Pytania wysokiego poziomu, odnoszące się do modelu relacyjnego, umożliwiają istotną optymalizację oraz opracowanie ukierunkowanych na zbiory środków pomocniczych sprzyjających efektywnemu wykonywaniu pytań.

SYSTEMY ZARZĄDZANIA ROZPROSZONYMI BAZAMI DANYCH

Poniżej zaprezentowano wybrane systemy o współdzielonych rozproszonych danych, mające największy udział w ewolucji zarządzania rozproszonymi bazami danych. Wszystkie prezentowane systemy, z wyjątkiem systemu ENCOMPASS, są obecnie na etapie prototypów badawczych. Niektóre z nich służą jako narzędzia do testowania nowych komercyjnych produktów programowych.

Distributed Database Manager (DDM)

DDM został zaprojektowany i opracowany w Computer Corporation of America [3]. Jego celem jest umożliwienie obsługi pytań zanurzonych w programy napisane w języku Ada. DDM może obsługiwać pytania wyrażone w języku Daplex (język pytań dla funkcjonalnego modelu danych). Dopuszcza się fragmentację poziomą oraz definiowanie replik. Pytania są kompilowane, a kryteriami optymalizacji są: koszt komunikacji oraz koszt CPU. Planowanie pytań jest scentralizowane, co jest konsekwencją założenia o wczesnym wiązaniu. Podstawą strategii sterowania współbieżnością jest mechanizm blokowania oraz algorytm scentralizowanego wykrywania zakleszczeń.

Distributed Database Testbed System (DDTS)

DDTS, opracowany w Honeywell Corporate Computer Sciences Center [12], jest próbą implementacji uogólnionej wersji trzypoziomowej architektury ANSI/SPARC na komputerach Honeywell Level 6. Konceptualny model danych jest oparty na podejściu byt-związek (ang. entity-relationship, ER), językiem danych jest Gordas, a funkcje zarządzania lokalną bazą danych są realizowane przez system IDS/II (Komitetu Codasyl). W DDTS nie ma żadnych możliwości fragmentacji, opracowano w nim natomiast środki do definiowania replik. Obecnie nie są dostępne żadne informacje na temat optymalizacji pytań. Przyjęto zasadę kompilacji pytań oraz półscentralizowanego planowania pytań. Sterowanie współbieżnością jest oparte na blokowaniu i zapobieganiu zakleszczeniom.

Distributed INGRES

Distributed INGRES jest ewolucją relacyjnego systemu INGRES, opracowanego w University of California w Berkeley. Dopuszcza się poziomą fragmentację danych, repliki natomiast nie mogą być w ogóle definiowane. Kryteriami optymalizacji pytań w tym systemie są koszty komunikacji i CPU. Przyjęto strategię interpretacyjnego wykonywania pytań. Planowanie pytań jest scentralizowane. Schemat sterowania współbieżnością jest oparty na blokowaniu.

ENCOMPAS

ENCOMPASS jest relacyjnym systemem rozproszonym, opracowanym w TANDEM Corp. [1, 10]. Głównym celem tego systemu jest wsparcie rozwoju zastosowań rozproszonego, interakcyjnego przetwarzania transakcji. Dopuszcza się fragmentację poziomą, nie dopuszcza się natomiast definiowania replik. Zastosowano algorytm blokowania z użyciem schematu rozwiązywania zakleszczeń, oparty na zasadzie przeterminowania (ang. time-out).

POREL

POREL jest systemem relacyjnym opracowanym na uniwersytecie w Stuttgarcie dla sieci minikomputerów firmy Dec [11]. Daje on możliwość fragmentacji poziomej oraz definiowania replik. Kryterium optymalizacji pytań jest koszt komunikacji. Zaimplementowano wiązanie podczas kompilacji oraz optymalizację scentralizowaną. Sterowanie współbieżnością jest oparte na strategii blokowania z unikaniem zakleszczeń.

R*

R* jest prototypem eksperymentalnym zaprojektowanym w IBM San Jose Research Laboratory [5]. System R*, będący rozwinięciem systemu R, jest obecnie poddawany wielostronnym badaniom eksperymentalnym z udziałem wybranych klientów. Jawnie nie jest w nim dozwolona ani fragmentacja, ani definiowanie replik. Przyjętym kryterium optymalizacji pytań są koszty komunikacji, wejścia-wyjścia oraz CPU. Pytania są kompilowane (częściowo) w miejscu przedłożenia. W planowaniu pytań przyjęto podejście półscentralizowane. Sterowanie współbieżnością jest oparte na blokowaniu oraz rozproszonej detekcji zakleszczeń.

SDD-1

SDD-1 jest relacyjnym systemem rozproszonym zaprojektowanym i zaimplementowanym w Computer Corporation of America [14, 15]. SDD-1 jest pierwszym systemem ZRBD, eksperymentalnie eksploatowany w sieci ARPA. Dopuszcza zarówno poziomą, jak i pionową fragmentację oraz definiowanie replik. Kryterium optymalizacji jest koszt komunikacji. Jako strategię przetwarzania pytań przyjęto zasadę interpretacyjnego ich wykonywania oraz scentralizowanego planowania. Sterowanie współbieżnością jest oparte na algorytmie datowników, etykiet czasowych oraz algorytmie unikania zakleszczeń. Ten ostatni wprowadza jednak poważne ograniczenia dotyczące sposobu przetwarzania transakcji.

SIRIUS-DELTA

Projekt SIRIUS jest ogólnonarodowym projektem zainicjowanym przez rząd francuski, a koordynowanym przez instytut INRIA. Jego wynikiem jest wiele prototypów systemów rozproszonych. SIRIUS-DELTA [6] jest systemem relacyjnym zaimplementowanym i działającym w INRIA. Dopuszcza się w nim zarówno fragmentację poziomą, jak i pionową oraz definiowanie replik. Kryterium optymalizacji jest koszt komunikacji. Planowanie pytań jest scentralizowane. Pytania są wiązane na wczesnym etapie przy kompilacji. Do rozwiązania problemu sterowania współbieżnością zaproponowano strategię zapobiegania zakleszczeniom.

VDN

VDN [8, 9] jest relacyjnym systemem rozproszonym zaimplementowanym w NIXDORF Computer AG dla mikroprocesorów firmy Intel. System umożliwia fragmentację poziomą oraz definiowanie replik. Planowanie pytań jest scentralizowane. Sterowanie współbieżnością pozostawiono użytkownikowi, wyposażając go w odpowiednie środki do blokowania.

RODAN*

RODAN* jest rozproszonym systemem zarządzania bazą danych opracowywanym obecnie w Centrum Projektowania i Zastosowań Informatyki w Warszawie [13]. Schemat globalny jest typu relacyjnego, natomiast schematy lokalne mogą być albo relacyjne, albo sieciowe. W systemie jest utrzymywana pełna przezroczystość miejsca, nie jest natomiast możliwa żadna forma fragmentacji. Kryterium optymalizacji pytań są koszty transmisji, wejścia-wyjścia oraz CPU. Planowanie pytań jest scentralizowane i oparte na charakterystykach statystycznych bazy danych. System zapewnia możliwość przetwarzania transakcyjnego.

* * *

Powyższe prototypowe systemy pozwoliły zbudować środowiska laboratoryjne do badań eksperymentalnych i testowania różnorodnych algorytmów wchodzących w skład systemu ZRBD. W szczególności, przedmiotem intensywnych eksperymentów były algorytmy przetwarzania pytań oraz sterowania współbieżnością. Przy okazji wypłynął także problem odporności na błędy procesorów obliczeniowych wykonywanych w środowisku RBD. Ogólnie uważa się, że aktualnie istniejące prototypy badawcze stanowią dobry punkt wyjścia dla opracowania komercyjnych produktów programowych.

Tłum. i oprac.:
WIKTOR RZECZKOWSKI

LITERATURA

[1] Borr A.: Transaction monitoring in ENCOMPASS — Reliable Distributed Transaction Processing. Proc. 7th Int. Conf. on VLDB, Cannes, 1981

[2] Ceri S., Pelagatti G.: Distributed Databases — Principles and Systems. McGraw-Hill, New York, 1984

[3] Chan A., et al.: Overview of an Ada Compatible Distributed Database Manager. Proc. ACM SIGMOD'83 Int. Conf. on Management of Data, 1983

[4] Gligor V. D., Popescu-Zeletin R.: Concurrency Control Issues in Distributed Heterogeneous Database Management Systems. Proc. 3rd Int. Seminar on Distributed Data Sharing Systems, Parma, Italy, March 1984; F. A. Schreiber, W. Litwin (eds.), North-Holland, Amsterdam 1985

[5] Haas L. M., et al.: R* — A Research Project on Distributed Relational DBMS. Database Engineering, Vol. 5, No. 2, December 1982

[6] Litwin W. et al.: SIRIUS Systems for Distributed Data Management. Proc. 2nd Int. Symposium on Distributed Data Bases. H.-J. Schneider (ed.), North-Holland, Amsterdam, 1982

[7] Mohan C.: Recent and Future Trends in Distributed Data Base Management. Proc. New York University Symposium on New Directions in Database Systems, Stony Brook (NY), 1984

[8] Munz R.: Gross Architecture of the Distributed Database System VDN. Proc. IFIP Working Conference on Database Architecture, G. Bracchi, G. M. Nijssen (eds.), Venice, 1979

[9] Munz R.: The Distributed Database System VDN. Database Engineering, Vol. 5, No. 2, December 1982

[10] Nauman J.: ENCOMPASS — Evolution of a Distributed Database/Transaction System. Database Engineering, Vol. 5, No. 2, December 1982

[11] Neuhold E., Walter B.: Overview of the Architecture of the Distributed Data Base System POREL. Proc. 2nd Int. Symposium on Distributed Data Bases. H.-J. Schneider (ed.), North-Holland, Amsterdam, 1982

[12] Rahimi S. K. et al.: A Structured View of Honeywell's Distributed Database Testbed System — DDTS. Database Engineering, Vol. 5, No. 2, December 1982

[13] Rodan*. Podręcznik projektowania systemu. CPiZI, Warszawa, 1985

[14] Rothnie J. B., Goodman N.: An Overview of the Preliminary Design of SDD-1 — A System for Distributed Databases. Proc. 2nd Berkeley Workshop on Distributed Data Management and Computer Networks, 1977

[15] Rothnie J. B. et al.: Introduction to a System for Distributed Databases (SDD-1). ACM Trans. on Database Systems. Vol. 5, No. 1, March 1980

[16] Stonebraker M. et al.: Performance Analysis of Distributed Data Base Systems. Database Engineering, Vol. 5, No. 2, December 1982.

Obliczenia na komputerze IBM 370/148

wyposażonym m. in.
w dyski magnetyczne 200 Mb,

wykona

w oparciu o programy zleceniodawców

Centrum Informatyki Handlu Zagranicznego

ul. Stępińska 9, 00-739 Warszawa
tel. 41-55-78

Umowy długookresowe z rabatem
System operacyjny VS1 lub własny zleceniodawcy
Możliwość pracy na 1 zmianie

EO/700/87

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

MULTICOMP — system rozwiązywania zadań metodą przeszukiwania drzew (2)

W pierwszej części artykułu omówiono metody rozwiązywania pewnej klasy zadań za pomocą przeszukiwania drzew. Poniżej przedstawiono język Multicomp [2, 3, 4] służący do opisu i rozwiązywania zadań tego rodzaju.

OPIS MULTICOMPU

Użytkownik opisuje za pomocą Multicompu właściwości (parametry) rozwiązywanego zadania:

- postać węzłów, które odpowiadają poszczególnym stanom procesu rozwiązywania,
- węzeł początkowy,
- warunki, które musi spełniać węzeł końcowy,
- operatory, za pomocą których można realizować przejście z jednego stanu do drugiego.

System automatycznie realizuje przeszukiwanie przestrzeni stanów w celu znalezienia rozwiązania. Użytkownik może wpływać na przebieg rozwiązywania, podając — na przykład — proponowaną strategię. W wypadku niezadeklarowania parametru otrzymuje on wartość standardową. Wiele parametrów definiuje się przez programy w Lispie, stąd znajomość tego języka jest konieczna do pełnego korzystania z możliwości Multicompu.

Program w Multicompie ma następującą strukturę (wydruk 1):
— definicja programu operacyjnego za pomocą deklaracji parametrów przeszukiwania przestrzeni stanów,
— definicje funkcji pomocniczych,
— wywołanie programu operacyjnego.

```
(*MULTICOMP* '(
  DEFINE nazwa (parametry) ;
    INITIAL ...
    OPERATORS...
    SOLUTION...
    ...
  END ;

    DEF fun1 (parametry)  ;
    ...
    END  ;

    DEF fun2 (parametry)  ;
    ...
    END  ;

  EXECUTE nazwa (parametry) ;
))
```

Wydruk 1

Węzeł w programie jest reprezentowany przez listę składającą się z siedmiu elementów:

NC — numer węzła,
GS — lista operatorów stosowanych do węzła,
QS — opis stanu,
AS — numer poprzednika,
SD — głębokość węzła w drzewie,
G — koszt osiągnięcia danego węzła,
H — heurystyczna funkcja określająca, w jakim stopniu dany węzeł jest odległy od rozwiązania.

Przykład węzła, który mógłby powstać podczas rozwiązywania problemu komiwojażera, jest następujący:

```
(52 (oper1 oper 2) (Warszawa Londyn Rzym) 44  3  1200  50)
NC      GS              QS                AS SD GS    H
```

Oprócz powyższych siedmiu współrzędnych można zdefiniować dowolną liczbę własnych współrzędnych, umieszczonych na końcu listy reprezentującej węzeł. Do nadania wartości współrzędnym węzła początkowego służy parametr INITIAL. Na przykład, deklaracja:

```
INITIAL (QS '(JAKIŚ STAN) (H 1000) (NOWA 5) );
```

powoduje nadanie współrzędnym QS i H węzła początkowego odpowiednio wartości (JAKIS STAN) i 1000. Jednocześnie zadeklarowana jest zmienna NOWA będąca ósmą współrzędną węzła. Współrzędna ta dla węzła początkowego otrzymuje wartość 5. Pozostałe współrzędne otrzymują wartości standardowe.

Opisując zadanie można odwołać się do danej współrzędnej węzła w następujący sposób:

(NC N) oznacza współrzędną NC aktualnego węzła,
(GS N) oznacza współrzędną GS itp.

Tak więc, jeśli N jest węzłem początkowym zadeklarowanym jak w powyższym przykładzie, to wywołanie (NC N) ma wartość 0, wywołanie (QS N) udostępni listę (JAKIŚ STAN), a (NOWA N) przyjmuje wartość 5.

Do zadeklarowania i zdefiniowania operatorów, za pomocą których można tworzyć następniki węzłów, używa się instrukcji OPERATORS. Jeśli operatory są podane w postaci listy (<nazwa operatora> <wartość>), to traktowane są jako funkcje. Funkcje te powinny być napisane w Lispie. Na przykład konstrukcja:

```
OPERATORS '( (OP1 (FUNCTION (LAMBDA (X)
                  (OP2 (FUNCTION (LAMBDA (X)
(CAR X)))
(CADR X)))));
```

deklaruje operator OP1 obliczający pierwszy element listy (będącej jego argumentem) oraz operator OP2 udostępniający drugi element listy (CAR i CADR są odpowiednimi funkcjami Lispu). Na podstawie parametru OPERATORS tworzona jest lista PZO (początkowy zbiór operatorów), w której są zapisywane nazwy wszystkich operatorów. Lista ta jest traktowana jako GS węzła początkowego (o ile w deklaracji INITIAL nie zadecydowano inaczej). W powyższym przykładzie wartością zmiennej PZO stanie się lista (OP1 OP2).

Jeśli operatory są nazwami zmiennych, a nie dwuelementowymi listami, to lista tych nazw jest traktowana jako PZO.

Sposób wykorzystania operatorów do tworzenia nowych węzłów jest podany za pomocą instrukcji NEW. Konstrukcja ta opisuje metody tworzenia elementów następnika z poprzednika i operatora. Dla każdej współrzędnej, której tworzenie ma być opisane, należy podać program w Lispie, którego argumentami są: poprzednik N i operator OP. Program ten powinien udostępniać wartość deklarowanej współrzędnej. Nazwy N i OP są zastrzeżone.

Na przykład, aby QS tworzonego węzła było wynikiem działania operatora OP na odwróconą listę QS poprzednika, a współrzędna H była równa iloczynowi współrzędnych G i H poprzednika, można napisać:

```
NEW (QS (EVALQUOTE OP (CONS (REVERSE (QS N))
NIL)))
      (H (* (G N) (H N)));
```

Argumentami funkcji WDO (warunek dobrego operatora) są: operator OP i węzeł N. Funkcja ta jest predykatem sprawdzającym, czy operator OP ma być zastosowany do węzła N. Wartością tego predyktu powinno być T, jeżeli dany operator ma być zastosowany do danego węzła, a NIL w przeciwnym wypadku. System automatycznie przegląda GS danego węzła i pozostawia na tej liście tylko operatory spełniające warunek zadeklarowany w konstrukcji WDO.

Warunek, który musi spełniać węzeł, aby był traktowany jako rozwiązanie, opisuje funkcja SOLUTION. Przykładowo, instrukcja SOLUTION (MEMBER (QS N) 'STOP) ; spowoduje, że węzeł zawierający w liście QS wyraz STOP będzie uznany za rozwiązanie problemu.

Parametrem STRATEGY wybiera się jedną z kilku podstawowych strategii:

BREADTH_FIRST — strategia przeszukiwania wszerz,
DEPTH_FIRST — strategia przeszukiwania w głąb,
DEPTH_FIRST_1 — strategia przeszukiwania w głąb z jednym następnikiem,
ORDERED_SEARCH — strategia uporządkowanego wyboru,
ORDERED_SEARCH_1 — strategia uporządkowanego wyboru z jednym następnikiem.

Oprócz wymienionych jest jeszcze kilkanaście innych parametrów (m.in. funkcje wybierające węzły i operatory, funkcje redagujące wydruk i ograniczające przestrzeń poposzukiwań).

Dotychczas Multicomp był stosowany m.in. do:
— projektowania układów kombinacyjnych i sekwencyjnych [2],
— syntezy blokowej układów cyfrowych [3],
— testowania i analizy automatów [1],
— rozwiązywania zadań z teorii grafów i badań operacyjnych [4],
— planowania działań robotów [1].

Poniżej przedstawiono przykładowe proste programy w Multicompie.

ZNAJDOWANIE CIĄGU WYRAZÓW

Zadanie polega na znalezieniu ciągu wyrazów, z których każdy następny różni się od poprzedniego jedną literą. Drzewo rozwiązania tego zadania przedstawiono w pierwszej części artykułu. Program w Multicompie rozwiązujący to zadanie przedstawiono na wydruku 2.

Opisem stanu (współrzędną QS węzła) jest lista składająca się z dotychczas znalezionych wyrazów. Ze względu na efektywność programu kolejne wyrazy dołącza się na początek listy QS.

Opisem stanu początkowego jest lista składająca się z wyrazu początkowego WORD1.

Postać parametru SOLUTION wynika z faktu, że rozwiązanie jest znalezione wówczas, gdy pierwszy element listy QS jest taki sam jak wyraz końcowy WORD2.

W konstrukcji OPERATORS podano słownik (zbiór wyrazów, przez które można przechodzić). Wyrazy z zadeklarowanego słownika są traktowane jako operatory.

```
(*MULTICOMP* '(
DEFINE WAY (WORD1 WORD2)  ;
INITIAL    (QS (LIST WORD1))  ;
OPERATORS  '(KOT LAS LOS BAS BAT LOT CIS LIS SOS POT)  ;
WDO        (AND (EQN (NOTEQ (CAR (QS N)) OP) 1)
                (NOT (MEMBER OP (QS N))))   ;
NEW        (QS (CONS OP (QS N)))
           (G  (LENGTH (QS N)))
           (H  (NOTEQ OP WORD2))   ;
SOLUTION   (EQ (CAR (QS N)) WORD2)   ;
STRATEGY   ORDERED-SEARCH  ;
FINALLY    (REVERSE (QS N))  ;
END  ;
COMMENT *** FUNKCJA NOTEQ OBLICZA LICZBE LITER, KTORYMI
            ROZNIA SIE DWA ATOMY BEDACE JEJ ARGUMENTAMI *** ;
DEF NOTEQ (A1 A2)  ;
   (PROG (LICZ K X1 X2)
      (SETQ LICZ 0)
      (SETQ K 0)
      (SETQ X1 (EXPLODE A1))
      (SETQ X2 (EXPLODE A2))
 L    (SETQ K (ADD1 K))
      (COND ((NOT (EQ (NTH X1 K) (NTH X2 K)))
             (SETQ LICZ (ADD1 LICZ))))
      (COND ((EQN K (LENGTH X1)) (RETURN LICZ))
            (T (GO L)))
      )   ;
END  ;
EXECUTE (WAY 'KOT 'CIS)  ;
))

OPTIMAL SOLUTION

(KOT LOT LOS LIS CIS)
```

Wydruk 2

Za wybór operatorów stosowanych do danego węzła jest odpowiedzialna funkcja WDO. Operator spełnia warunek zadeklarowany w WDO, gdy różni się od ostatniego wygenerowanego wyrazu tylko jedną literą. Warunek ten jest sprawdzany przy użyciu funkcji NOTEQ, która udostępnia listę niezgodnych liter w wyrazach będących jej argumentami.

Obrana strategia jest strategią uporządkowanego wyboru. Jako funkcję $\hat{h}$ przyjęto liczbę liter, którymi różni się ostatni wyraz znalezionego dotychczas ciągu od wyrazu końcowego. W pierwszej kolejności są więc rozwijane węzły, w których ostatni wyraz ciągu jest jak najbardziej zbliżony do wyrazu WORD2, gdyż rozwinięcie tych węzłów rokuje największe nadzieje na szybkie znalezienie rozwiązania.

Funkcja NEW opisuje sposób tworzenia nowego węzła:
— QS nowego węzła otrzymuje się przez dołączenie operatora na początek QS poprzednika, np. następnikami stanu (LOT KOT) są stany (LOS LOT KOT i (POT LOT KOT),
— cena dojścia do danego węzła, przechowywana we współrzędnej G, jest równa długości aktualnego ciągu, obliczanej za pomocą funkcji Lispu LENGTH,
— współrzędna H przyjmuje wartość równą liczbie liter, którymi wyraz aktualny różni się od wyrazu końcowego.

Jako wynik działania programu została udostępniona lista zawierająca QS węzła będącego rozwiązaniem. Kolejność wyrazów na tej liście została odwrócona za pomocą funkcji FINALLY. Funkcja FINALLY służy do przetwarzania węzła końcowego przed zapamiętaniem go na liście rozwiązań. Funkcji tej można na przykład użyć do zredagowania wydruku.

PERMUTACJE ZBIORU

W postaci drzewa można przedstawić wiele zadań z kombinatoryki, np. problem komiwojażera, problem upakowania plecaka, znajdowanie permutacji i kombinacji zbioru. Na wydruku 3 przedstawiono program w Multicompie wypisujący wszystkie permutacje zadanego zbioru.

```
(*MULTICOMP* '(
DEFINE PERMUT (ZBIOR)  ;
INITIAL    (GS (SETQ PZO ZBIOR))  ;
NEW        (QS  (CONS OP (QS N)))  ;
WDO        (NOT (MEMBER OP (QS N)))  ;
SOLUTION   (EQN (LENGTH (QS N)) (LENGTH ZBIOR))  ;
END  ;
EXECUTE (PERMUT '(DOROTA MA KOTA))  ;
))

OPTIMAL SOLUTION

(KOTA MA DOROTA)
(MA KOTA DOROTA)
(KOTA DOROTA MA)
(DOROTA KOTA MA)
(MA DOROTA KOTA)
(DOROTA MA KOTA)
```

Wydruk 3

W powyższym przykładzie zbiór, którego permutacje należy znaleźć, jest reprezentowany przez listę wejściową ZBIÓR. Wartość początkowa współrzędnej QS nie jest podana, przyjmuje się więc wartość standardową NIL, równoważną liście pustej. Początkowa wartość współrzędnej GS oraz lista PZO są równe liście wejściowej.

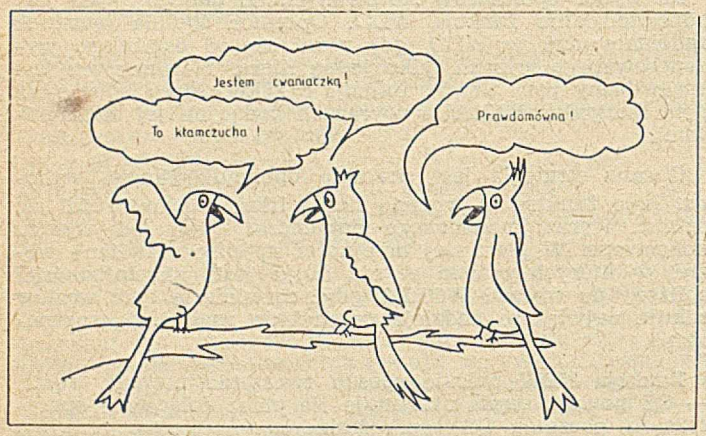
Operatorami są elementy listy ZBIÓR. Tworzenie nowego węzła polega na dostawieniu nowego operatora do QS poprzednika.

Aby uniknąć wielokrotnego występowania jakiegoś wyrazu na liście wynikowej, za pomocą parametru WDO sprawdza się, czy dany operator nie należy do listy QS.

Funkcja SOLUTION sprawdza, czy w QS węzła znajdują się już wszystkie elementy permutowanego zbioru. Przy założeniu, że wszystkie wyrazy na liście wejściowej były różne, wystarczy sprawdzić, czy długość listy QS w danym węźle jest równa długości listy ZBIÓR. Gdy warunek SOLUTION będzie spełniony, to permutacja odpowiadająca temu węzłowi zostanie zapamiętana i udostępniona wraz z innymi permutacjami jako wynik działania programu.

ZADANIE O PAPUGACH

Multicomp można wykorzystać do uporządkowanego przeglądania przypadków generowanych przez dane zadanie. Przykładem może być rozwiązanie poniższego zadania o papugach.



Ilustracja zadania o papugach

Na gałęzi siedziały w rzędzie trzy papugi. Prawdomówna, Kłamczucha i Cwaniaczka. Prawdomówna zawsze mówiła prawdę, Kłamczucha zawsze kłamała, a Cwaniaczka odpowiadała zależnie od humoru. Każdą z nich zapytano o imię papugi, która siedziała pośrodku. Papuga, siedząca z lewej strony, odpowiedziała, że to Kłamczucha, środkowa papuga odpowiedziała, że jest Cwaniaczką, natomiast skrajnie prawa odrzekła, że obok niej siedzi Prawdomówna (rys.). Należy ustalić imię każdej z papug.

Rozwiązanie polega na przejrzeniu wszystkich możliwości i sprawdzeniu, czy spełniają one warunki zadania. Odpowiedni program przedstawiono na wydruku 4. We współrzędnej QS znajduje się lista imion trzech papug w pewnej kolejności. Funkcja SOLUTION sprawdza, czy taka kolejność papug na gałęzi nie jest wewnętrznie sprzeczna (zdanie wypowiedziane przez Prawdomówną musi być prawdziwe, zdanie wypowiedziane przez Kłamczuchę — fałszywe). Lista wejściowa LIST1 zawiera wypowiedzi papug w kolejności, w jakiej zostały wygłoszone. Parametr INITIAL deklaruje listę imion w dowolnej kolejności. Operatory OP1 i OP2 służą do generowania kolejnych możliwości przyporządkowania imion papugom. Operator OP1 zamienia miejscami papugę środkową i prawą, operator OP2 zamienia prawą z lewą. Parametr FILTR dba o to, aby nie rozważać przypadków, które już przeanalizowano. Funkcja pomocnicza KTÓRY udostępnia położenie zadanego wyrazu (w naszym przykładzie — imienia papugi) na liście.

Po sprawdzeniu wszystkich możliwych przypadków jedyne możliwe rozwiązanie jest drukowane jako wynik działania programu.

```
(*MULTICOMP* '(
DEFINE PAPUGI (LIST1)  ;
INITIAL    (QS '(CWANIACZKA PRAWDOMOWNA KLAMCZUCHA))  ;
SOLUTION   (AND
             (EQUAL (NTH LIST1 (KTORY 'PRAWDOMOWNA (QS N)))
                    (NTH (QS N) 2))
             (NOT (EQUAL (NTH LIST1 (KTORY 'KLAMCZUCHA (QS N)))
                         (NTH (QS N) 2))))   ;
OPERATORS  '((OP1  (LIST)
                   (CONS (CAR LIST) (REVERSE (CDR LIST))))
             (OP2  (LIST)
                   (REVERSE LIST)))  ;
FILTR      (NOT (MEMBER (QS N) OPTIMAL))   ;
END  ;
DEF KTORY (ATOM  LIST)  ;
      (COND ((NULL LIST) NIL)
            ((EQUAL ATOM (CAR LIST)) 1)
            (T (ADD1 (KTORY ATOM (CDR LIST))))))  ;
END  ;
EXECUTE (PAPUGI '(KLAMCZUCHA CWANIACZKA PRAWDOMOWNA))  ;
))

OPTIMAL SOLUTION

(PRAWDOMOWNA KLAMCZUCHA CWANIACZKA)
```

Wydruk 4

LITERATURA

[1] Dyśko P.: Implementacja systemu Multicomp w języku Fortran. Praca dyplomowa, Politechnika Warszawska, 1978

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

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

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

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

Zakład Usług Informatycznych
R. Brykajło
ul. J. Bojki 6/22, 30-612 Kraków
tel. 34-50-93

poleca

ODRA-1305
- system redagowania i uruchamiania z monitorów lokalnych zadań George-2,
- interfej ODRA — AMSTRAD 6128,
- pomoc przy wdrożeniu systemów George 2 i 3.

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

ORAZ
- usługi w zakresie projektowania i programowania systemów przetwarzania danych.

EO/901/87

JAN BIELECKI
Instytut Informatyki
Politechnika Warszawska

Podstawy grafiki w języku Turbo Pascal (3)

Zarządzanie oknami i grafika żółwia

Wyprowadzanie na ekran nie musi dotyczyć całego ekranu. Zarówno w trybach tekstowych jak i graficznych jest możliwe wyodrębnienie prostokątnego okienka, które będzie traktowane tak, jakby stanowiło cały dostępny ekran, oczywiście o mniejszych rozmiarach.

ZARZĄDZANIE OKNAMI

Do definiowania okienka tekstowego służy procedura Window, a do definiowania okienka graficznego procedura GraphWindow. W rozdziale poświęconym grafice żółwia zostanie omówione okienko żółwia definiowane za pomocą procedury TurtleWindow. W każdej chwili może być zdefiniowane tylko jedno okienko. Można przyjąć, że bezpośrednio po ustanowieniu trybu tekstowego albo graficznego zostaje niejawnie zdefiniowane odpowiednio okienko tekstowe albo graficzne. Okienko takie zajmuje cały ekran. Zasługuje na podkreślenie, że zdefiniowanie okienka nie powoduje zmiany obrazu na ekranie. Dzięki temu wymaganie, aby w danej chwili było zdefiniowane tylko jedno okienko jest mało istotne, gdyż efekt wystąpienia wielu okienek łatwo uzyskać dokonując przełączeń między okienkami.

Przykład 1. Posługiwanie się okienkiem tekstowym

```
program JanAndEwa;
var
  Ch : char;
begin
  TextMode(BW40);
  Window(10,10,20,20);
  GotoXY(2,2);
  Write('Jan');
  Read(Kbd,Ch);
  Window(1,1,20,20);
  GotoXY(11,11);
  Write('Ewa');
  repeat until KeyPressed
end.
```

- Bezpośrednio po rozpoczęciu wykonywania programu zostaje niejawnie zdefiniowane okienko tekstowe domniemane (1,1,80,25).
- W ramach wykonania procedury TextMode(BW40) zostaje niejawnie zdefiniowane okienko tekstowe (1,1,40,25).
- Po wykonaniu procedury Window (10,10,20,20) zostaje zdefiniowane okienko (10,10,20,20).

Wykonanie przytoczonego programu powoduje pojawienie się napisu **Jan**, a po wprowadzeniu z klawiatury dowolnego znaku, zastąpienie trzech liter tego napisu trzema literami napisu **Ewa**.

Jeśli tekst umieszczony w okienku nie mieści się w wierszu okienka, to jest kontynuowany w wierszu następnym. Jeśli nie mieści się w ostatnim wierszu, to w celu zwolnienia miejsca dla wyprowadzanego tekstu, wszystkie wiersze okienka zostają przesunięte o jeden wiersz do góry. W okienku jest więc realizowane przeglądanie pionowe.

Przykład 2. Przeglądanie pionowe

```
program Scroll;
begin
  TextMode(BW40);
  Window(1,1,2,2);
  Write('Janek');
  repeat until KeyPressed
end.
```

- Wykonanie programu powoduje wyprowadzenie napisu Janek.
- Ponieważ szerokość okienka wynosi zaledwie 2 znaki, wyprowadzenie takiego napisu wymagałoby okienka co najmniej o 3 wierszach.
- W następstwie przeglądania, litery **Ja** zanikają. Na ekranie pozostają litery **ne**, a pod nimi litera **k**.

Do zdefiniowania okienka graficznego służy procedura GraphWindow. Wywołanie tej procedury ma w ogólnym wypadku postać

GraphWindow(xMin,yMin,xMax,yMax)

W zapisie tym xMin, yMin, xMax yMax są wyrażeniami typu integer. Wykonanie procedury GraphWindow powoduje zdefiniowanie okienka, którego lewy górny narożnik ma współrzędne pikselowe (xMin,yMin), a prawy dolny narożnik ma współrzędne pikselowe (xMax,yMax). Można przyjąć, że w chwili ustanowienia trybu graficznego średniej rozdzielczości zostaje niejawnie wykonana instrukcja

GraphWindow(0,0,319,199)

Po wykonaniu procedury GraphWindow wszystkie współrzędne są liczone względem okienka. W szczególności, punkt o współrzędnych (xMin,yMin), ma obecnie współrzędne (0,0). Wyjątek stanowią jedynie współrzędne w wywołaniu procedury GraphWindow. Jeśli po zdefiniowaniu okienka jest definiowane inne okienko, to współrzędne xMin, yMin, xMax i yMax są uznawane za współrzędne ekranowe.

Przykład 3. Definiowanie okienka graficznego

```
program Diagonal;
begin
  HiRes;
  HiResColor(Red);
  GraphWindow(20,20,50,50);
  Draw(0,0,30,0,1);
  Draw(30,0,30,30,1);
  Draw(30,30,0,30,1);
  Draw(0,30,0,0,1);
  Draw(0,0,30,30,1);
  repeat until KeyPressed;
  TextMode
end.
```

- Wykonanie programu powoduje zdefiniowanie okienka graficznego i wykreślenie jego przekątnej.

Jeśli po zdefiniowaniu okienka graficznego jest wykreślany obiekt, który nie mieści się w okienku, to punkty obiektu znajdujące się poza okienkiem nie są wykreślane. Ta właściwość okienka jest nazywana obcinaniem.

Przykład 4. Obcinanie na granicach okienka

```
program Clipping;
begin
  HiRes;
  GraphWindow(0,0,319,199);
  Draw(0,0,639,199,1);
  repeat until KeyPressed;
  TextMode
end.
```

- Wykonanie programu powoduje zdefiniowanie okienka graficznego w lewej części ekranu i wykreślenie odcinka łączącego punkt o współrzędnych (0,0) z punktem współrzędnych (639,199).
- Wyświetlona zostaje tylko połowa odcinka mieszcząca się w okienku. Pozostała część odcinka zostaje odcięta.
- Instrukcja Draw jest więc wykonywana tak, jakby miała postać:

Draw(0,0,319,99,1).

GRAFIKA ŻÓŁWIA

Idea grafiki żółwia (ang. turtle graphics) wiąże się z pojęciem „żółwia", który poruszając się po ekranie kreśli odcinki linii prostych. Ponieważ odcinki te mogą być dowolne krótkie i niemal dowolnie zorientowane, nic nie stoi na przeszkodzie, aby figury kreślone przez żółwia miały nie tylko postać łamanych, ale również postać dowolnych, aproksymowanych przez nie linii krzywych.

Wykonywanie wykresów przez żółwia jest możliwe w dowolnym trybie graficznym, ale wymaga włączenia do programu deklaracji znajdujących się w zbiorze Graph.p.

Na ekranie monitora żółw jest przedstawiany w postaci małego trójkąta, który może być dowolnie przesuwany i obracany. Bezpośrednio po aktywowaniu trybu graficznego trójkąt ten jest niewidoczny, ale po wykonaniu procedury ShowTurtle jest przywoływany na ekran. Ma on wówczas kolor nr 3 bieżącej palety barw.

Wykresy wykonywane przez żółwia są umieszczane w obszarze ekranu nazywanym okienkiem żółwia. Bezpośrednio po ustanowieniu trybu graficznego takim oknem jest cały ekran. Zmiana położenia okienka żółwia następuje po wykonaniu procedury TurtleWindow. Początek układu współrzędnych żółwia znajduje się zawsze w środku okienka. Jeśli okienkiem żółwia jest cały ekran, to zakres dostępnych współrzędnych wynika z następującego zestawienia:

- dla trybów średniej rozdzielczości

x = —159..160; y = —99..100

- dla trybów wysokiej rozdzielczości

x = —319..320; y = —99..100

Bezpośrednio po zdefiniowaniu okienka żółwia żółw znajduje się w punkcie o współrzędnych (0,0), jest skierowany do góry i niewidoczny.

Z żółwiem jest związane pióro, które może być opuszczone albo podniesione. Jeśli jest opuszczone, to podczas ruchu żółwia w obszarze, kreśli ono linię prostą. Opuszczenie pióra odbywa się za pomocą procedury PenDown, a podnoszenie za pomocą procedury PenUp. Bezpośrednio po zdefiniowaniu okienka żółwia pióro jest opuszczone. Przemieszczenie żółwia poza okienko czyni go niewidocznym. Widoczność żółwia można zapewnić sprowadzając go do obszaru okienka albo wykonując procedurę Wrap (zawiń). Po jej wykonaniu wszystkie współrzędne żółwia będą brane modulo rozmiary okienka, co m.in. spowoduje, że przekroczenie przez żółwia np. górnej krawędzi okienka, uczyni go widocznym w pobliżu krawędzi dolnej.

Liczba podprogramów umożliwiających realizowanie grafiki żółwia jest dość znaczna. Te, które zostały zrealizowane w implementacji języka Turbo Pascal dla IBM PC zostaną tu przytoczone w porządku alfabetycznym.

Procedura Back

Wywołanie: Back(Num)
Argumenty: Num jest wyrażeniem typu integer.
Wykonanie procedury Back powoduje przemieszczenie żółwia do tyłu o Num pikseli. Jeśli Num reprezentuje daną o wartości ujemnej, to żółw przemieszcza się do przodu. Jeśli pióro związane z żółwiem jest opuszczone, a koniec pióra dotyka punktu w obrębie okienka, to przemieszczanie się żółwia powoduje kreślenie linii.

Procedura ClearScreen

Wywołanie: ClearScreen
Argumenty: Procedura ClearScreen jest bezargumentowa.
Wykonanie procedury ClearScreen powoduje wyczyszczenie okienka i przemieszczenie żółwia do punktu o współrzędnych (0,0), znajdującego się w środku okienka. Po wykonaniu tej operacji żółw jest niewidoczny, a pióro związane z żółwiem opuszczone.

Procedura Forwd

Wywołanie: Forwd (Num)
Argumenty: Num jest wyrażeniem typu integer.
Wykonanie procedury Forwd powoduje przemieszczenie żółwia do przodu o Num pikseli. Jeśli Num reprezentuje daną o wartości ujemnej, to żółw przemieszcza się do tyłu. Jeśli pióro związane z żółwiem jest opuszczone, a koniec pióra dotyka punktu w obrębie okienka, to przemieszczanie się żółwia powoduje kreślenie linii.

Funkcja Heading

Wywołanie: Heading
Argumenty: Funkcja Heading jest bezargumentowa.
Rezultatem funkcji jest dana typu integer. Wartość tej danej określa kierunek, w którym jest zwrócony żółw. Kierunek jest określany z dokładnością do 1 stopnia. Wartość 0 oznacza kierunek „do góry", a kolejne wartości dodatnie, aż do 359 włącznie, oznaczają kierunki tworzone na skutek obrotu zgodnego z ruchem wskazówek zegara.

Procedura HideTurtle

Wywołanie: HideTurtle
Argumenty: Procedura HideTurtle jest bezargumentowa.
Wykonanie procedury HideTurtle powoduje, że żółw staje się niewidoczny. Niewidoczność nie pozbawia jednak żółwia innych właściwości, jak np. zdolności do przemieszczania się albo kreślenia linii za pomocą pióra.

Procedura Home

Wywołanie: Home
Argumenty: Procedura Home jest bezargumentowa.
Wykonanie procedury Home powoduje umieszczenie żółwia w punkcie o współrzędnych (0,0) i nadanie mu kierunku „do góry". Taka zmiana pozycji żółwia nie powoduje kreślenia linii nawet wtedy, gdy pióro związane z żółwiem jest opuszczone.

Procedura NoWrap

Wywołanie: NoWrap
Argumenty: Procedura NoWrap jest bezargumentowa.
Wykonanie procedury NoWrap powoduje, że przemieszczenie się żółwia poza okienko czyni go niewidocznym.

Procedura PenDown

Wywołanie: PenDown
Argumenty: Procedura PenDown jest bezargumentowa.
Wykonanie procedury PenDown powoduje opuszczenie pióra związanego z żółwiem. Spowoduje to, że przemieszczanie się żółwia w obrębie okienka będzie powodować kreślenie linii.

Procedura PenUp

Wywołanie: PenUp
Argumenty: Procedura PenUp jest bezargumentowa.
Wykonanie procedury PenUp powoduje podniesienie pióra związanego z żółwiem. Spowoduje to, że przemieszczanie się żółwia nie będzie powodować kreślenia linii.

Procedura SetHeading

Wywołanie: SetHeading(Num)
Argumenty: Num jest wyrażeniem typu integer.
Wykonanie procedury SetHeading powoduje zwrócenie żółwia w kierunku określonym przez Num. Kierunek jest określany z dokładnością do 1 stopnia. Wartość Num=0 oznacza kierunek „do góry", a kolejne wartości dodatnie, aż do Num=359 włącznie, oznaczają kierunki tworzone na skutek obrotu zgodnego z ruchem wskazówek zegara.

Procedura SetPenColor

Wywołanie: SetPenColor(Num)
Argumenty: Num jest wyrażeniem typu integer.
Wykonanie procedury SetPenColor powoduje wybranie koloru używanego do kreślenia linii. Wybierany kolor ma numer Num bieżącej palety barw. Jeśli Num=1, to wybór koloru odbywa się poprzez wektor barw.

Procedura SetPosition

Wywołanie: SetPosition(xCoord,yCoord)
Argumenty: xCoord i yCoord są wyrażeniami typu integer.
Wykonanie procedury SetPosition powoduje przemieszczenie żółwia do punktu o współrzędnych (xCoord,yCoord). Taka zmiana pozycji żółwia nie powoduje kreślenia linii ani zmiany kierunku, w którym żółw jest zwrócony. Z tego powodu wykonanie procedury SetPosition(0,0) nie jest równoważne wykonaniu procedury Home.

Procedura ShowTurtle

Wywołanie: ShowTurtle
Argumenty: Procedura ShowTurtle jest bezargumentowa.
Wykonanie procedury ShowTurtle powoduje, że żółw staje się widoczny.

Procedura TurnLeft

Wywołanie: TurnLeft(Num)
Argumenty: Num jest wyrażeniem typu integer.
Wykonanie procedury TurnLeft powoduje obrót żółwia w lewo o liczbę stopni określoną przez Num. Jeśli Num reprezentuje daną wartość ujemnej, to obrót następuje w prawo.

Procedura TurnRight

Wywołanie: TurnRight (Num)
Argumenty: Num jest wyrażeniem typu integer.
Wykonanie procedury TurnRight powoduje obrót żółwia w prawo o liczbę stopni określoną przez Num. Jeśli Num reprezentuje daną o wartości ujemnej, to obrót następuje w lewo.

Procedura TurtleWindow

Wywołanie: TurtleWindow(xCoord,yCoord,Hor,Ver)
Argumenty: xCoord, yCoord, Hor i Ver są wyrażeniami typu integer.
Wykonanie procedury TurtleWindow powoduje zdefiniowanie okienka żółwia. Środek tego okienka znajduje się w punkcie o współrzędnych (xCoord,yCoord), a okienko ma rozmiar poziomy Hor pikseli i rozmiar pionowy Ver pikseli. Bezpośrednio po ustanowieniu trybu graficznego okienkiem żółwia jest cały ekran. Wynika to z niejawnego wykonania procedury TurtleWindow zgodnie z następującym zestawieniem:

— dla trybów średniej rozdzielczości

TurtleWindow(159,99,320,200)

— dla trybu wysokiej rozdzielczości

TurtleWindow(319,99,640,200)

Funkcja TurtleThere

Wywołanie: TurtleThere
Argumenty: Funkcja TurtleThere jest bezargumentowa.
Rezultatem funkcji TurtleThere jest dana typu boolean. Wartość tej danej określa prawdziwość zdania „żółw jest widoczny". W szczególności, bezpośrednio po wykonaniu procedury HideTurtle rezultatem funkcji TurtleThere jest dana o wartości false.

Procedura Wrap

Wywołanie: Wrap
Argumenty: Procedura Wrap jest bezargumentowa.
Wykonanie procedury Wrap powoduje, że następujące po wykonaniu tej procedury przemieszczenie się żółwia poza krawędź okienka sprowadzi go do tego punktu okienka, który znajduje się na krawędzi przeciwległej. W szczególności, przemieszczanie się żółwia w kierunku pionowym odbywa się tak, jakby poziome krawędzie były ze sobą połączone, a okienko było zwinięte w walec.

Funkcja xCor

Wywołanie: xCor
Argumenty: Funkcja xCor jest bezargumentowa.
Rezultatem funkcji xCor jest dana typu integer. Wartością tej danej jest odcięta bieżącej pozycji żółwia. W szczególności, bezpośrednio po wykonaniu procedury Home rezultatem funkcji xCor jest dana o wartości 0.

Funkcja yCor

Wywołanie: yCor
Argumenty: Funkcja yCor jest bezargumentowa.
Rezultatem funkcji yCor jest dana typu integer. Wartością tej danej jest rzędna bieżącej pozycji żółwia. W szczególności, bezpośrednio po wykonaniu procedury Home, rezultatem funkcji yCor jest dana o wartości 0.

Przykład 5. Wykreślanie wielokątów o zadanej liczbie boków

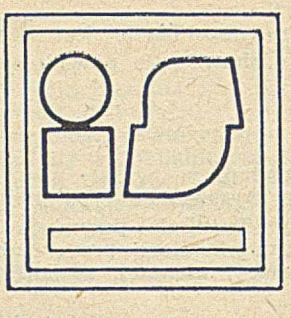
```
program Polygon;
{$i Graph.p}
var
  aStep,cnt,Num : byte;
begin
  Read(Num);
  while Num > 2 do begin
    GraphColorMode;
    PenUp;
    Back(50);
    PenDown;
    TurnLeft(90);
    aStep := trunc(360 / Num);
    for cnt := 1 to Num do begin
      Forwd(10);
      TurnLeft(aStep)
    end;
    repeat until KeyPressed;
    TextMode;
    Read(Num)
  end
end.
```

• Wykonanie programu powoduje wykreślenie wielokątów o żądanej liczbie boków.

Przykład 6. Wyświetlanie fraktala o kształcie płatka śniegu

```
program SnowFlake;
{$i Graph.p}
var
  Angle : integer;
  Length,Level,Side : integer;
procedure DrawSide(Length, Level : integer);
begin
  if Level = 0 then
    Forwd(Length)
  else begin
    Length := Length div 2;
    DrawSide(Length, Level - 1);
    TurnLeft(60);
    DrawSide(Length, Level - 1);
    TurnLeft(-120);
    DrawSide(Length, Level - 1);
    TurnLeft(60);
    DrawSide(Length, Level - 1)
  end
end;
begin
  GraphColorMode;
  SetPosition(-50,-80);
  Length := 2 * 2 * 2 * 2 * 2;
  Level := 4;
  for Side := 1 to 3 do begin
    DrawSide(Length, Level);
    TurnLeft(-120)
  end;
  repeat until KeyPressed;
  TextMode
end.
```

INTERSOFT



Sp. z o.o.

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

o f e r u j e:

WYBRANE POZYCJE DOKUMENTACJI W JĘZYKU POLSKIM DO KOMPUTERÓW IBM

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

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

WYBRANE POZYCJE DOKUMENTACJI W JĘZYKU POLSKIM DO KOMPUTERÓW AMSTRAD

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

Uwaga: pozycje oznaczone "x" są w przygotowaniu.

EO/792/87

STANISŁAWA OSSOWSKA
Gdańsk

Wordstar 3.30 — zasady działania i sposób użytkowania (2)

W drugiej części artykułu omówiono dalsze polecenia Wordstara, zawarte w menu głównym, menu samouczka, menu dla bloków oraz w menu szybkim.

MENU GŁÓWNE

Menu główne (ang. main menu) jest udostępniane po otwarciu pliku tekstowego lub nietekstowego. Znika wówczas z ekranu napis ⟨⟨⟨OPENING MENU⟩⟩⟩, a jego miejsce zajmuje napis ⟨⟨⟨MAIN MENU⟩⟩⟩ (rys. 1). Towarzyszą temu także inne zmiany na ekranie — w wierszu statusu pojawia się nazwa zadeklarowanego pliku oraz informacje o aktualnej pozycji kursora, ewentualnie inne informacje.

```
A:NAUKA  PAGE 1 LINE 1 COL 01                    INSERT ON
                ( ( (    M A I N   M E N U      ) ) )
  --Cursor Movement--  : -Delete-  :  -Miscellaneous-  :  -Other  Menus-
^S char left ^D char right :^G  char  : ^I Tab   ^B Reform : (from Main only)
^A word left ^F word right :DEL chr lf: ^V INSERT ON/OFF   :^J Help  ^K Block
^E line  up  ^X line down  :^T word rt: ^L Find/Replce again:^Q Quick ^P Print
    --Scrolling--          :^Y  line   :RETURN End paragraph:^O Onscreen
^Z line down ^W line up    :           : ^N Insert a RETURN :
^C screen up ^R screen down:           : ^U Stop a command  :
L----!----!----!----!----!----!----!----!----!----!----!----!-------R
```

Rys. 1. Polecenia zawarte w menu głównym

Menu głównemu odpowiada tryb pracy, w którym można redagować, tzn. tworzyć nowy lub zmieniać treść istniejącego, zadeklarowanego pliku. Pisanie rozpoczyna się od miejsca, w którym aktualnie znajduje się kursor. Zainicjowanie realizacji określonego polecenia występującego w menu głównym wymaga jednoczesnego naciśnięcia klawisza CTRL, oznaczonego symbolicznie w artykule grotem strzałki ( ), oraz klawisza z wybraną literą.

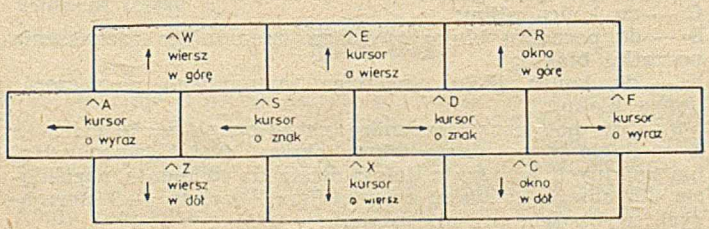
• **Przesuwanie kursora** (ang. cursor movement):

^S — przesunięcie kursora o jeden znak w lewo;
^D — przesunięcie kursora o jeden znak w prawo;
^A — przesunięcie kursora o jeden wyraz w lewo;
^F — przesunięcie kursora o jeden wyraz w prawo;
^E — przesunięcie kursora o jeden wiersz w górę;
^X — przesunięcie kursora o jeden wiersz w dół.

Przy przesuwaniu kursora tekst pozostaje bez zmian, jeśli nie wychodzi się poza granice okna tekstowego.

• **Przeglądanie tekstu** (ang. scrolling):

^Z — przemieszczanie tekstu o jeden wiersz w górę i udostępnianie nowego wiersza z dołu; kursor nie zmienia swego położenia w tekście;
^W — przemieszczanie tekstu o jeden wiersz w dół i udostępnianie wiersza z góry; kursor nie zmienia swego położenia w tekście;
^C — przemieszczenie tekstu o jedno okno tekstowe (minus dwa wiersze) w górę i udostępnianie następnego okna (z dołu); kursor pozostaje w tej samej linii na ekranie;
^V — przemieszczanie tekstu o jedno okno tekstowe (minus dwa wiersze) w dół i udostępnianie poprzedniego okna (z góry); kursor pozostaje w tej samej lini na ekranie.



Rys. 2. Sterowanie przesuwaniem kursora i przeglądaniem tekstu

Podstawowe polecenia zawarte w menu głównym, służące do przesuwania kursora po ekranie oraz do przeglądania tekstu, zostały tak zaprogramowane, że odpowiadające im pozycje poszczególnych klawiszy są zgodne z geometrycznymi ruchami kursora (rys. 2). Przy bardziej odległych przesunięciach kursora bądź przeglądaniu tekstu korzysta się z poleceń dostępnych w menu szybkim (ang. quick menu).

• **Usuwanie** (ang. delete):

^G — usuwanie znaku znajdującego się w miejscu wskazywanym przez kursor, przy czym kursor nie zmienia swej pozycji;
DEL — usuwanie znaku znajdującego się na lewo od kursora; kursor jest przesuwany o znak w lewo;
jscuwkn idmieięd
^T — usuwanie słowa na prawo od kursora wraz ze znakiem będącym w miejscu kursora; usuwane są także znaki RETURN sygnalizowane w skrajnej prawej kolumnie; kursor nie zmienia swego położenia;
^Y — usuwanie wiersza, w którym znajduje się kursor; tekst przesuwa się o jeden wiersz w górę; kursor nie zmienia swej pozycji.

• **Polecenia różnorodne** (ang. miscellaneous):

^I — przesuwanie kursora na następną pozycję tabulacji zgodnie z zadanym rodzajem tabulacji (zob. ^OV);
^B — przeredagowanie akapitu zgodnie z aktualnie obowiązującym formatem dokumentu, bądź dosuwanie do marginesu po korektach; wymaga to uprzedniego ustawienia kursora na początku przeredagowywanego akapitu lub nie dosuniętego wiersza;
V — włączanie (wyłączanie) możliwości dopisywania tekstu; jeżeli przełącznik jest włączony (w wierszu statusu wyświetla się INSERT ON), to w miejscu wskazanym przez kursor można dopisywać dowolny tekst; tekst dotychczasowy przesuwa się w prawo, a po przekroczeniu prawego marginesu znika z ekranu (co jest zaznaczone w skrajnie prawej kolumnie znakiem +), pozostając jednak w pamięci komputera; stanie się znowu widoczny po przeredagowaniu akapitu poleceniem ^B;
^L — powtórzenie ostatnio wykonywanej operacji wyszukiwania lub zmiany napisu (zob. menu szybkie);
RETURN — oznaczanie końca akapitu (w skrajnie prawej kolumnie pojawia się znak <) i przesunięcie kursora do nowego wiersza; w różnych rodzajach klawiatur funkcja ta może być inicjowana różnymi klawiszami, np. CR;
^N — wprowadzanie pustego wiersza przed wiersz, w którym znajduje się kursor; polecenie to jest przydatne zwłaszcza przy wprowadzaniu pustych wierszy przed tekstem (wymaga wówczas ustawienia kursora w kolumnie 1);
^U — przerwanie wykonywania ostatnio wprowadzonego polecenia.

• **Inne listy menu** (ang. other menus)

Ta część menu występuje we wszystkich rodzajach list menu z wyjątkiem menu otwierającego. Do wyspecyfikowanych tutaj list menu można przejść tylko z menu głównego. W pozostałych rodzajach menu część ta pełni jedynie funkcję informacyjną:

^J — przejście do menu samouczka;
^K — przejście do menu bloków;
^Q — przejście do menu szybkiego;
^P — przejście do menu sterowania wydrukiem;
^O — przejście do menu dla ekranu.

Zainicjowanie wybranego rodzaju menu spośród wymienionych wyżej wymaga jednoczesnego naciśnięcia klawisza CTRL i klawisza z odpowiednią literą. Zainicjowanie

dowolnego polecenia spośród wyspecyfikowanych w innych listach menu wymaga jedynie naciśnięcia klawisza z odpowiednią literą, odpowiadającą temu poleceniu. Po wykonaniu polecenia z innych list menu sterowanie wraca do menu głównego.

MENU SAMOUCZKA

Menu samouczka (ang. help menu) można wywołać na dowolnym etapie redagowania tekstu z poziomu menu głównego, przez jednoczesne naciśnięcie klawisza CTRL i J. Zadanie samouczka polega na wyjaśnieniu problemów, określonych w jego menu i wybranych przez użytkownika stosownie do potrzeb (rys. 3):

H — wyświetlenie informacji o czterech poziomach samouczka (stopniach objaśnień udzielanych użytkownikowi);
B — opis sposobu ponownego formatowania akapitu;
F — opis znaczenia znaków występujących w skrajnie prawej kolumnie;
D — opis sposobu działania poleceń z kropką (ang. dot commands);
I — opis sposobu inicjowania funkcji realizowanych z wyłączeniem poleceń z kropką;
S — opis wiersza statusu;
R — opis poleceń służących do ustawiania marginesów i tabulacji;
M — opis zasad i sposobów ustawiania marginesów i tabulatorów;
P — opis sposobu ustawiania znaczników miejsc oraz przesuwania kursora do ustawionych znaczników;
V — opis sposobu przesyłania bloków.

```
^J        A:NAUKA  PAGE 1 LINE 1 COL 01                    INSERT ON
                 < < <      H E L P   M E N U         > > >
                                           : --Other  Menus--
 H  Display & set the help level  : S  Status line     : (from Main only)
 B  Paragraph reform (CONTROL-B)  : R  Ruler line      : ^J  Help  ^K  Block
 F  Flags in right-most column    : M  Margins & Tabs  : ^Q  Quick ^P  Print
 D  Dot commands, print controls  : P  Place markers   : ^O  Onscreen
 I  Index of commands             : V  Moving text     :  Space Bar returns
                                  :                    :  you to Main Menu.
L----!----!----!----!----!----!----!----!----!----!----!--------R
```

Rys. 3. Polecenia zawarte w menu samouczka

W prawej części menu samouczka znajdują się informacje przypominające użytkownikowi o dostępnych rodzajach menu (ang. other menus), do których można przejść tylko przez menu główne.

MENU DLA BLOKÓW

Menu dla bloków (ang. block menu) służy do podejmowania działań związanych z blokami (przez blok rozumie się dowolnie wybrany przez użytkownika fragment tekstu oznaczony odpowiednio znacznikiem początku i końca bloku) bądź plikami, takich jak kopiowanie, usuwanie, przemieszczanie itp. Zawiera pięć grup poleceń przedstawionych na rysunku 4.

```
^K        A:NAUKA  PAGE 1 LINE 1 COL 01                   INSERT ON
                 < < <      B L O C K  M E N U       > > >
-Saving  Files- : -Block Operations- : -File  Operations- :  -Other  Menus-
 S Save & resume : B  Begin  K  End   : R  Read  P  Print  :  (from Main only)
 D Save--done    : H  Hide / Display  : O  Copy  E  Rename : ^J Help   ^K Block
 X Save & exit   : C  Copy   Y  Delete: J  Delete          : ^Q Quick ^P Print
 Q Abandon file  : V  Move   W  Write : -Disk  Operations- : ^O Onscreen
 -Place Markers- : N  Column  now OFF :L Change logged disk: Space Bar returns
 0-9 set/hide 0-9:                    :F Directory now OFF : you to Main Menu.
L----!----!----!----!----!----!----!----!----!----!----!--------R
```

Rys. 4. Polecenia zawarte w menu dla bloków

● **Zapisywanie plików** (ang. saving files)

Polega to na zapisaniu na dysk pliku rezydującego w pamięci mikrokomputera i zainicjowaniu pożądanego stanu systemu:

S — zapisanie redagowanego pliku na dysk i powrót do menu głównego (na ekranie wyświetla się początek pliku);
D — zapisanie redagowanego pliku na dysk (na ekranie wyświetla się menu otwierające);
X — zapisanie redagowanego pliku na dysk i powrót do systemu operacyjnego;
Q — zaniechanie zapisu redagowanego pliku na dysk, co powoduje przejście do menu otwierającego bez utrwalenia wprowadzonych zmian w redagowanym pliku.

Podczas zapisywania kolejnych wersji pliku na dysk zachowywane są jedo dwie wersje: ostatnia zgodnie z podaną nazwą oraz przedostatnia z rozszerzeniem BAK. Wcześniejsze wersje są usuwane.

● **Znakowanie miejsc** (ang. place markers):

0-9 — wstawianie znaczników miejsc do pliku w miejsce aktualnie ustawionego kursora; wyświetlany znacznik jest zaznaczany jako <n>, gdzie n jest numerem znacznika; znaczniki są ustawiane tylko w czasie redagowania pliku na ekranie (nie są utrwalane na dysku);
0-9 — wyłączanie wyświetlania znaczników przez ponowne wprowadzenie tego samego polecenia przy tym samym ustawieniu kursora.

● **Operacje blokowe** (ang. block operations):

B — oznaczenie początku bloku w miejscu aktualnie ustawionego kursora (jeśli nie polecono inaczej, to w miejscu początku bloku pojawia się znak <B>);
K — oznaczenie końca bloku w miejscu aktualnie ustawionego kursora (jeśli nie polecono inaczej, to w miejscu końca bloku pojawia się znak <K> i oznaczony blok jest rozjaśniany; taki blok można kopiować, kasować, przemieszczać w tekście, przenosić do innego pliku itp.);
H — przełącznik umożliwiający wyświetlanie, bądź wyłączanie wyświetlania znaczników początku i końca bloku;
C — kopiowanie oznaczonego bloku w miejsce zaczynające się od aktualnej pozycji kursora;
Y — usuwanie oznaczonego bloku;
V — przesyłanie oznaczonego bloku w miejsce zaczynające się od aktualnej pozycji kursora;
W — zapisanie oznaczonego bloku do innego pliku na dysku (już istniejącego lub tworzonego w trakcie działania tego polecenia);
N — odmienna interpretacja bloku (tzn. od znacznika początku do znacznika końca bloku według kolumn, a nie według wierszy, co jest przydatne zwłaszcza przy tablicach).

● **Operowanie plikami** (ang. file operations):

R — odczyt pliku z dysku i jego zapis do aktualnie redagowanego pliku, poczynając od miejsca ustawienia kursorem;
P — inicjowanie drukowania pliku z dysku;
O — kopiowanie pliku z dysku na dysk;
E — zmiana nazwy pliku (jest to przydatne zwłaszcza wówczas, gdy zachodzi potrzeba korzystania z poprzedniej wersji pliku opatrzonej rozszerzeniem BAK, gdyż operacje na tych plikach nie są możliwe bezpośrednio);
J — usuwanie pliku z dysku.

● **Operacje dyskowe** (ang. disk operations):

L — zmiana napędu dyskowego;
F — włączanie bądź wyłączanie wyświetlania skorowidza.

Przedstawione powyżej dwie grupy poleceń dotyczących plików oraz dysków pokrywają się w znacznej mierze z poleceniami zawartymi w menu otwierającym.

MENU SZYBKIE

Menu szybkie (ang. quick menu) służy do przesuwania kursora bądź przeglądania tekstu w większych porcjach informacji, niż pozwala na to menu główne. Jest przydatne zwłaszcza przy wprowadzaniu poprawek. Zawiera cztery grupy poleceń (rys. 5).

```
^Q        A:NAUKA  PAGE 1 LINE 1 COL 01                   INSERT ON
                 < < <      Q U I C K   M E N U     > > >
   ---Cursor Movement---     : -Delete- :  --Miscellaneous-- : --Other  Menus--
 S left side  D right side   :Y line  rt:F Find text in file : (from Main only)
 E top scrn   X bottom scrn  :DEL lin lf:A Find & Replace    :^J Help  ^K Block
 R top file   C end file               :L Find Misspelling  :^Q Quick ^P Print
 B top block  K end block              :Q Repeat command or :^O Onscreen
 0-9 marker   Z down  W up             :  key  until  space :Space Bar returns
 P previous   V last Find or Block     :  bar  or other key :you to Main Menu.
L----!----!----!----!----!----!----!----!----!----!----!--------R
```

Rys. 5. Polecenia zawarte w menu szybkim

● **Przesuwanie kursora** (ang. cursor movement):

S — do początku wiersza;
D — do końca wiersza (za ostatnie słowo);
E — do pierwszego wiersza tekstu na ekranie;
X — do ostatniego wiersza tekstu na ekranie;
R — do początku pliku;
C — do końca pliku;
B — do początku bloku (wymaga uprzedniego zaznaczenia początku bloku);
K — do końca bloku (wymaga uprzedniego zaznaczenia końca bloku);
0-9 — do podanego znacznika ustawionego w tekście;
Z — ciągłe przeglądanie tekstu oknem w dół (wprowadzanie kolejnych wierszy z dołu); działa do momentu naciśnięcia dowolnego klawisza, z wyjątkiem klawiszy 1—9 sterujących szybkością przemieszczania;

dokończenie na str. 31

JAN BIELECKI
Instytut Informatyki
Politechnika Warszawska

Język C — programowanie operacji wejścia-wyjścia

Operacje wejścia-wyjścia dotyczą zbiorów danych albo urządzeń. W programach zarówno zbiory danych jak i urządzenia są reprezentowane przez pliki. Plik nie jest obiektem fizycznym, lecz stanowi tylko pewien aspekt, pewną logiczną interpretację obiektu fizycznego, jakim jest zbiór danych w pamięci masowej lub zbiór danych pochodzących z urządzenia znakowego, takiego jak np. klawiatura.

Pisząc program w języku wysokiego poziomu i koncentrując się na zakodowaniu algorytmu wygodnie jest posługiwać się abstrakcyjnym pojęciem pliku. Tym niemniej należy pamiętać, że zawarte w algorytmie operacje na plikach, będą podczas wykonywania programu realizowane jako operacje na zbiorach i urządzeniach. Z tego względu wykonywanie operacji na plikach musi być poprzedzone skojarzeniem pliku z ustalonym zbiorem danych albo urządzeniem.

W języku C do skojarzenia pliku z urządzeniem służy funkcja **fopen**, a do usunięcia tego skojarzenia funkcja **fclose**. Rezultatem funkcji **fopen** jest wskazanie pewnej danej typu (FILE), zależnego od implementacji i zdefiniowanego w zbiorze **stdio.h**. Argumentem funkcji **fclose** jest natomiast wskazanie udostępnione przez funkcję **fopen**.

Poza wskazaniami udostępnionymi za pomocą funkcji **fopen** zdefiniowano w języku C trzy dodatkowe wskazania reprezentowane przez nazwy: **stdin**, **stdout** i **stderr**. Każde z tych wskazań jest typu (FILE *) i jest zdefiniowane w zbiorze **stdio.h**. Ponieważ wskazania **stdin**, **stdout** i **stderr** są zwyczajowo nazywane plikami, odpowiednio: standardowym plikiem wejściowym, standardowym plikiem wyjściowym i standardowym plikiem do sygnalizowania błędów także i każda zmienna, której przypisano wskazania plikowe będzie tu nazywana plikiem. Dzięki takiemu potraktowaniu, obiekty plikowe języka C będą w znacznym stopniu przypominać analogiczne obiekty innych języków programowania, takich jak np. PL/I i Turbo Pascal.

Funkcja fopen

Rozszerzony nagłówek: **FILE *fopen(name, mode)**
char xname,*mode;

Wykonanie funkcji **fopen** powoduje skojarzenie zbioru danych o nazwie określonej przez **name**, z unikalnym wskazaniem plikowym udostępnionym jako rezultat funkcji **fopen**, a tym samym stworzenie warunków do przetwarzania wspomnianego zbioru w trybie mode. Jeśli mode reprezentuje łańcuch "r" to zbiór zostanie otwarty w trybie do wprowadzenia, a jeśli reprezentuje łańcuch "w", to zostanie otwarty w trybie do wyprowadzania. Jeśli zbiór otwarty w trybie do wyprowadzania już istnieje, to tuż przed otwarciem zostanie usunięty. Jeśli otwarcie zbioru okaże się niemożliwe, to rezultatem funkcji będzie wskazanie puste.

Funkcja fclose

Rozszerzony nagłówek: **fclose (file)**
FILE *file;

Wykonanie funkcji **fclose** powoduje zamknięcie pliku identyfikowanego przez wskazanie plikowe **file**. Większość operacji szeregowego przetwarzania plików może być zrealizowana za pomocą funkcji **getc**, **putc** i **ungetc**. Funkcje te oraz pokrewne im funkcje **getchar** i **putchar** są jednak niekiedy realizowane nie jako „prawdziwe" funkcje, lecz jako makrodefinicje zdefiniowane w zbiorze **stdio.h**. W większości programów nie ma to jednak istotnego znaczenia.

Funkcja getc

Rozszerzony nagłówek: **int getc(file)**
FILE *file;

Wykonanie funkcji **getc** powoduje wprowadzenie kolejnego znaku z pliku **file**. Rezultatem funkcji jest dana typu **(int)** o wartości równej kodowi wprowadzonego znaku. Jeśli tuż przed wykonaniem operacji **getc** plik znajdował się w pozycji końcowej to rezultatem funkcji jest dana o wartości **EOF**. **EOF** jest nazwa literału zdefiniowanego w zbiorze **stdio.h**. Wywołanie **getc(stdin)** może być w każdym kontekście zastąpione wywołaniem **getchar()**.

Funkcja putc

Rozszerzony nagłówek: **int putc(c,file)**
int c;
FILE *file;

Wykonanie funkcji **putc** powoduje wyprowadzenie do pliku **file** znaku o kodzie **c**. Rezultatem funkcji jest dana typu **(int)** o wartości kodu wyprowadzonego znaku. Wywołanie **putc(c,stdout)** może być w każdym kontekście zastąpione wywołaniem **putchar(c)**.

Funkcja ungetc

Rozszerzony nagłówek: **ungetc(c,file)**
int c;
FILE *file;

Wykonanie funkcji **ungetc** powoduje cofnięcie do pliku identyfikowanego przez **file**, znaku o kodzie **c**. Cofnięcie polega na tym, że najbliższe wywołanie funkcji **getc** dotyczące tego samego pliku, spowoduje udostępnienie cofniętego znaku. Jeśli do pliku cofnięto znak, ale go jeszcze nie udostępniono, to zabrania się wykonania funkcji **ungetc** dotyczącej tego pliku.

```
#include <stdio.h>

main()
{
  FILE *inp;
  int ch;

  inp = fopen("JANB.DOC", "r");

  if(inp){
    while((ch = getc(inp)) != EOF)
    putchar(ch);
    close(inp);
  } else
    putc('?',stderr);
}
```

Wydruk 1. Operacje na pliku

Użycie funkcji **fopen** i **fclose** przedstawiono na wydruku 1. Wykonanie przytoczonego programu powoduje wyprowadzenie na monitor zawartości zbioru tekstowego **JANB.DOC**. Wykonanie instrukcji:

inp = fopen(JANB.DOC","r");

powoduje otwarcie pliku, a wykonanie instrukcji cyklu powoduje przeniesienie zawartości zbioru do standardowego pliku wyjściowego, tj. na ekran monitora. Wykonywanie cyklu kończy się z chwilą wprowadzenia znaku o kodzie **EOF**. Transmisja tego znaku następuje w chwili rozpoznania końca zbioru. Ponieważ kod tego znaku jest różny od kodów wszystkich innych znaków, które mogą pochodzić z pliku, niezbędne było zadeklarowanie **ch** jako zmiennej typu **(int)**, a nie typu **(char)**. W wypadku, gdy podczas otwierania pliku zostanie udostępnione wskazanie puste, nastąpi wykonanie alternatywy **else** instrukcji **if** i do pliku **stderr** zostanie wyprowadzony znak **'?'**.

Znacznie częściej niż funkcje **getc i putc** są wykorzystywane w programach funkcje **fprintf** i **fscanf** oraz dotyczące standardowych plików **stdin** i **stdout** funkcje **printf** i **scanf.** Zostaną one tu przedstawione w znacznym uproszczeniu, ilustrującym zasady ich wykonywania.

Funkcja fprintf

Rozszerzony nagłówek: **fprinft(file,format,p1,p2, ... , pn)**
FILE *file;
char *format;

Wykonanie funkcji **fprinft** powoduje zinterpretowanie ciągu znaków wskazywanego przez format i wyprowadzenie do pliku **file** sekwencji znaków określonych łącznie przez format oraz parametry **p1, p2, ... , pn.**

Przyjmuje się, że ciąg wskazywany przez format składa się ze znaków oraz z wzorców konwersji. Wymaga się, aby każdemu wzorcowi konwersji odpowiadał jeden parametr **pj.** Zinterpretowanie znaku powoduje wyprowadzenie go do pliku **file.** Zinterpretowanie wzorca konwersji związanego z parametrem **pj** powoduje wyprowadzenie ciągu znaków mającego postać literału reprezentującego wartość parametru. Jeśli wyprowadzenie odbywa się do pliku **stdout,** to wywołanie funkcji może zostać przedstawione w postaci:

printf(format,p1,p2, ... ,pn)

Wzorzec konwersji ma postać **%c, %s** albo **%d.** Stosownie do przypadku następuje wyprowadzenie danej reprezentowanej przez parametr jako pojedynczego znaku, ciągu znaków albo liczby dziesiętnej (opatrzonej znakiem minus, jeśli jest ujemna).

```
#include <stdio.h>

main()
{
  printf("Jan%c is %d %s", 'B',44,"now");
}
```

Wydruk 2. Użycie funkcji printf

Użycie funkcji **printf** przedstawiono na wydruku 2. Wykonanie przytoczonego programu powoduje wyprowadzenie napisu:

JanB is 44 now

Funkcja fscanf

Rozszerzony nagłówek: **int fscanf(file,format,p1,p2, ... , pn)**
FILE *file;
char *format;

Wykonanie funkcji **fscanf** powoduje zinterpretowanie ciągu znaków wskazywanego przez format, wprowadzenie z pliku **file** sekwencji znaków określonych łącznie przez format oraz parametry **p1, p2, ... , pn,** potraktowanie ich jako zapisów wartości danych i przypisanie wartości tych danych zmiennym wskazywanym przez parametry.

Przyjmuje się, że ciąg wskazywany przez format składa się z odstępów (najdłuższych ciągów składających się wyłącznie ze spacji, tabulacji i znaków nowego wiersza), z innych znaków oraz z wzorców konwersji. Wymaga się, aby każdemu wzorcowi konwersji nie zawierającemu znaku * (gwiazdka) odpowiadał jeden parametr **pj.** Jeśli wprowadzenie odbywa się z pliku **stdin,** to wywołanie funkcji może zostać przedstawione w postaci:

scanf(format,p1,p2, ... , pn)

Zinterpretowanie odstępu powoduje zignorowanie odstępu w pliku **file.** Zinterpretowanie znaku powoduje pominięcie takiego samego znaku w pliku **file** (w wypadku stwierdzenia niezgodności następuje zakończenie wykonywania funkcji). Zinterpretowanie wzorca konwersji powoduje potraktowanie kolejnego pola pliku jako zapisu literału. Jeśli bezpośrednio po znaku **%** (procent) rozpoczynającym wzorzec konwersji występuje znak * (gwiazdka), to wartość tego literału zostanie przypisana kolejnej zmiennej wskazywanej przez parametr **pj.** W przeciwnym razie pole zostanie zignorowane.

Wzorzec konwersji ma postać **%c, %s** albo **%d.** Zinterpretowanie wzorca konwersji powoduje pominięcie w pliku **file** najbliższego odstępu, a następnie, stosownie do przypadku, uznanie za pole jednego znaku, ciągu znaków aż do odstępu albo ciągu znaków aż do znaku nie należącego do liczby. Interpretowanie wzorca konwersji odbywa się w taki sposób, że w wypadku wprowadzenia znaku nie należącego do pola zostaje on cofnięty do pliku.

Wykonanie funkcji **fscanf** kończy się z chwilą zinterpretowania całego ciągu **format,** z chwilą wprowadzenia znaku niepoprawnego albo uniemożliwiającego wykonanie konwersji, albo z chwilą rozpoznania końca pliku. Rezultatem funkcji jest dana typu **(int)** o wartości równej liczbie pomyślnie dokonanych konwersji albo dana o wartości **EOF** — w wypadku napotkania końca pliku.

```
#include <stdio.h>

main()
{
  char Initial;
  int Age;
  char Name[4];

  scanf("%s %c%*s is %d%*s", Name,
                             &Initial,
                             &Age);
  printf("%s%c is %d now", Name,
                           Initial,
                           Age);
}
```

Wydruk 3. Użycie funkcji scanf

Użycie funkcji **scanf** przedstawiono na wydruku 3. Wykonanie przytoczonego programu dla danych:

Jan Bielecki is 44 now_

spowoduje wyprowadzenie napisu:

JanB is 44 now

LITERATURA

[1] Bielecki J.: Język C — interpretacja standardu, WNT 1987
[2] Bielecki J.: Wprowadzenie do języka C, WNT (w druku)
[3] Bielecki J.: Oprogramowanie mikrokomputerów, WKiŁ (w druku)
[4] Kernighan B., Ritchie D.: The C programming language, Prentice Hall, 1978
[5] Aztec C86 for PC DOS, MS DOS and CP/M-86, Manx Software Systems, 1985.

Wprowadzenie do programowania w języku asemblera IBM PC (2)

dokończenie ze str. 22

Tymczasowy blok FCB porównuje się z blokiem FCB wykorzystywanym przez funkcje DOS do przeszukiwania skorowidza (FCBSEA). Jeżeli identyfikatory plików w tych blokach różnią się, to procedura wyszukuje w skorowidzu pierwszy plik, a w przeciwnym przypadku kolejny plik odpowiadający wyspecyfikowanemu identyfikatorowi. Jeżeli nie ma takiego pliku, to wynikowy blok FCB jest wypełniany spacjami.

Początkowa zawartość rejestru ES zostaje przywrócona na podstawie zawartości stosu. Umożliwia to przesłanie wyniku przeszukiwania do zmiennej ANSWER$ występującej w programie w Basicu. Nie dokonuje się żadnych kontroli poprawności. Napis odpowiedzi musi składać się co najwyżej z 12 znaków i nie może być literałem.

Na zakończenie przywrócona zostaje pierwotna zawartość rejestrów DS, CX, BX, AX, rejestru flagowego i wskaźnika bazy BP. Rozkaz powrotu powoduje usunięcie ze stosu 4 bajtów zawierających adresy parametrów.

Utworzenie wersji binarnej omawianego podprogramu wymaga wykonania następujących kroków:

MASM SEARCH;
LINK SEARCH;
EXE2BIN SEARCH SEARCH.BLO

Użycie podprogramu zilustrowano na wydruku 3. Instrukcja CLEAR jest wymagana w celu zarezerwowania pamięci, a sam podprogram musi zostać załadowany instrukcją BLOAD, 7 bajtów za początkiem zarezerwowanego obszaru ze względu na rozmiar prefiksu wykorzystywanego przez tę instrukcję. Należy też zwrócić uwagę, że pole do przechowywania napisu zarezerwowano używając wyrażenia SPACE$(12) zamiast literału.

Opracował GRABOWICZ
na podst. Byte, nr 11, 1981

Wprowadzenie do programowania w języku asemblera IBM PC (2)

W drugiej części artykułu omówiono zasady programowania, ilustrując je wywołaniami funkcji systemu operacyjnego PC-DOS oraz wykonywaniem operacji napisowych, a także — technikę pisania kodu, który może być wywoływany z programów w Basicu.

PRZYKŁADOWY PROGRAM

Na wydruku 1 przedstawiono źródłową wersję programu MAKEBAT.ASM, napisanego w języku asemblera. Jego wywołanie ma dwa parametry: identyfikator pliku wyjściowego i identyfikator pliku wyszukiwanego w bieżącym skorowidzu. Tworzony plik wyjściowy stanowi plik wsadowy (tj. makropolecenie systemowe), do którego jest zapisywana linia tekstu dla każdego pliku o identyfikatorze odpowiadającym drugiemu parametrowi polecenia. Każda linia przybiera postać:

%1 filename

gdzie przez filename rozumie się identyfikator pliku (złożony z nazwy stacji dysków, nazwy i rozszerzenia nazwy pliku).

Przykładowo, polecenie:

MAKEBAT ALLASM.BAT *.ASM

tworzy plik, który mógłby zawierać linie:

%1 MAKEBAT.ASM
%1 SEARCH.ASM

```
;======== STRUKTURY
;------FILE CONTROL BLOCK
FCB             STRUC
DRIVE_NUMBER    DB      ?
FILE_NAME       DB      8 DUP(?)
FILE_EXTENSION  DB      3 DUP(?)
CURRENT_BLOCK   DW      ?
REC_SIZE        DW      ?
FILE_SIZE_LO    DW      ?
FILE_SIZE_HI    DW      ?
FILE_DATE       DW      ?
                DB      10 DUP(?)
CURRENT_REC     DB      ?
RANDOM_REC_LO   DW      ?
RANDOM_REC_HI   DW      ?
FCB             ENDS
;======= PREFIKS SEGMENTU PROGRAMU
PSP-    SEGMENT AT 0
        INCLUDE PSP.INC
PSP     ENDS
;======= SEGMENT STOSU
SSEG    SEGMENT PARA STACK 'STCK'
        DW      256 DUP(?)
SSEG    ENDS
;======= SEGMENT DANYCH
DSEG    SEGMENT 'DATA'
;------- FILE CONTROL BLOCKS
FCBOUT  FCB     <>
FCBRES  FCB     <>
FCBSEA  FCB     <>
;------- KOMUNIKATY O BLEDACH
MSGCRE  DB      'UNABLE TO CREATE THE OTPUT FILE.',00DH,00AH,'$'
MSGWR1  DB      'ERROR DURING WRITE TO THE OUTPUT FILE.',00DH,00AH,'$'
;------- WZORZEC NA LINIE WYJSCIOWA
OUTLINE DB      '%1 ?:????????.???',00DH,00AH
;------- PODWOJNE SLOWO NA POWROT
RETURN  LABEL   DWORD
RETIP   DW      0
RETCS   DW      ?
;------- FUNKCJA POSZUKIWANIA, PIERWSZE WYWOLANIE 011H, NASTEPNE 012H
VARFUN  DB      011H
DSEG    ENDS
;======= SEGMENT KODU
CSEG    SEGMENT 'CODE'
;******* PROCEDURY
        ASSUME CS:CSEG,DS:DSEG,ES:DSEG,SS:SSEG
TRAIL   PROC    NEAR
TRAILL  LABEL   NEAR
        CMP     BYTE PTR ES:[DI][-1],'
        JNE     TRAILX
        DEC     DI
        JMP     TRAILL
TRAILX  LABEL   NEAR
        RET
TRAIL   ENDP
;******* PROGRAM GLOWNY
        ASSUME  CS:CSEG,DS:PSP,ES:PSP,SS:SSEG
CONDSEG DW      DSEG
;------- ENTRY POINT
IP      LABEL   NEAR
;******* PRZETWARZANIE PSP I CZYNNOSCI WSTEPNE
;------- ES=DSEG
        MOV     ES,CONDSEG
        ASSUME  ES:DSEG
;------- UTWORZENIE PODWOJNEGO SLOWA DLA KONCA PROGRAMU
        MOV     RETCS,DS
;------- POBRANIE NUMERU STACJI, NAZWY I ROZSZERZENIA DLA PLIKU WYJSCIOWEGO
        MOV     CX,(SIZE DRIVE_NUMBER)+(SIZE FILE_NAME)+(SIZE FILE_EXTENSION)
        MOV     SI,OFFSET FORMATTED_AREA_1
        MOV     DI,OFFSET FCBOUT
        REP     MOVSB
;------ POBRANIE NUMERU STACJI, NAZWY I ROZSZERZENIA DLA KRYTERIUM POSZUKIWANIA
        MOV     CX,(SIZE DRIVE_NUMBER)+(SIZE FILE_NAME)+(SIZE FILE_EXTENSION)
        MOV     SI,OFFSET FORMATTED_AREA_2
        MOV     DI,OFFSET FCBSEA
        REP     MOVSB
;------- DS=DSEG
        MOV     DS,CONDSEG
        ASSUME  DS:DSEG
;******* UTWORZENIE PLIKU WYJSCIOWEGO
        MOV     AH,016H
        MOV     DX,OFFSET FCBOUT
        INT     021H
        OR      AL,AL
        JZ      CREATED
;------- NIEPOWODZENIE
        MOV     AH,009H
        MOV     DX,OFFSET MSGCRE
        INT     021H
        JMP     EXIT
;------- SUKCES
CREATED LABEL   NEAR
        MOV     FCBOUT.REC_SIZE,1
        MOV     FCBOUT.CURRENT_REC,0
        MOV     FCBOUT.RANDOM_REC_LO,0
        MOV     FCBOUT.RANDOM_REC_HI,0
;******* POSZUKIWANIE NAZW PLIKOW
NEXT    LABEL   NEAR
;------- USTAWIENIE ADRESU BUFORA DLA REZULTATU WYSZUKIWANIA
        MOV     AH,01AH
        MOV     DX,OFFSET FCBRES
        INT     021H
;------- WYKONANIE WYSZUKIWANIA
        MOV     AH,VARFUN
        MOV     DX,OFFSET FCBSEA
        INT     021H
        OR      AL,AL
        JNZ     DONE
;******* BUDOWA LINII WYNIKOWEJ NA PODSTAWIE REZULTATU
;------- '%1'
        MOV     DI,OFFSET OUTLINE
        MOV     AL,'%'
        STOSB
        MOV     AL,'1'
        STOSB
        MOV     AL,' '
        STOSB
;******* NAZWA PLIKU & ':'
;------- PRZESLANIE CALEJ NAZWY
        MOV     CX,(SIZE FILE_NAME)
        MOV     SI,OFFSET FCBRES.FILE_NAME
        REP     MOVSB
;------- USUNIECIE KONCOWYCH SPACJI
        CALL    TRAIL
;------- ':'
        MOV     AL,':'
        STOSB
;******* ROZSZERZENIE & CARRIAGE RETURN & LINE FEED
;------- PRZESLANIE CALEGO ROZSZERZENIA NAZWY
        MOV     CX,(SIZE FILE_EXTENSION)
        MOV     SI,OFFSET FCBRES.FILE_EXTENSION
        REP     MOVSB
;------- USUNIECIE KONCOWYCH SPACJI
        CALL    TRAIL
;------- CARRIAGE RETURN
        MOV     AL,00DH
        STOSB
;------- LINE FEED
        MOV     AL,00AH
        STOSB
;******* ZAPIS REKORDU WYJSCIOWEGO
;------- NADANIE WARTOSCI ADRESOWI BUFORA DLA PLIKU WYJSCIOWEGO
        MOV     AH,01AH
        MOV     DX,OFFSET OUTLINE
        INT     021H
;------- ZAPIS REKORDU
        MOV     AH,028H
        MOV     CX,DI
        SUB     CX,OFFSET OUTLINE
        MOV     DX,OFFSET FCBOUT
        INT     21H
        OR      AL,AL
        JZ      WRITTEN
;------- ZAPIS BLEDU
        MOV     AH,009H
        MOV     DX,OFFSET MSGWR1
        INT     021H
        JMP     DONE
;------- SUKCES
WRITTEN LABEL   NEAR
;------- USTAWIENIE VARFUN DLA KONTYNUOWANIA PRZESZUKIWANIA
        MOV     VARFUN,012H
        JMP     NEXT
;******* PRZESZUKIWANIE SKONCZONE
DONE    LABEL   NEAR
;------- ZAMKNIECIE PLIKU WYJSCIOWEGO
        MOV     AH,010H
        MOV     DX,OFFSET FCBOUT
        INT     021H
EXIT    LABEL   NEAR
;------- SKOK DO PODWOJNEGO SLOWA Z ADRESEM POWROTU
        JMP     RETURN
CSEG    ENDS
        END     IP
```

Wydruk 1. Poniższy program tworzy plik wsadowy i jest wywoływany z dwoma parametrami — nazwą pliku wyjściowego i nazwą pliku do wyszukania

Wtedy wykonanie polecenia ALLASM TYPE spowoduje wyświetlenie zawartości obu plików MAKEBAT.ASM i SEARCH.ASM.

STRUKTURA PROGRAMU

Język asemblera 8088 posiada dokładnie określone typy. Żadnego nazwanego obiektu, danej lub etykiety o typie nadanym w programie, nie można użyć niezgodnie z tym typem. Jeśli zmienna została zadeklarowana jako bajtowa, to asembler nie pozwoli na przesłanie wartości tej zmiennej do rejestru 16-bitowego. Jeśli etykieta została zadeklarowana jako nazwa procedury bliskiej, to asembler nie pozwoli na dalekie wywołanie tej procedury. Przez deklarowanie danych przed ich użyciem ułatwia się asemblerowi dokonywanie odpowiednich kontroli.

Program MAKEBAT rozpoczyna się od definicji struktury bloku kontrolnego pliku FCB. Jej poszczególne pola odpowiadają pozycjom bloku FCB. Definicja ta umożliwia łatwy dostęp do poszczególnych elementów bloku. Rozpoczyna się dyrektywą STRUC, a kończy dyrektywą ENDS. Dyrektywy DB (ang. define byte) i DW (ang. define word) definiują obiekty typu bajtowego i słowowego, a DUP określa powtarzanie wartości. Znak „?" oznacza brak nadanej wartości początkowej.

Informacje podane w linii polecenia MAKEBAT są przechowywane w prefiksie segmentu programu PSP. Wszystkie segmenty rozpoczynają się dyrektywą SEGMENT, a kończą dyrektywą ENDS. Dyrektywa INCLUDE powoduje, że w czasie tłumaczenia treść programu jest uzupełniana o dodatkowe linie tekstu wczytane z pliku PSP.INC (por. część 1 artykułu). Zadeklarowanie segmentu PSP z wyrażeniem AT 0 oznacza, że program łączący nie rezerwuje pamięci na ten segment w module wynikowym z rozszerzeniem nazwy EXE.

Dyrektywa SSEG SEGMENT deklaruje rozmiar stosu programu. Jeśli nie będzie definicji segmentu stosu, to program łączący wygeneruje komunikat o błędzie, mający charakter ostrzeżenia, i sam utworzy odpowiedni segment stosu. Zwykle nie powoduje to żadnych negatywnych skutków.

W segmencie DSEG zadeklarowane są wszystkie zmienne. Niektóre z nich, jak np. MSGCRE, mają nadane wartości początkowe. W deklaracji zmiennych FCBOUT, FCBRES i FCBSEA wykorzystuje się wcześniejszą definicję FCB. Pusty ciąg znaków w nawiasach kątowych oznacza, że wszystkie pola w strukturze mają takie wartości początkowe, jak określono w definicji FCB. Zmienne zadeklarowano w kolejności alfabetycznej, aby łatwiej można było odszukać ich definicję w treści programu.

Pierwszą dyrektywą w segmencie CSEG jest ASSUME. Przekazuje ona asemblerowi informacje, że poszczególne rejestry segmentowe wskazują odpowiednie segmenty logiczne. Pozwala to na wygenerowanie przebicia prefiksu segmentu, gdy zajdzie taka potrzeba.

W programie występuje tylko jedna procedura o nazwie TRAIL. Jest ona procedurą bliską. Jej działanie polega na zmniejszaniu zawartości rejestru indeksowego DI tak długo, dopóki bajt poprzedzający bajt wskazywany przez DI zawiera spację. Program główny wykorzystuje tę procedurę do usuwania końcowych spacji z linii, która ma być zapisana w pliku wyjściowym. Należy zwrócić uwagę na operator przebijania typu (ang. type override) BYTE PTR w rozkazie porównującym zawartość bajtu ze spacją. Jego wystąpienie jest niezbędne, ponieważ argument rozkazu ma typ anonimowy (tzn. odwołania dotyczą tylko rejestrów i stałych). Bez tego operatora nie byłby znany typ argumentu. Na podstawie typu zadeklarowanego po nazwie procedury asembler generuje odpowiedni rozkaz powrotu.

Po procedurze TRAIL występuje program główny. Dyrektywa ASSUME przed kodem programu głównego odzwierciedla warunki na początku wykonywania programu z pliku typu EXE. Etykieta IP, występująca także w końcowej dyrektywie END IP wskazuje, gdzie rozpoczyna się wykonywanie programu.

Pierwszym wykonywanym rozkazem jest:

MOV ES,CONDSEG

Zmienna CONDSEG została zadeklarowana w segmencie CSEG, dlatego asembler domyślnie generuje dla tego rozkazu przebicie prefiksu segmentu określające rejestr segmentowy jako CS. Gdyby nie wystąpiła wcześniej dyrektywa ASSUME, to rozkaz ten musiałby mieć postać:

MOV ES,CS:CONDSEG

Tę samą operację koduje się za pomocą pary rozkazów:

MOV AX,DSEG
MOV ES,AX

ale pierwsza z tych metod nie wymaga użycia dodatkowego rejestru AX. Stałe nie mogą być przesyłane bezpośrednio do rejestrów segmentowych, ponieważ rejestry te są umieszczone w jednostce sprzężenia z magistralą BIU. Bezpośrednio po omówionym rozkazie występuje dyrektywa ASSUME informująca asembler, że od tej chwili rejestr segmentu dodatkowego ES jest związany z logicznym segmentem danych DSEG.

OPEROWANIE NAPISAMI

Identyfikatory plików podane w linii polecenia są przesyłane z prefiksu segmentu programu PSP do bloków FCB w segmencie danych DSEG za pomocą rozkazu REP MOVSB (ang. move string byte); REP oznacza, że rozkaz MOVSB będzie wykonywany tyle razy, ile wynosi wartość zapamiętana w rejestrze CX. Rejestr indeksu źródłowego SI i rejestr indeksu wynikowego DI wskazują początek napisu do przesłania i obszar, dokąd ten napis ma być przesłany. Ponieważ nie występuje przebicie prefiksu segmentu, to ciąg bajtów począwszy od adresu DS:SI będzie przesyłany do obszaru zaczynającego się od adresu ES:DI. Zawartość rejestru CX jest równa sumie długości trzech pól w bloku FCB dotyczących identyfikatora pliku (tj. 12). Słowo OFFSET w rozkazach przesyłania wartości do rejestrów indeksu źródłowego SI i indeksu przeznaczenia DI oznacza przesłanie wyrównania (tj. adresu) zmiennych w segmencie kodu, a nie wartości tych zmiennych. Po przesłaniu identyfikatorów nie ma już potrzeby, aby rejestr DS wskazywał prefiks PSP, dlatego też zostaje mu nadana nowa wartość wskazująca segment DSEG, a asembler zostaje o tym poinformowany kolejną dyrektywą ASSUME.

FUNKCJE DOS

Aby wykonać funkcję DOS, numer funkcji umieszcza się w rejestrze AH i wykonuje rozkaz przerwania INT 21H. Dodatkowe informacje przekazuje się w pozostałych rejestrach, różnych w zależności od funkcji. Pierwszą wywołaną funkcją jest 16H, służąca do utworzenia pliku wyjściowego. Przed jej wywołaniem, adres DS:DX musi wskazywać na blok FCB. Jeśli funkcja została wykonana poprawnie, to rejestr AL zawiera 0, a w przeciwnym przypadku — kod błędu.

Jeżeli wystąpił błąd, to za pomocą funkcji 09H wyświetla się komunikat o błędzie. Adres DS:DX wyznacza początek wyświetlanego napisu, zakończonego znakiem dolara „$". Jeżeli błąd nie wystąpił, to w programie następuje nadanie wartości polom bloku FCB, które nie zostały zainicjowane przez funkcję tworzenia (lub otwierania) pliku. Przykładowo, rozmiar rekordu wyjściowego zainicjowano na 1, choć wartość domyślna wynosi 128.

Funkcja 1AH nadaje wartość adresowi bufora pliku DTA (ang. disk transfer address), który określa para rejestrów DS i DX. Bufor stanowi obszar pamięci, w którym przechowywane są dane zapisywane na dysk lub odczytane z dysku. W buforze tym umieszcza się także wyniki przeszukiwania skorowidza. Do wyszukiwania plików o wyspecyfikowanym identyfikatorze służą dwie funkcje 11H i 12H. Funkcja 11H wyszukuje pierwszą pozycję skorowidza odpowiadającą wycpecyfikowanemu identyfikatorowi, a funkcja 12H następne. Zmienna VARFUN przybiera wartość 11H przy pierwszym wykonaniu pętli, a wartość 12H przy następnych.

W kolejnym fragmencie programu, na podstawie przeszukiwania skorowidza tworzona jest linia tekstu wynikowego, w której procedura TRAIL eliminuje zbędne końcowe spacje. Następnie nadaje się wartość adresowi bufora DTA i oblicza długość linii do zapisania w pliku wyjściowym. Linię tekstu zapisuje się do pliku za pomocą funkcji 28H. Przed jej wykonaniem adres DS:DX powinien wskazywać początek bufora, a rejestr CX zawierać liczbę rekordów, które mają być zapisane. W omawianym przykładzie liczba rekordów równa się długości linii, ponieważ rozmiar rekordu wynosi 1 bajt.

Program działa w pętli tak długo, aż wyszukane zostaną wszystkie pliki odpowiadające podanemu identyfikatorowi. Na zakończenie, zamykany jest plik wyjściowy za pomocą funkcji DOS o numerze 10H.

Tworzenie całych programów w języku asemblera jest zajęciem bardzo czasochłonnym. Bardziej praktyczne podejście polega na pisaniu programów w języku wysokiego poziomu, np. w Basicu, z których — w miarę potrzeby — wywołuje się podprogramy napisane w języku asemblera.

PRZYKŁADOWY PODPROGRAM

Na wydruku 2 przedstawiono przykładowy podprogram SEARCH.ASM, wywoływany instrukcją CALL z programu napisanego w Basicu. Podprogram ten przeszukuje skorowidz w celu sprawdzenia, czy występuje w nim plik określony przez drugi parametr wywołania. Dopuszcza się wystąpienie znaków „*" lub „?" pozwalających na określenie zmiennych identyfikatorów plików. Jeżeli podprogram znajdzie wyspecyfikowany plik, to konkretna nazwa i jej rozszerzenie zostaną udostępnione przez pierwszy parametr wywołania. Kolejne wywołania podprogramu z tym samym drugim parametrem powodują wyszukanie kolejnych konkretnych identyfikatorów plików.

Podprogram SEARCH.ASM wykorzystuje inną metodę relokacji niż podawane w podręcznikach Basica. Pozwala to użytkownikowi na umieszczenie podprogramu za pomocą instrukcji BLOAD w dowolnym miejscu pamięci. Nie korzysta się przy tym z instrukcji POKE do ładowania kodu na podstawie danych zawartych w treści instrukcji DATA. W metodzie tej można wyróżnić dwa kluczowe składniki: instrukcję BLOAD i procedurę ADJUST.

Na początku pliku zawierającego wynikową wersję podprogramu znajduje się siedem dodatkowych bajtów, wykorzystywanych przez instrukcję BLOAD. Ten plik z rozszerzeniem nazwy COM powstaje przez konwersję pliku typu EXE. Konwersja ta jest niezbędna, ponieważ pliki typu EXE zawierają dodatkowe informacje potrzebne do relokacji, natomiast plik typu COM zawiera kod w postaci stanowiącej dokładny obraz pamięci.

Procedura ADJUST dopasowuje segment kodu w taki sposób, że zachowane są adresy wyrównania asemblowane przy założeniu, że segment kodu wskazuje na CSEG.

Podprogram SEARCH rozpoczyna się od typowej definicji bloku FCB. SEARCHP jest strukturą odzwierciedlającą wygląd stosu po wywołaniu podprogramu SEARCH i ułożeniu na szczycie stosu zawartości wskaźnika bazy BP. Elementy tej struktury są wykorzystywane do pobierania adresów parametrów ze stosu.

Segment o nazwie BASIC reprezentuje interpreter Basica. W czasie pracy interpretera wszystkie rejestry segmentowe wskazują prefiks segmentu programu BASIC. Odzwierciedla to dyrektywa ASSUME.

Ponieważ wersja wynikowa podprogramu SEARCH.ASM ma być przekształcona na plik typu COM, musi on składać się tylko z jednego segmentu logicznego. Kod podprogramu będzie załadowany do wnętrza obszaru interpretera, dlatego też segment CSEG zagnieżdżono w segmencie BASIC.

```
;======= STRUKTURY
;------- BLOK KONTROLI PLIKU
FCB             STRUC
DRIVE_NUMBER    DB    ?
FILE_NAME       DB    8 DUP(?)
FILE_EXTENSION  DB    3 DUP(?)
CURRENT_BLOCK   DW    ?
REC_SIZE        DW    ?
FILE_SIZE_LO    DW    ?
FILE_SIZE_HI    DW    ?
FILE_DATE       DW    ?
                DB    10 DUP(?)
CURRENT_REC     DB    ?
RANDOM_REC_LO   DW    ?
RANDOM_REC_HI   DW    ?
FCB             ENDS
;======= STRUKTURA DLA PARAMETROW PRZESZUKIWANIA
SEARCHP  STRUC
         DW       ?        ;BP
         DD       ?        ;ADRES POWROTU
REQUEST  DW       ?
ANSWER   DW       ?
SEARCHP  ENDS
;======= SEGMENT KODU
;******* SEGMENT BASICA
BASIC    SEGMENT PARA
         ASSUME  CS:BASIC,DS:BASIC,ES:BASIC,SS:BASIC
;******* CSEG SEGMENT
CSEG     SEGMENT PARA
         ASSUME  CS:CSEG
;******* ZALADOWANIE INSTRUKCJA BLOAD INFORMACJI NAGLOWKA
         DB      0FDH        ;ZNACZNIK BLOAD
         DW      00000H      ;OFFSET
         DW      00000H      ;SEGMENT
         DW      FINISH-START ;DLUGOSC
START    EQU     $
;******* PUNKT WEJSCIA
         JMP     NEAR PTR SEARCH
;******* DANE
FCBRES   FCB     <>
FCBSEA   FCB     <>
FCBTMP   FCB     <>
VARFUN   DB      ?
;******* PROCEDURA NADAWANIA WARTOSCI REJESTROWI CS
;------- NADAJE WARTOSC OFFSET DLA INSTRUKCJI BLOAD
;------- UZYWA DWOCH INSTRUKCJI PUSH I DALEKIEGO POWROTU ABY
;-------       ZMIENIC CS:IP
ADJUST   PROC    NEAR
;------- ABY WSTAWIC AKTUALNY OFFSET DO AX UZYWA SIE SLEPEGO
;-------       WYWOLANIA CALL I INSTRUKCJI POP
         CALL    ADJUSTL
ADJUSTL  PROC    NEAR
ADJUSTL  ENDP
         POP     AX      ;AX = wyrownanie ADJUSTL wzgledem segm.
                         ;     segmentu interpretera BASIC
;------- WYLICZENIE ROZNICY MIEDZY POCZATKAMI SEGMENTOW BASIC I CSEG
         SUB     AX,OFFSET ADJUSTL   ; OFFSET ADJUSTL ma wartosc wyrownania
         MOV     BX,AX               ; wzgledem segmentu BASIC
;------- PRZEKSZTALCENIE NA PARAGRAFY I WYLICZONY CS W CX
         SHR     AX,1
         SHR     AX,1
         SHR     AX,1
         SHR     AX,1
         MOV     CX,CS
         ADD     CX,AX       ; CX zawiera adres segmentowy dla CSEG
;------- WYLICZONY OFFSET DLA POWROTU W AX
         POP     AX          ; AX = wyrownanie powrotu wzgledem seg. BASIC
         SUB     AX,BX       ; AX = wyrownanie powrotu wzgledem seg. CSEG
;------- WSTAWIENIE NA STOS ZAWARTOSCI CS:OFFSET I DALEKI POWROT
         PUSH    CX
         PUSH    AX
ADJUSTF  PROC    FAR
         RET
ADJUSTF  ENDP
ADJUST   ENDP
;******* WYSZUKIWANIE
;======= PROCEDURA WYSZUKIWANIA
SEARCH   PROC    FAR
;------- POBIERZ WSKAZNIK BAZY
         PUSH    BP
         MOV     BP,SP
;------- ZAPAMIETANIE ZAWARTOSCI REJESTROW
         PUSHF
         PUSH    AX
         PUSH    BX
         PUSH    CX
         CLD
;------- WYLICZENIE CS
         CALL    ADJUST
;------- NADANIE WARTOSCI ES=CS
         PUSH    CS
         POP     ES
         ASSUME  ES:CSEG
;------- KOPIOWANIE NAZWY PLIKU DO TYMCZASOWAGO FCB
         MOV     AH,029H
         MOV     AL,000H
         MOV     SI,REQUEST[BP]
         MOV     SI,[SI][1]
         MOV     DI,OFFSET FCBTMP
         INT     021H
;------- NADANIE WARTOSCI DS=CS
         PUSH    CS
         POP     DS
         ASSUME  DS:CSEG
;------- POROWNANIE FCBTMP Z FCBSEA
         MOV     CX,(SIZE DRIVE_NUMBER)+(SIZE FILE_NAME)+(SIZE FILE_EXTENSION)
         MOV     SI,OFFSET FCBTMP
         MOV     DI,OFFSET FCBSEA
         REP     CMPSB
         JZ      MORE
;------- NOWE ZADANIE FCB
         MOV     CX,(SIZE DRIVE_NUMBER)+(SIZE FILE_NAME)+(SIZE FILE_EXTENSION)
         MOV     SI,OFFSET FCBTMP
         MOV     DI,OFFSET FCBSEA
         REP     MOVSB
         MOV     VARFUN,011H
MORE     LABEL   NEAR
;------- WYSZUKIWANIE
         MOV     AH,01AH
         MOV     DX,OFFSET FCBRES
         INT     021H
         MOV     AH,VARFUN
         MOV     DX,OFFSET FCBSEA
         INT     021H
         MOV     VARFUN,012H
;------- TESTOWANIE SUKCESU
         OR      AL,AL
         JZ      SUCCESS
;------- BRAK SUKCESU
         MOV     VARFUN,011H
         MOV     AL,' '
         MOV     CX,(SIZE DRIVE_NUMBER)+(SIZE FILE_NAME)+(SIZE FILE_EXTENSION)
         MOV     DI,OFFSET FCBRES
         REP     STOSB
SUCCESS  LABEL   NEAR
;------- PRZYWROCENIE WARTOSCI ES
         PUSH    SS
         POP     ES
         ASSUME  ES:BASIC
;------- WYNIKOWA NAZWA PLIKU W ANSWER$
         MOV     CX,(SIZE DRIVE_NUMBER)+(SIZE FILE_NAME)+(SIZE FILE_EXTENSION)
         MOV     SI,OFFSET FCBRES
         MOV     DI,ANSWER[BP]
         MOV     DI,ES:[DI][1]
         REP     MOVSB
;------- PRZYWROCENIE WARTOSCI DS
         PUSH    SS
         POP     DS
         ASSUME  DS:BASIC
;------- PRZYWROCENIE ZAWARTOSCI REJESTROW
         POP     CX
         POP     BX
         POP     AX
         POPF
;------- PRZYWROCENIE ZAWARTOSCI BP I POWROT DO BASICA
         POP     BP
         RET     4
SEARCH   ENDP
;******* KONCOWE INFORMACJE INSRTUKCJI BLOAD
FINISH   EQU     $
         DB      01AH
CSEG     ENDS
BASIC    ENDS
         END
```

Wydruk 2. Podprogram w języku asemblera dla przeszukiwania skorowidza z programu w języku Basic

Rozróżnienie między tymi segmentami jest ważne. Wszystkie wyrównania asemblowane w programie są liczone względem początku segmentu CSEG. W czasie wykonywania programu, między początkiem segmentu BASIC a CSEG znajduje się nieznana liczba bajtów.

Pierwsze 7 bajtów danych w segmencie CSEG odpowiada formatowi pliku wykorzystywanego przez instrukcję BLOAD. Pierwszym rozkazem jest rozkaz skoku do etykiety SEARCH będącej punktem wejścia podprogramu. Skok ten jest możliwy, ponieważ adres wyrównania punktu wejścia jest zawsze znany. W segmencie CSEG umieszczone są także definicje danych ze względu na ograniczenie długości podprogramu do jednego segmentu.

Procedura ADJUST oblicza różnicę wyrównania dla etykiety ADJUSTL raz względem początku segmentu BASIC, a za drugim razem względem początku segmentu CSEG. Obliczona różnica pozwala zmienić zawartość rejestru CS i wskazać na CSEG. Dzięki dopasowaniu zawartości rejestru CS, wyrównania zmiennych zdefiniowanych w segmencie CSEG są poprawne — bez tego dopasowania w czasie wykonywania podprogramu wyznaczałyby adresy względem segmentu BASIC.

Procedura ADJUST dokonuje bliskiego wywołania procedury ADJUSTL, aby pobrać adres wyrównania ADJUSTL względem segmentu BASIC. Adres wyrównania tej etykiety względem adresu początku segmentu CSEG jest znany, tak więc można obliczyć różnicę obu wyrównań. Wartość tę wykorzystuje się do utworzenia nowej zawartości pary CS:IP dla rozkazu dalekiego powrotu w programie ADJUSTF, gdzie rejestr segmentu kodu wskazuje na CSEG. Jeżeli w podprogramie nie występowałyby odwołania do danych zdefiniowanych w segmencie CSEG, to procedura ADJUST byłaby zbędna.

PROCEDURA SEARCH

Pierwszymi dwoma rozkazami procedury SEARCH są:

PUSH BP
MOV BP, SP

Służą one do zachowania bieżącej zawartości rejestru BP i wpisania do niego wartości wskaźnika bloku parametrów (ang. frame pointer). Adresy parametrów są układane na stosie przez interpreter Basica przed wywołaniem procedury SEARCH. Następnie wywoływana jest procedura ADJUST oraz wykonywany rozkaz CLD (ang. clear direction flag — zerowanie bitu flagowego D), którego skutkiem jest zmiana kierunku przetwarzania napisów na zgodny z wzrastającymi adresami.

```
20 REM
1000 REM ZAREZERWOWANIE PAMIECI
1010 CLEAR,&H8000
1020 REM ZALADOWANIE POD ADRES PARAGRAPH+7
1030 SEARCH%=&H8000+7
1040 BLOAD "SEARCH.BLO",SEARCH%
1050 REM ZADANY CIAG ZNAKOW I TERMINATOR
1060 REM SYMBOL STACJI DYSKOW MUSI BYC PODANY JAWNIE
1070 REQUEST$="A:*.ASM"+CHR$(13)
1080 REM ZAREZERWOWANIE OBSZARU NA ODPOWIEDZ
1090 ANSWER$=SPACE$(12)
1100 PRINT "WYWOLANIE SEARCH"
1110 CALL SEARCH%(ANSWER$,REQUEST$)
1120 REM JESLI ODPOWIEDZIA SA SPACJE TO ZAKONCZ
1130 IF ANSWER$=SPACE$(12) THEN 1200
1140 REM PRZEKSZTALC NUMER STACJI DYSKOW Z 0-3 NA A-D
1150 DRIVE$=CHR$(ASC(LEFT$(ANSWER$,1))+ASC("@"))
1160 FILENAME$=MID$(ANSWER$,2,8)
1170 EXTENSION$=MID$(ANSWER$,10,3)
1180 PRINT DRIVE$+":"+FILENAME$+"."+EXTENSION$
1190 GOTO 1110
1200 END
```

Wydruk 3. Program w języku Basic SEARCH.BAS

Dopasowana zawartość rejestru segmentu kodu zostaje przesłana do rejestru ES, tak aby indeks wynikowy DI wskazywał na dane w segemencie CSEG. Rejestr SI nadal wskazuje na dane w segmencie BASIC. Następnie procedura SEARCH, przy użyciu struktury SEARCHP i wskaźnika bloku parametrów, pobiera ze stosu adres 3-bajtowego deskryptora napisu. Pierwszy bajt deskryptora zawiera długość napisu, a drugi i trzeci bajt adres napisu (wyrównanie) w segmencie BASIC. Adres ten służy funkcjom DOS do utworzenia bloku FCB dla pliku. Procedura nie kontroluje poprawności poszczególnych kroków.

W dalszej części, dopasowaną zawartość rejestru CS przesyła się do rejestru segmentu danych DS, aby wszystkie adresy odnosiły się do zmiennych w segmencie CSEG.

dokończenie na str. 18

Zakładowy Ośrodek Informatyki i Organizacji w Bielskiej Fabryce Maszyn Włókienniczych „BEFAMA" Bielsko-Biała, ul. Pstrowskiego 13

posiada do upłynnienia:

1. Sterownik EDS-8 ICL 2802/0 — 2 szt.
2. Transporter EDS-8 ICL 2802/3 — 7 szt.
3. Transporter 8 MB BASF 6133 — 2 szt.

Ponadto oferujemy zestaw części zamiennych zawierających między innymi:

1. 78 pakietów 23 rodzajów do sterownika 2802/0
2. Pełny zestaw zasilaczy do sterownika
3. 24 głowice (nowe) — do 2802/3
4. Pakiety do zasilaczy do transportera 2802/3
5. Komplet pakietów do 2802/3
6. Silnik główny napędu 2802/3
7. Aktuator hydrauliczny — 2 szt.
8. Pompa hydrauliczna itp.

Ceny umowne

Zgłoszenia przyjmuje i informacji udziela:

Zakładowy Ośrodek Informatyki
i Organizacji „BEFAMA"
43-300 Bielsko-Biała, ul. Pstrowskiego 13
mgr Ryszard Markowicz
tel. 215-75 i 230-61 w. 15—60

EO/722/87

Przykład wielozadaniowości w Adzie

Tradycyjny sposób rozwiązywania zagadnień typu a(1)*b(1)+... jest sekwencyjny. Oznacza to, że poszczególne składniki oblicza się po kolei. Jeżeli działanie jest bardziej skomplikowane niż proste mnożenie, to można wykorzystać procedurę podobną do poniższej:

```
DO_OP(N1,N2 : in INTEGER;
      P : out INTEGER) is
end DO_OP;
```

W podejściu sekwencyjnym wykorzystuje się prostą pętlę:

```
for I in COMPONENT_RANGE loop
  DO_OP(A(I), B(I), C);
  TOTAL := TOTAL + C;
end loop;
```

Jeśli przyjąć, że na osi pionowej będzie odmierzany czas, to powyższe obliczenia można zobrazować następująco:

```
A(1)*B(1)
+A(2)*B(2)
+A(3)*B(3) itd.
```

Problem jest sformułowany i rozwiązany w sposób sekwencyjny.

Bieżące obliczenia są niezależne od siebie i dlatego można je przedstawić równolegle:

```
A(1)*B(1)          A(2)*B(2)          A(3)*B(3) itd.
+ (pierwsze dodawanie)
+ (drugie dodawanie itd.)
```

Wszystkie składniki oblicza się równolegle (w tym samym czasie) i później dodaje sekwencyjnie.

Ada pozwala programiście wykorzystać taką logikę nawet wtedy, gdy realizacja fizyczna może być inna — na przykład, jeśli dostępny jest tylko jeden procesor. Do tego służy jednostka programowa Ady zwana zadaniem. Potrzebne są tylko wielokrotne kopie procedury wykonujące obliczenia. Dlatego najstosowniejszy wybór stanowi typ zadaniowy (ang. task type).

```
task type OP_TASK is
  entry PASSTO(E,F : in INTEGER);
  entry RESULT(G : out INTEGER);
end OP_TASK;
DO_OP : array(1..COMPONENT_RANGE) of OP_TASK;
```

Dwa punkty spotkania (wejścia) wykorzystywane do komunikacji z zadaniami OP_TASK, służą do przekazywania wartości do OP_TASK (PASSTO) i udostępniania wyników obliczeń (RESULT). Program wykorzystujący obydwa wejścia może mieć następującą postać:

```
for I in COMPONENT_RANGE loop
  DO_OP(I).PASSTO(A(I),B(I));
end loop;
for I in COMPONENT_RANGE loop
  DO_OP.RESULT(P);
  TOTAL := TOTAL + P;
end loop;
```

Takie podejście równoległe, polegające na zawarciu zadań w pętli nie przyspiesza rozwiązania problemu. Zadania są tylko nieznacznie rozłożone w czasie, ponieważ użyto sekwencyjnej metody (pętla) przekazywania parametrów do typów zadaniowych. Po przekazaniu wszystkich parametrów wykonuje się druga pętla, w której po zakończeniu każdego zadania wyniki są sumowane.

Istnieje jednak jeszcze inne podejście do problemu. W powyższym przykładzie posłużono się tradycyjnym sposobem przekazywania parametrów do zadania, tj. przez spotkania. Rozważany problem nie ma krytycznych ograniczeń czasowych, gdyż parametry są wymieniane tylko na początku i końcu realizacji zadania. Istnieje inna metoda nieglobalnego przekazywania parametrów do zadania. Zadanie umieszcza się wewnątrz procedury, a komunikacja z nim odbywa się przez parametry formalne tej procedury:

```
procedure ANOTHER_DO_OP (N1, N2 : in INTEGER;
                         P : out INTEGER) is
  task DO_IT;
  task body DO_IT is
  begin -- DO_IT
    P := N1+N2;
  end DO_IT;
end ANOTHER_DO_OP;
```

Tym razem, każdorazowe wywołanie procedury powoduje rozpoczęcie zawartego w niej zadania. Wywołania można przeprowadzic w następującej pętli:

```
for I in COMPONENT_RANGE loop
  ANOTHER_DO_OP(A(I),B(I), C);
  TOTAL := TOTAL + C;
end loop;
```

Mimo wszystko nie stanowi to żadnego ulepszenia. Wywołanie procedury ANOTHER_DO_OP powoduje rozpoczęcie zawartych w niej zadań, ale procedura nie zostanie zakończona, dopóki nie zakończą się wszystkie jej zadania. Oznacza to, że pętla nie będzie kontynuowana, dopóki nie zakończy się instrukcja ANOTHER_DO_OP(A(1),B(1), C). Tym samym A(2)*B(2) nie zostanie obliczone równolegle z A(1)*B(1). W rzeczywistości nie zachodzi przetwarzanie równoległe, lecz liniowe. Trudność tę można wyeliminować za pomocą rekurencji. Do procedury ANOTHER_DO_OP dołącza się następujące instrukcje, zapisane schematycznie w języku PDL (ang. program design language):

```
begin
  if (JEST WIĘCEJ SKŁADOWYCH) then
     WEŹ NASTĘPNE SKŁADOWE;
     WYWOŁAJ ANOTHER_DO_OP DLA NOWYCH
                              SKŁADOWYCH;
  end if;
  CZEKAJ AŻ OSTATNIE ZADANIE SKOŃCZY
                              OBLICZENIA;
  TOTAL := TOTAL + C;
end ANOTHER_DO_OP;
```

Główna procedura wywołuje ANOTHER_DO_OP i przesyła do niej pierwsze składowe. W tym czasie ciało procedury generuje nowe składowe i wywołuje samo siebie. Nowa kopia rozpoczyna swoje zadanie, mnożąc dwie następne składowe i wywołuje siebie z trzecią z kolei parą składowych. Gdy nie ma już więcej składowych i ostatnie zadanie zakończy się, to proces zaczyna się odwracać.

```
with TEXT_IO; use TEXT_IO;
with CALENDAR; use CALENDAR;
procedure MAIN is
   package TIME_IO is new INTEGER_IO(DURATION);
   use TIME_IO;
      START, FINISH : TIME;
      AMOUNT : DURATION;
   package INT_IO is new INTEGER_IO(INTEGER);
   use INT_IO;
      type COLLECTION is array(1..5) of INTEGER;
      A: COLLECTION := (2,3,4,6,8);
      B: COLLECTION := (3,6,9,12,15);
      C: INTEGER;
      TOTAL: INTEGER;
      COMPONENT : INTEGER;

procedure ANOTHER_DO_OP(N1,N2,LAST_COMP : in INTEGER) is
   C1 :INTEGER;
   task DO_IT;
   task body DO_IT is
      begin
         C1 := N1+N2;
         delay 1.0;  -- przedłużenie obliczeń
   end DO_IT;

begin  -- ANOTHER_DO_OP
   if COMPONENT <= LAST_COMP then
      COMPONENT := COMPONENT + 1;
      ANOTHER_DO_OP(A(COMPONENT), B(COMPONENT), LAST_COMP);
   end if;
   if COMPONENT = LAST_COMP then
      while not DO_IT'TERMINATED loop null; end loop;
      TOTAL := TOTAL + C;
   end if;
end ANOTHER_DO_OP;

begin -- MAIN
   for DIM in 2..5 loop  -- pętla po wymiarze tablicy
      TOTAL := 0;
      COMPONENT := 1;
      START := clock;
      ANOTHER_DO_OP(A(1),B(1),DIM);
      FINISH := clock;
      AMOUNT := FINISH - START;
      PUT_LINE("RESULTS");
      PUT(" DIMENSION="); PUT(DIM); PUT_LINE(" ");
      PUT("TOTAL="); PUT(TOTAL); PUT_LINE(" ");
   end loop;     -- koniec pętli po wymiarze tablicy
end MAIN;
```

Wydruk 1. Program sekwencyjny

```
with TEXT_IO; use TEXT_IO;
with CALENDAR; use CALENDAR;
procedure MAIN is
   package TIME_IO is new INTEGER_IO(DURATION);
   use TIME_IO;
      START, FINISH : TIME;
      AMOUNT : DURATION;
   package INT_IO is new INTEGER_IO(INTEGER);
   use INT_IO;
      type COLLECTION is array(1..5) of INTEGER;
      A: COLLECTION := (2,3,4,6,8);
      B: COLLECTION := (3,6,9,12,15);
      C: INTEGER;
      TOTAL: INTEGER;

task type OP_TASK is
   entry PASSTO(E,F : in INTEGER);
   entry RESULT(G : out INTEGER);
end OP_TASK;

task body OP_TASK is
   FIRST, SECOND, ANSWER : INTEGER;
begin -- OP_TASK
   accept PASSTO(E,F : in INTEGER) do
      FIRST := E;
      SECOND := F;
   end PASSTO;
   ANSWER := FIRST*SECOND
   delay 1.0; -- przedłużenie obliczeń
   accept RESULT(G : out INTEGER) do
      G := ANSWER;
   end RESULT;
end OP_TASK;

-- następująca procedura sprawia, że DO_OP jest tablicą
-- dynamiczną; DIM jest rozmiarem tablicy i liczbą
-- równoległych zadań

procedure DOING_IT (DIM : in INTEGER) is
   TOTAL : INTEGER := 0;
   START, FINISH : TIME;
   AMOUNT : DURATION;
   DO_OP : array(1..DIM) of OP_TASK; -- wieloprocesowość
begin -- DOING_IT
   START := clock;
   for I in 1..DIM loop  -- odebranie wyniku
      DO_OP.PASSTO(A(I),B(I));
   end loop;

   for I in 1..DIM loop  -- odebranie wyniku
      DO_OP.RESULT(C);
      TOTAL := TOTAL + C;
   end loop;

   FINISH := clock;
   AMOUNT := FINISH - START;
   PUT_LINE("RESULTS");
   PUT(" DIMENSION="); PUT(DIM); PUT_LINE(" ");
   PUT(" TIME="); PUT(AMOUNT); PUT_LINE(" ");
end DOING_IT;


begin -- MAIN
   for N in 2..5 loop  -- pętla po wymiarze tablicy
      DOING_IT(N);
   end loop;
end MAIN;
```

Wydruk 2. Program wykorzystujący spotkania

W celu upewnienia się czy końcowy wynik obliczeń jest dostępny, niezbędna jest zwłoka. Otrzymana wartość jest przesyłana do procedury wywołującej i dodawana do

```
with TEXT_IO; use TEXT_IO;
with CALENDAR; use CALENDAR;
procedure MAIN is
   package TIME_IO is new INTEGER_IO(DURATION);
   use TIME_IO;
      START, FINISH : TIME;
      AMOUNT : DURATION;
   package INT_IO is new INTEGER_IO(INTEGER);
   use INT_IO;
      type COLLECTION is array(1..5) of INTEGER;
      A: COLLECTION := (2,3,4,6,8);
      B: COLLECTION := (3,6,9,12,15);
      C: INTEGER;
      TOTAL: INTEGER;

   procedure DO_OP(N1,N2 : in INTEGER;
                   P : out INTEGER) is

   begin
      P := N1*N2;
      delay 1.0; -- przedłużenie obliczeń
   end DO_OP;

begin -- MAIN
   for DIM in range 2..5 loop -- pętla po wymiarze
      TOTAL := 0;
      START := clock;
      for I in range 1..DIM loop -- składowe
         DO_OP(A(I),B(I),C);
         TOTAL := TOTAL + C;
      end loop; -- składowe
      FINISH := clock;
      AMOUNT := FINISH - START;
      PUT ("DIMENSION="); PUT(DIM); PUT_LINE(" ");
      PUT("TOTAL="); PUT(TOTAL); PUT_LINE(" ");

   end loop; -- koniec pętli po wymiarze tablicy
end MAIN;
```

Wydruk 3. Program rekurencyjny

TOTAL. Proces ten trwa aż do zakończenia wszystkich procedur.

Zaprezentowane trzy algorytmy są przykładami przetwarzanie tablic w Adzie i można je wykorzystać w testach porównawczych. Wszystkie trzy metody: sekwencja, z wykorzystaniem spotkań i rekurencyjna mają swoje wady i zalety. Metoda sekwencyjna nie wykorzystuje wszystkich możliwości komputera wieloprocesorowego. Algorytmy wykorzystujące spotkanie zadań i rekurencję są obarczone pewnym narzutem wewnętrznym.

W zależności od rozwiązywanego problemu jedna metoda może być szybsza od innych. Wydruki 1, 2 i 3 zawierają przykłady (z pewnymi modyfikacjami) wykorzystania każdej z metod. Aby można było wykryć efekt przekroczenia liczby procesorów w systemie wieloprocesorowym, rozmiar tablicy zadeklarowano jako zmienną. Z tego powodu poniższe programy są przeznaczone na system co najwyżej o czterech procesorach.

Oprac. M. KUC
na podst. Journal of Pascal, Ada and Modula-2
May—June 1984

WARUNKI PRENUMERATY

Wpłaty na prenumeratę przyjmowane są w terminach:

— do 10 listopada na każdy kwartał, I i II półrocze oraz cały rok następny,
— do 28 lutego na II, III i IV kwartał oraz II półrocze,
— do 31 maja na III i IV kwartał oraz II półrocze,
— do 31 sierpnia na IV kwartał.

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

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

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

Cena prenumeraty wg cennika na 1985 rok

kwartalna		półroczna		roczna	
normalna	ulgowa	normalna	ulgowa	normalna	ulgowa
600,—	150,—	1200,—	300,—	2400,—	600,—

Cena egzemplarza 200 zł (50 zł — cena ulgowa).

Słownik IBM PS/2

Wraz z wyprodukowaniem nowego wyrobu PS/2 firma IBM wprowadziła następujące odmiany nazw:

adapter — peripheral card

W języku polskim można więc spokojnie używać słowa adapter na oznaczenie sterownika dyskietek czy drukarki.

channel — bus

Może więc zamiast kłócić się o lepszy odpowiednik polski, szyna czy magistrala, wprowadzić słowo kanał?

facility — software + hardware

W języku polskim mamy nowy problem — może „sprzętogram"?

feature — hardware + peripheral card/disk drive

Oczekujemy propozycji polskiej nazwy.

fixed disk — hard disk

Można więc pozostać przy nazwie „dysk stały".

pel — pixel

Może właśnie w języku polskim użyć tego skrótu?

planar — motherboard

Sic!

W.I.

II Walny Zjazd Polskiego Towarzystwa Informatycznego

W dniu 23 maja br. w Warszawie odbył się II Walny Zjazd Delegatów PTI. Od pierwszego Zjazdu (1984 rok) liczba członków Towarzystwa wzrosła z 418 do 1323 członków zwyczajnych. Jest też 202 członków wspierających. Towarzystwo ma dwa oddziały (Dolnośląski i Małopolski), 13 kół terenowych i 15 sekcji. Podstawowym zadaniem Zjazdu było wybranie władz na nową kadencję oraz ocena dotychczasowej działalności Towarzystwa i ustalenie kierunków działania na przyszłość.

Na początku Zjazdu przedstawiono wyniki rozstrzygnięcia Konkursu im. Jerzego Trybulskiego na najlepszą pracę z zakresu zastosowań informatyki w gospodarce. Wyrażane obawy, że w polskich warunkach nie da się podać wymiernych efektów wdrażania informatyki, okazały się niesłuszne — wszystkie nagrodzone prace można było precyzyjnie wycenić. **Medale zostały przyznane zespołom z PZL Delta-Hydral, za informatyczny system zarządzania produkcją i ze spółdzielni Samopomoc Chłopska w Szczecinie, za zautomatyzowany system rachunkowości.** Warto podkreślić, że w obu nagrodzonych zespołach zdecydowaną większość stanowiły kobiety, a medale odbierali sami mężczyźni. Dopiero zespół wyróżniony dyplomem za system nadzoru przewozu kontenerów dla przedsiębiorstwa GAL pojawił się w pełnym koedukacyjnym składzie. Pamiątkowy medal wręczono również wdowie po Jerzym Trybulskim, co zgromadzeni delegaci przyjęli burzą oklasków.

Następnie ustępujący Prezes, prof. W. M. Turski, przedstawił główne tezy referatu Zarządu Głównego (pełna treść referatu znajduje się w Biuletynie PTI, nr 5, 1987), które streszczono poniżej. Towarzystwo rozwija się szybko, pomimo wyraźnie profesjonalnego charakteru, i nabiera statutowego kształtu organizacyjnego, choć jeszcze wyraźnie brak Oddziału Warszawskiego. Wszyscy chętni mają możliwość działalności w sekcjach, których liczba ciągle się zwiększa. Wielkim uznaniem cieszą się coroczne Szkoły Jesienne. Z dużym zainteresowaniem spotkały się konkursy — zarówno konkurs im. J. Trybulskiego, jak i coroczny konkurs na najlepsze prace magisterskie. Niestety ten ostatni jest całkowicie ignorowany przez Ministerstwo Nauki, a także przez potencjalnych pracodawców absolwentów.

Towarzystwo organizuje odczyty, seminaria i wykłady. Wydawany Biuletyn ma przyznany numer ISSN, a zatem jest już formalnym czasopismem. Opracowano i wydano Informator, zawierający szczegółowe dane o Towarzystwie, jego statut oraz skład i adresy władz centralnych i terenowych.

Towarzystwo opracowało i przekazało kilka stanowisk i opinii, związanych z rozwojem informatyki. Przyczyniło się do zniesienia indywidualnych zezwoleń oraz ceł na import mikrokomputerów. Doprowadziło to do powstania rynku konsumenta na komputery osobiste, choć ostatnio pojawiły się szkodliwe tendencje protekcjonistyczne, forsowane przez nieudolnych producentów krajowych. Historyczną zasługą Towarzystwa było zapobieżenie wyposażeniu szkół polskich w prymitywne komputery do gier, nie nadające się do celów dydaktycznych, co osiągnięto dzięki rzadkiej jedności profesjonalistów — informatyków. Wymagania, sformułowane przez Towarzystwo a przyjęte przez Ministerstwo Oświaty i Wychowania, spełnia Elwro 800 Junior, opracowany w Poznaniu i Wrocławiu. Oba powyższe przypadki były zresztą rzadkimi przykładami właściwej i odważnej reakcji władz — Urzędu Ceł i Ministerstwa Oświaty.

Towarzystwo zachowało swą niezależność, nie przyjmując żadnych dotacji i nie wiążąc się z żadną nadrzędną organizacją (np. NOT). W 1986 roku Towarzystwo uzyskało prawo przyznawania stopni specjalizacyjnych, pomimo oporu ze strony organizacji dotychczas przyznających takie stopnie. Powołano już odpowiednią Komisję Stowarzyszeniową. Środki na działalność Towarzystwa pochodzą przede wszystkim z usługowej działalności badawczo-szkoleniowej. W 1986 r. w realizacji 500 umów brało udział 3800 osób, z tego 30% stanowili członkowie PTI. Podejmuje się przedsięwzięcia ważne z gospodarczego lub społecznego punktu widzenia, pozwalające informatykom na zrealizowanie swoich zawodowych ambicji. Część wypracowanych środków przeznaczono na zakup mikrokomputerów dla szkolnych kół zainteresowań, a także na dotacje dla studenckich kół naukowych.

Część referatu stanowiły uwagi o stanie informatyki w Polsce. Nadal wśród władz politycznych, a także we władzach PAN i NOT panuje przeświadczenie, że coś takiego jak informatyka w ogóle samodzielnie nie istnieje. Uważa się ją za część elektroniki czy elektronizacji albo za dział matematyki. Świadczy o tym pominięcie informatyki przy okazji organizowania Kongresu Nauki Polskiej (PTI nie było zresztą zaproszone na Kongres, w przeciwieństwie do rozmaitych młodzieżowych klubów mikrokomputerowych). Utrzymuje się szkodliwy podział kompetencji. Nikt nie odpowiada za zastosowania i wdrażanie informatyki. Przemysł nie prowadzi żadnej polityki rynkowej, produkuje to, co się akurat uda i stara się wszystko wyeksportować. Beztrosko zaprzestano produkcji Odry, nie oferując żadnych perspektyw dotychczasowym użytkownikom. Nikt nie troszczy się o ujednolicenie kodów znaków oraz klawiatur.

Ogólne zainteresowanie mikrokomputerami przesłoniło fakt zamieniania się Polski w pustynię pod względem większych komputerów, niezbędnych dla poważnych zastosowań w zarządzaniu, bankowości, handlu, przemyśle i badaniach naukowych. Nieliczne produkowane komputery mają skandaliczną jakość — znane są przypadki niesprawności przez 350 dni w roku.

Brak prawnej ochrony oprogramowania powoduje zalew rynku przez programy pochodzące z nielegalnych źródeł (po prostu kradzione). Nikt nie jest zainteresowany inwestowaniem w kosztowne oryginalne oprogramowanie. Użytkownicy nie zdają sobie sprawy, że cena poważnego oprogramowania znacznie przekracza koszt sprzętu. Nikt nie zajmuje się tworzeniem narzędzi sprzętowych i programistycznych inżynierii oprogramowania. Nie istnieje sensowny standard kodów polskich liter.

Kadra profesjonalnych informatyków jest wciąż bardzo szczupła, a mimo tego trzeba ograniczać nabór studentów na kierunek „informatyka" ze względu na brak pomieszczeń i sprzętu. Niewłaściwie uczy się informatyki na kierunkach nieinformatycznych.

Ciągle brakuje warunków do wydawania poważnej i rzetelnej literatury informatycznej, a równocześnie pojawiają się informatyczne „brukowce". Niski jest poziom publikacji „informatycznych" w czasopismach.

Zarząd Główny postuluje powołanie ponadresortowej Państwowej Agencji Informatyki, wytyczającej politykę państwa, realizowaną następnie przez wszystkie resorty. Finansowanie rozwoju informatyki powinno odbywać się przez dotacje lub ulgi dla przedsiębiorstw wdrażających informatykę, a nie przez dotowanie przemysłu produkcji komputerów. Przemysł musi produkować takie komputery, jakie są potrzebne do osiągnięcia wytyczonych celów; nie może być tak, aby informatycy musieli zagospodarowywać to, co akurat przemysłowi udało się wyprodukować. Aby obniżyć ceny komputerów importowanych, należy zliberalizować przepisy zmniejszając liczbę pośredników lub pozwalając przedsiębiorstwom kupować za granicą za dewizy nabyte na wolnym rynku.

Należy nadać indywidualnej działalności usługowej w dziedzinie informatyki status działalności twórczej, a zawodowi informatyka — odpowiednią rangę.

Referat ustępującego Zarządu Głównego został przyjęty przez Zjazd jako Uchwała.

Następnie wiceprzewodnicząca Komisji Rewizyjnej, E. Wiśniewska, przedstawiła sprawozdanie tej komisji. Uznano w nim działalność ustępującego Zarządu za zgodną ze statutem i Uchwałą I Zjazdu, choć w tej kadencji nie opracowano kodeksu etycznego i nie udało się doprowadzić do poprawy sytuacji zawodowej informatyków. Działalność ekonomiczną i finansową oceniono również pozytywnie, postulując dalsze zaostrzenie kontroli nad tematami i jakością prowadzonych prac. Przewodnicząca podkreśliła ogromny wkład pracy ustępującego Prezesa i wystąpiła o udzielenie ustępującym władzom absolutorium, co też nastąpiło.

Prof. A. Mazurkiewicz, jako przewodniczący Sądu Koleżeńskiego, poinformował, iż do Sądu nie zgłoszono żadnych spraw, co zebrani delegaci przyjęli oklaskami.

Następnie równolegle odbywały się wybory władz oraz dyskusja. W dyskusji poruszano głównie problemy zasygnalizowane już w referacie ustępującego Zarządu. Jako dowód arogancji producentów sprzętu przytaczano przykłady nagminnego oczekiwania po 5—7 miesięcy na proste naprawy. Mówiono o konieczności szkolenia nauczycieli wszystkich przedmiotów w zakresie wykorzystywania komputerów na lekcjach. Zaproponowano organizowanie konkursów na konkretne programy użytkowe (tak, jak w wypadku konkursów architektonicznych). Postulowano utworzenie agencji autorskiej zajmującej się ochroną praw autorskich programistów (w rodzaju ZAiKSU). Środowisko krakowskie poinformowało o organizowaniu na początku lipca seminarium na temat stanu prawnego w dziedzinie ochrony oprogramowania. Dyskutowano na temat powołania nowego czasopisma informatycznego (większość delegatów uznała, że raczej należy podnosić poziom Biuletynu PTI oraz istniejących czasopism). Wyrażano krytyczne opinie na temat nowych pism informatycznych; poziom zamieszczonych tam artykułów o wysokich ambicjach profesjonalnych bywa skandaliczny. Wedle malowniczego określenia jednego z dyskutantów: „Pisma propagujące prymitywy informatyczne są groźniejsze od pornografii, bo ta ostatnia, choć w sposób wynaturzony, pokazuje prawdę".

Nowym Prezesem PTI został wybrany prof. Andrzej Blikle, który był zresztą jedynym kandydatem. Wybrany w trzech turach Zarząd Główny ukonstytuował się ostatecznie następująco: wiceprezesi — W. Cellary, M. Maniecki i J. Nowak; skarbnik — J. Zalewski; członkowie Prezydium — J. Bańkowski, P. Gizbert-Studnicki, Z. Mazur, W. M. Turski, J. Zabrodzki; członkowie Zarządu — Z. Huzar, J. Irlik, W. Iszkowski, S. Jaskólski, A. Mazurkiewicz, B. Osuchowska (rzecznik prasowy), T. Syryjczyk, S. Waligórski, H. Woźniakowski. Sekretarzem Generalnym pozostał A. Wiśniewski.

JAROSŁAW DEMINET

Post Scriptum. Obejrzałem listę zaproszonych redakcji i instytucji, które nie uznały Zjazdu PTI za imprezę interesującą i nie przysłały żadnego przedstawiciela. Oto kilka z nich: magazyn TV Spectrum, Bajtek, Mikroklan, Przegląd Techniczny, Polityka, Trybuna Ludu, Życie Warszawy, Min. Łączności, Sekretarz Naukowy PAN, Wydział Nauki i Postępu Technicznego KC PZPR, Komitet Informatyki PAN, Federacja Edukacji Komputerowej. Komentarza nie będzie.

Ośrodek Postępu Technicznego NOT
zaprasza do zwiedzenia

STAŁEJ GIEŁDY ROZWIĄZAŃ TECHNICZNYCH

w Warszawie, ul. Żelazna 51/53 (dawne Zakłady Norblina)
Godziny otwarcia: 9.00—15.00

(oprócz sobót i niedziel)

Najstarsze przedsiębiorstwo polonijne branży elektronicznej **impol II**

02 658 warszawa, ul. filona 16, tel. 43 03 84, 437566, 43 93 41, tlx 817218

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

✓ **imp 85 w+** rewelacyjnie tani mikrokomputer z 4 stanowiskami i pamięcią masową 20 MB

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

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

✓ **master** najtańszy wielodostęp do PC/AT (4 stanowiska)

✓ **imp 86** standard XT Turbo

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

✓ Oprogramowanie użytkowe na dowolne mikrokomputery

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

impol II posiada własne opracowania konstrukcyjne dostosowywane do potrzeb odbiorcy oraz wyspe[illegible]zowane biuro programowania. Serwis naszego s[illegible] prowadzą punkty w Warszawie, Bydgoszczy, Gliwicach, Szczecinie i Krakowie.

EO/544/87

Krzemowy Forth czyli procesor NOVIX NC 4000

Oprogramowanie, zresztą tak jak każdy wytwór kultury, przeżywa najpierw okres rozwoju, potem pełnię świetności, po czym okres schyłkowy i zapomnienie. Choć w tym wypadku cały proces jest wolniejszy aniżeli w wypadku sprzętu, stopniowo jednak idą w zapomnienie również języki programowania. W okresie schyłkowym czasami podejmuje się próbę ratowania danego języka i często kończy się ona powodzeniem. Operacja ta polega zwykle na opracowaniu nowego standardu języka lub zastosowaniu preprocesora poprawiającego syntaktykę.

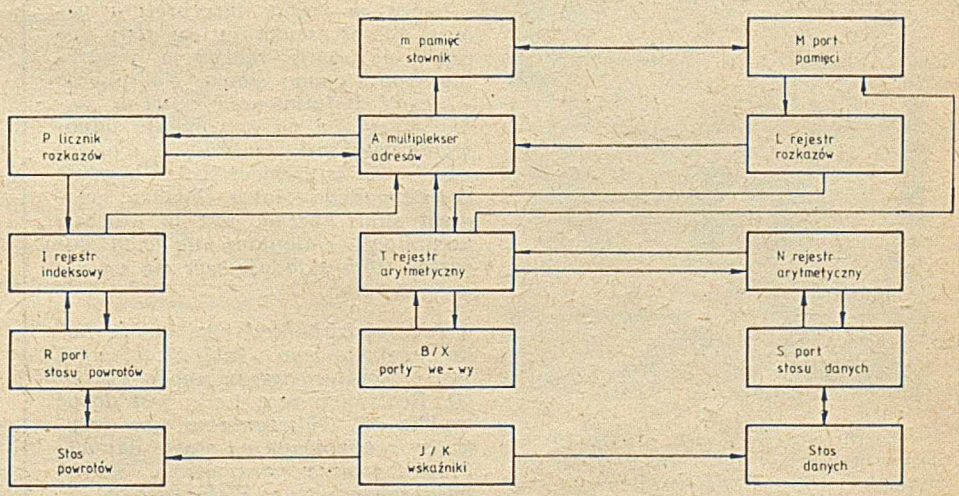
Kto z młodych czytelników słyszał o Algolu 60? Język ten poszedł już właściwie w zapomnienie, podobnie jak i jego następca Algol 68, mimo iż jest on jednym z najważniejszych i najciekawszych języków programowania, jakie do tej pory powstały. Nieco lepiej wygląda sytuacja z Fortranem. Co prawda, opracowany nowy standard języka Fortran 77 nie uzyskał takiej popularności jak jego poprzednik Fortran IV, to jednak pewne sukcesy przyniosło pojawienie się Ratforu (preprocesora poprawiającego składnię). Podobne próby podejmowane były z językiem Pascal, którego następca Modula 2 jakoś nie może dorównać popularnością swojemu poprzednikowi. Niedawno opracowano odpowiednik Ratforu dla Pascala. Nosi on nazwę Rascal. Inaczej wyglądała rzecz z Lispem. Język ten powstały w końcu lat pięćdziesiątych, został początkowo zarzucony jako mało efektywny, by po dwudziestu latach „odkurzony" głównie za sprawą firmy Xerox znów odzyskać popularność. Szczytowy okres popularności przeżywa obecnie język C, aczkolwiek również dla niego przygotowane są ulepszone wersji, np. C++.

Język Forth, choć nigdy nie osiągnął (i prawdopodobnie już nie osiągnie) takiej popularności jak język C, przeżywa też swój okres świetności. Opracowane zostały kolejne, ulepszone standardy języka Forth 79 i Forth 83. Stopniowo zostały wyeliminowane (przez standardowe lub niestandardowe rozszerzenia) takie wady języka jak brak operacji zmiennoprzecinkowych, „prymitywny" system plików itd. Niewątpliwym brakiem jest także brak standardu dla mikroprocesorów 32-bitowych, wobec czego wszystkie implementacje 32-bitowe (a powstało ich sporo) są siłą rzeczy niestandardowe. Nie na wszystkich mikroprocesorach Forth jest jednakowo efektywny. Krytyczny jest w tym wypadku czas realizacji operacji NEXT wtrącanej po wykonaniu każdego słowa, a także — operacji rozpoczęcia i zakończenia realizacji kodu kaskadowego CALL i EXIT [1]. Istotny jest także czas aktywacji i deaktywacji zadania ACTIV i DEACT dla wersji wielozadaniowych (tab. 1).

Z tabeli wynika, że na niektórych procesorach implementacje Fortha są mało efektywne. Dość dobrze natomiast pracuje Forth realizowany przy wykorzystaniu procesorów posiadających rozbudowane operacje stosowe (PDP-11 i LSI-11). Każda jednakże implementacja Fortha będzie niejako z definicji gorsza od języka C czy Pascala, których kompilatory generują kod liniowy (bez wtrącania operacji NEXT). Rozwiązaniem tego problemu jest realizacja wirtualnej maszyny Fortha.

Próba taka już została podjęta [2] i zakończona powodzeniem. Efektem jej jest nowy 16-bitowy mikroprocesor Novix NC 4000. Sama idea nie jest zresztą nowa. Znane są już bowiem sprzętowe realizacje języków wysokiego poziomu, m.in. języka C (BBN C-machine), Lispu (Lisp-machine), Pascala (sprzętowy interpreter p-codu), a nawet Cobolu. Istniały też wcześniej procesory Fortha (na przykład — R6511 firmy Rockwell). Większość z nich jednak emuluje jedynie wirtualne maszyny języków wysokiego poziomu, przy użyciu odpowiednio rozbudowanego mikroprogramu lub zapisanego w pamięci stałej programu (procesor R6511, na przykład, jest w istocie połączeniem procesora 6502 z pamięcią ROM, zawierającą „jądro" Fortha w jednej strukturze półprzewodnikowej).

Procesor NC 4000 jest czymś zupełnie innym. Stanowi pełną sprzętową realizację wirtualnej maszyny Fortha (rys. 1). Autorem jego architektury jest twórca języka Charles Moore (fot.). Dzięki sprzętowej realizacji „pięta achillesowa" innych implementacji Fortha — operacja NEXT — jest realizowana w czasie pojedynczego taktu zegarowego. Wskutek tego, jak również wskutek zrównoleglenia wielu operacji dzięki jednoczesnemu dostępowi do obszaru słownika i obu stosów (które nie są emulowane w pamięci operacyjnej, lecz istnieją fizycznie), możliwe stało się osiągnięcie szybkości rzędu 8 mips (milionów operacji na sekundę). Należy podkreślić, że są to operacje w języku wysokiego poziomu. Wiele sekwencji słów Fortha. np. { SWAP — }, { OVER SWAP — }, { ᵾ + }, { DUP ᵾ SWAP nn + } jest wykonywanych w czasie jednego taktu zegarowego. W efekcie jest to jeden z najszybszych mikroprocesorów, jakie dotąd skonstruowano. Dla porównania można podać, że bardzo popularny program testowy generujący liczby pierwsze metodą sita Eratostenesa jest wykonywany przez procesor NC 4000 przeszło dziesięciokrotnie szybciej aniżeli analogiczny program napisany w języku asemblera mikroprocesora MC 68000 firmy Motorola [3].



Architektura procesora NC 4000

Twórcy tego procesora, dzięki zmianie technologii (obecnie NC 4000 jest wykonany jako tzw. matryca bramek CMOS), zamierzają uzyskać znacznie wyższą częstotliwość zegara, a co za tym idzie — także szybkość. Szacuje się, że będzie to możliwe dzięki specjalnej technologii CMOS lub ECL, w której mają być wykonane następne,

Tabela 1. Porównywanie liczby rozkazów niezbędnych do realizacji podstawowych operacji Fortha dla różnych procesorów

CPU	NEXT	EXIT	CALL	ACTIV	DEACT
8080/Z80	12	7	10	17	16
PDP/LSI—11	2	1	2	7	4

prawdopodobnie już 32-bitowe, wersje procesora. Najważniejsze parametry wersji 16-bitowej są zebrane w tab. 2.

Tabela 2. Parametry procesora NC 4000

Dane techniczne procesora NC 4000	
Producent	Mostek
Technologia	HCMOS
Szybkość	8 mips
Częstotliwość zegara	8 MHz
Maksymalna pojemność pamięci	4 MB
Stos danych	$\frac{1}{2}$ KB
Stos powrotów	$\frac{1}{2}$ KB
Liczba linii we-wy	24
Liczba końcówek	124
Obudowa	Wielorzędowa (ang. pin grid array)
Cena	225—275 dolarów

W odróżnieniu od klasycznych implementacji Fortha, procesor NC 4000 używa tzw. bezpośredniego kodu kaskadowego DTC (ang. direct threaded code) zamiast pośredniego kodu kaskadowego ITC (ang. indirect threaded code) [1]. Oznacza to, że program w kodzie kaskadowym nie jest sekwencją adresów, lecz sekwencją kodów operacji. Umożliwia to dodatkową oszczędność pamięci obszaru słowników. Składające się na program określonego słowa poszczególne 16-bitowe słowa w pamięci operacyjnej są interpretowane następująco: jeżeli najbardziej znaczący bit słowa jest równy „0", to pozostałe 15 bitów jest traktowane jako adres innego słowa w kodzie kaskadowym (czyli coś w rodzaju skoku do podprogramu), w przeciwnym zaś razie te 15 bitów jest traktowane jako kod realizowanej operacji. Warto zauważyć, że jest on 15-bitowy, co daje możliwość zakodowania ogromnej liczby rozkazów (należy pamiętać, że Forth jest maszyną stosową, czyli bezadresową). Ponieważ rzeczywista liczba rozkazów jest niewielka (rzędu kilkudziesięciu) 15-bitowy kod operacji znakomicie ułatwia konstrukcję i przyspiesza działanie dekodera rozkazów. Można powiedzieć, że NC 4000 jest „mikroprogramowany zewnętrznie" w odróżnieniu od większości innych procesorów „mikroprogramowanych wewnętrznie".

Zestaw rozkazów realizowanych przez procesor NC 4000 zawiera operacje stosowe DUP, SWAP, OVER, DROP, >R, R>, I, dodawanie i odejmowanie (z przeniesieniem lub bez), pojedyncze kroki mnożenia i dzielenia, a także pierwiastkowania, operacje logiczne AND, OR i XOR, przesunięcia i skoki, operację powtórzenia, pobranie i załadowanie danej, operacje rozszerzenia adresu i parę innych. W sumie jest to 40 rozkazów (tab. 3). Dodatkowo, dzięki możliwości zrównoleglania wykonywanych operacji, niektóre ich sekwencje (tzn. sekwencje słów Fortha) kodowane w jednym 15-bitowym kodzie rozkazu wykonują się w czasie jednego taktu. Powoduje to faktyczne rozszerzenie listy rozkazów do ok. 170 kodów operacji. Owe 40 rozkazów tworzy „jądro" Fortha. Pozostałe słowa Fortha są realizowane za pomocą kodu kaskadowego.

Procesor pracuje w tzw. cyklu „decode and load". Oznacza to, że w czasie wykonania bieżącego rozkazu rejestr rozkazów procesora L zawiera już następny rozkaz, którego realizacja może rozpocząć się natychmiast po zakończeniu wykonywania rozkazu bieżącego. W sytuacji, gdy następny rozkaz jest skokiem, adres efektywny skoku pojawia się na magistrali adresowej przed rozpoczęciem następnego cyklu zegarowego. Rozkaz powtórzenia TIMES umożliwia wielokrotne wykonanie rozkazu, którego kod zawiera rejestr L. Liczbę powtórzeń określa w tym wypadku zawartość szczytu stosu powrotów.

Procesor ma piętnaście rejestrów 16-bitowych. Ich zestaw obejmuje wskaźniki stosów J/K, rejestr rozkazu L, rejestr liczby powtórzeń #TIMES, rejestr indeksowy INDEX, licznik rozkazów PC, rejestry arytmetyczne T i N, rejestr mnożenia oraz dzielenia MD, rejestr pierwiastkowania SR oraz po cztery specjalne rejestry dla każdego portu we-wy B i X. Dwa argumenty ze szczytu stosu danych są pamiętane stale w rejestrach procesora T (słowo leżące na szczycie stosu) i N (słowo leżące bezpośrednio pod nim). Przyspiesza to znacznie realizację operacji arytmetyczno-logicznych. Dodatkowy specjalny rejestr TRUE, którego zawartość można jedynie odczytać, umożliwia szybkie generowanie wartości logicznej „true" (FFFFH).

Pierwsze 32 słowa obszaru słowników mogą być bezpośrednio adresowane przy użyciu wyróżnionego 5-bitowego pola w kodzie niektórych rozkazów. Umożliwia to bardzo szybki dostęp do tychże słów pamięci, dzięki czemu obszar ten może być traktowany jako zbiór 32 szybkich pseudorejestrów. Lista rozkazów procesora przewiduje specjalne rozkazy do komunikacji z tym obszarem.

Magistrala adresowa 16-bitowa zapewnia procesorowi dostęp zaledwie do 64 K słów (128 KB) pamięci obszaru słowników. Adres ten może być rozszerzony o dodatkowe 5 bitów (zawartość portu X), umożliwiając w ten sposób dostęp do pamięci o pojemności 4 MB.

Czym jest Forth?

Forth jest językiem programowania opracowanym przez Charlesa Moore'a (fot.). Został stworzony w celu uzyskania większej wydajności w pisaniu programów. Jest przeznaczony przede wszystkim dla programistów systemowych. Jego system programowania składa się z interpretera oraz kompilatora języka wysokiego poziomu. Edytor, a często i asembler, są również integralnymi składnikami tego systemu.

Program napisany w języku Forth składa się z ciągu słów rozdzielonych spacjami. Słowa składają się z dowolnych znaków mających reprezentację graficzną. Każdemu słowu odpowiada w pamięci pewna struktura, w której zapamiętana jest nazwa słowa oraz jego program. Struktury te są połączone w listy zwane słownikami.

Praca w trybie interpretera polega na przeszukaniu tej listy, po czym następuje realizacja programu znalezionego słowa. Kompilacja — czyli definiowanie nowego słowa — polega na utworzeniu odpowiadającej mu struktury i włączeniu jej do istniejącej listy. Program definiowanego słowa, będący ciągiem słów Fortha, jest w procesie kompilacji zmieniany na ciąg adresów słów składających się na jego program.

Tego rodzaju kod nosi nazwę kodu kaskadowego lub nizanego i jest charakterystyczną cechą Fortha [1]. Realizacja tego kodu jest dosyć szybka [4]. Obliczenia w tym języku są realizowane na stosie danych. Przekazywanie parametrów następuje także przy użyciu tego stosu. Drugi stos, zwany stosem powrotów, służy — w czasie realizacji programu — do przechowywania adresów zagnieżdżających się słów w kodzie kaskadowym. Pamięć dyskowa traktowana jest przez Forth jako pamięć wirtualna.

(Zb. Szk.)

Procesor Novix 4000 znalazł już szereg zastosowań, m.in. w profesjonalnej grafice komputerowej oraz przy przetwarzaniu sygnałów. Amatorzy mogą nabyć, za około 400 dolarów zestaw (pod nazwą Delta-board) do budowy własnego jednopłytkowego komputera opartego na procesorze NC 4000. Zastosowany w nim procesor pracuje przy częstotliwości zegara 4 MHz, osiągając szybkość 4 mips. Takie celowe zmniejszenie częstotliwości taktowania pozwala zastosować tańsze pamięci. Oprócz procesora, na płytce znajduje się pamięć RAM (maksymalnie 32 K słów, 8 układów 6264P-12), pamięć EPROM (4 K słów, 2 układy 2732A-2) mieszcząca wersję cm-Forth języka, 3 porty we-wy i parę innych układów. Całość (17 ukiadów scalonych) mieści się na płytce o wymiarach 6*4 cala i pobiera moc rzędu 1 W. Dla posiadaczy komputerów IBM PC produkowana jest pod nazwą Beta-board specjalna karta z procesorem NC 4000. Gwarantuje ona szybkość 8 mips dzięki zastosowaniu szybkich statycznych pamięci CMOS o czasie dostępu 35 ns. Kosztuje jednak około 3500 dolarów.

Tymczasem konkurencja nie śpi. W Wielkiej Brytanii został opracowany przez firmę Metaforth konkurencyjny procesor MF 1600. Wykonuje on 6,6 miliona operacji Fortha na sekundę (mips). Wykonany jest w technologii bipolarnej i taktowany zegarem o częstotliwości 20 MHz. Dzięki 32 liniom adresowym może bezpośrednio zaadresować 2 GB pamięci operacyjnej. Produkowana karta z tym procesorem jest dostosowana do współpracy z magistralą VME. Procesor ten doczekał się już także dość licznego oprogramowania, m.in. dobrych i szybkich translatorów języków C, Pascal i Lisp.

Czas pokaże, czy prezentowane rozwiązania przyjmą się czy też nie. Niemniej warto śledzić ten nurt w rozwoju mikroprocesorów stanowiący ciekawą alternatywę dla procesorów RISC (zastosowany w IBM RT/PC nie osiąga nawet szybkości 3 mips) i Transputera firmy INMOS (który, co prawda, pracuje z szybkością 10 mips, niemniej jest trudny do oprogramowania).

Tabela 3. Lista rozkazów procesora NC 4000 (zawiera tylko rozkazy odpowiadające pojedynczym słowom Fortha)

Nazwa		Znaczenie
Operacje stosowe		
dup	(n — — — n n)	Powielenie liczby
drop	(n — — —)	Usunięcie liczby
over	(a b — — — a b a)	Kopiowanie drugiej liczby
swap	(a b — — — b a)	Zamiana liczb
Operacje arytmetyczno-logiczne		
+	(a b — — — a+b)	Dodawanie
+c	(a b — — — a+b)	Dodawanie z przeniesieniem
—	(a b — — — a—b)	Odejmowanie
—c	(a b — — — a—b)	Odejmowanie z przeniesieniem
or	(a b — — — aORb)	Suma logiczna
and	(a b — — — aANDb)	Iloczyn logiczny
xor	(a b — — — aXORb)	Różnica symetryczna
2/	(n — — — n/2)	Przesunięcie w prawo liczby n
2*	(n — — — n*2)	Przesunięcie w lewo liczby n
0<	(n — — — ?)	Testowanie czy n<0?
D2/	(d — — — d/2)	Przesunięcie w prawo liczby d
D2*	(d — — — d*2)	Przesunięcie w lewo liczby d
*'	(d — — — d)	Pojedynczy krok mnożenia
*_	(d — — — d)	Krok mnożenia ze znakiem
*F	(d — — — d)	Krok mnożenia ułamkowy
/'	(d — — — d)	Pojedynczy krok dzielenia
/"	(d — — — d)	Ostatni krok dzielenia
S'	(d — — — d)	Krok pierwiastkowania
Sterowanie stosem powrotów		
R>	(— — — n)	Pobranie zawartości szczytu stosu powrotów
Ru	(— — — n)	Kopiowanie zawartości szczytu stosu powrotów
#I	(— — — n)	Kopiowanie indeksu pętli
>R	(n — — —)	Przesłanie na stos powrotów
Operacje sterujące		
if		Skok warunkowy
else		Skok bezwarunkowy
#loop		Skok z dekrementacją licznika
times	(n — — —)	Powtórzenie
call		Skok do podprogramu
exit		Powrót
Operacje komunikacji z pamięcią i we-wy		
@	(adr — — — n)	Pobranie z pamięci
!	(n adr — — —)	Zapis do pamięci
a	(adr — — — n)	Pobranie z rejestru
a!	(n adr — — —)	Zapis do rejestru
n	(— — — n)	Pobranie literału

LITERATURA

[1] Kott R., Magdzik D.: Techniki interpretacji dla mikrokomputerów. Informatyka, nr 4, 1984

[2] Novix NC 4000P Microprocessor. Karta katalogowa

[3] Strass H., Brodie L.: Der Forth-Mikroprozessor — Programmiersprache und CPU aufeinander optimal abgestimmt. Design und Elektronik, Nr. 5, 1986

[4] Szkaradnik Z.: Porównanie języków programowania mikrokomputerów. Informatyka, nr 11—12, 1985.

ZBIGNIEW SZKARADNIK

Przyp. red.: Czytelnikom bliżej zainteresowanym tym językiem przypominamy, że opisano go dokładniej w Informatyce nr 5, 7, i 8, 1984.

W skrócie • W skrócie

Po pewnym spadku zamówień w 1985 r., wytwórcy stacji dysków stałych o średnicy 5 1/4 cala przeżywali gwałtowny rozwój w 1986 r. Ośmiu czołowych producentów, a mianowicie: Iomega Corp., Maxtor Corp., Micropolis Corp., MiniScribe Corp., Priam Corp., Quantum Corp., Seagate Technology i Tandon Corp., osiągnęło w pierwszym kwartale 401 milionów dolarów za sprzedaż swych wyrobów, co oznacza wzrost o 65% w stosunku do analogicznego okresu przed rokiem i o 18% w stosunku do poprzedniego kwartału. Jeszcze lepsze są wskaźniki zysków: kwartalny wzrost dla Seagate wyniósł 152%, dla Tandon — 132%, dla Miniscribe — 90% i dla Micropolis — 66%.

Wzrost sprzedaży w II kwartale był przewidywany na 4% w stosunku do poprzedniego kwartału, co oznacza 47-procentowy wzrost w ciągu roku, zaś zysk większy o 325% w stosunku do roku 1985.

Sytuacja ta jest spowodowana ogólnym wzrostem sprzedaży komputerów osobistych o 15—20%, wśród których systemów z dyskami stałymi jest coraz więcej. Ponadto w roku 1986 istniało 3—5 milionów systemów z dyskami elastycznymi; pojemność pamięci wielu z nich zwiększono przez dołączenie stacji dysków stałych. Wreszcie ostatnim czynnikiem sprzyjającym rozwojowi było zastępowanie dysków 8-calowych dyskami o średnicy 5 1/4 cala. Wszystkie nowe systemy komputerowe, jak układy zapewniające dostęp do zbiorów (ang. file servers), wielozadaniowe mikrokomputery dla wielu użytkowników i skomputeryzowane stanowiska pracy, zawierają stację dysków 5 1/4 cala o pojemności 40—85 MB. W roku 1985 sprzedano tych stacji za 150 mln dolarów, a w roku 1986 przewidywano sprzedaż za ponad 400 mln dolarów. (JR)

W skrócie • W skrócie

Co nowego na rynku komputerowym?

SYSTEMS 87

Monachium, 19—23 października 1987

Fachowa odpowiedź na pytania:

Co nowego na rynku komputerowym?

Jakie trendy występują w tej dziedzinie?

Jakie systemy wiążą się z dniem dzisiejszym, a jakie z przyszłością?

Jakie posiadamy urządzenia i oprogramowanie?

SYSTEMS — jedyne w Europie specjalistyczne targi z kongresem — ponad 1000 wystawców z 16 krajów

Szeroka oferta zarówno dla początkujących, jak i doświadczonych użytkowników

Systems'87 — efekt pomysłowości

Specjalistyczne targi dla fachowców

Osiągnięcia w dziedzinie techniki komputerowej

MESSE MÜNCHEN INTERNATIONAL

Informacji udziela:

Przedstawicielstwo Monachijskiego Towarzystwa Wystaw i Targów

POLEXPO Przedsiębiorstwo Wystaw i Targów Zagranicznych,

ul. Łopuszańska 38, 00-971 Warszawa
tel. 46-45-92, teleks: 813633

EO/72

Procesory Intela — inny punkt widzenia

Niewątpliwie najlepiej „sprzedającym się" komputerem osobistym na świecie został IBM PC. Wypełnił on lukę między systemami profesjonalnymi, jak PDP-11 czy ONYX-8001, a mikrokomputerami nadającymi się wyłącznie do zabawy. Merytorycznej czy — co ważniejsze — cenowej konkurencji nie widać. Czy na pewno oznacza to, iż procesory Intela nie mają sobie równych w świecie?

Dla przeciętnego użytkownika nie jest istotny poziom kodu maszynowego komputera. Używa, na przykład, TURBO-87 i na dobrą sprawę może zapomnieć o istnieniu systemu operacyjnego. Jeśli ktoś mu już napisał odpowiedni AUTOEXEC, to wystarczy wiedza o tym, jak włączyć komputer, drukarkę i monitor. Jednakże pisanie programów o charakterze systemowym pobudza do pewnych refleksji.

Przyzwyczajenie się do składni języka asemblera I-8086 idzie dosyć opornie, co jest wspólną cechą chyba wszystkich mikroprocesorów wydanych na świat przez firmę Intel. Wyjątkowa jest nieczytelność programów, a odwołując się do zjawisk mniej subiektywnych — ścisłe przyporządkowanie funkcji poszczególnym rejestrom bywa dosyć uciążliwe, szczególnie dla programistów przyzwyczajonych do posługiwania się dużą liczbą rejestrów (Zilog Z-8000, Motorola 68000). Są to wszystko skutki łańcucha decyzji firmy, która wierząc (i słusznie) w szeroką bazę oprogramowania powstałego dla mikroprocesora 8080 postanowiła, że planowany do produkcji mikroprocesor 16-bitowy musi umożliwiać wykorzystanie programów z 8080. Stąd rejestry mikroprocesora 8086 mają swoje odpowiedniki w 8080. Stąd też wynika pojawienie się translatora XLT86. Jak ta decyzja wpłynęła na elastyczność systemu i jego szybkość — widać przy każdym bardziej skomplikowanym problemie, który trzeba rozwiązać przy użyciu procesora 8086. A w IBM PC/XT siedzi jego 8-bitowa wersja 8088... Zresztą mikroprocesor 8086 wcale nie jest tak bardzo 16-bitowy, jak na to wygląda. Pozostałości 8-bitowych zostało w nim bowiem bardzo dużo, z powolną i „krótką" arytmetyką na czele.

Warto też pamiętać o dosyć istotnym, historycznym już dzisiaj fakcie, którego przypomnienie wobec tak dalekich związków pomiędzy 8080 a jego 16-bitowymi następcami wydaje się niezbędne. Jest nim „obciążenie dziedziczne" procesora 8080 pozostałościami z systemu 8008, a więc z czasów, kiedy znikome doświadczenie firm nie pozwalało na śmiałe odchodzenie od już istniejących konstrukcji, jak to nastąpiło później (np. rodzina 8048, czy Zilog Z-8). Konsekwencje tego widzimy do dziś — i to nie tylko w rodzinie Intel 8086, ale także w Z-80 czy Z-800. Wystarczy porównać zestaw rejestrów i listę rozkazów procesorów 8008 i 8080. Dla przypomnienia — 8008 zawiera 7 rejestrów: **A, B, C, D, E, H, L**, a ósmy, wolny kod identyfikacji rejestru, nazywa się **M** i oznacza zawartość pamięci o adresie w parze **HL**. Lista rozkazów procesora 8080 stanowi nadzbiór listy 8008. Wynikające stąd niespójności są widoczne przy analizie rozkazów i kodów operacyjnych 8080 czy Z-80. Wygoda stosowania trybu pośredniego rejestrowego i dążenie do „ortogonalności względem rejestrów" wymusiło wprowadzenie rozkazów **LDAX** i **STAX** o zmienionej nazwie i strukturze kodu w stosunku do funkcjonalnego odpowiednika **MOV r,M** i **MOV M,r** dla pary **HL**. Podobne wyróżnienie pary **HL** jest widoczne w rozkazach **LHLD, SHLD, XCHG, SPHL, PCHL, XTHL**. Z kolei rozszerzenie listy rozkazów przy opracowywaniu Z-80 zapewniło większą swobodę w doborze rejestrów procesora, szczególnie wobec dodania rejestrów indeksowych, ale założenie binarnej kompatybilności z procesorem 8080 spowodowało zmianę długości czasu wykonania i struktury kodu rozkazu w zależności od tego, jakich rejestrów dotyczy (odpowiedniki LHLD, SHLD, SPHL, PCHL, XTHL). Pytanie o sens binarnej odpowiedniości w odniesieniu do Z-80 wydaje się retoryczne — zakres stosowania tego procesora potwierdza słuszność przedsięwzięcia mimo uwarunkowań z przeszłości.

Podsumowując, w procesorach rodziny Intel 8086 dają się zauważyć obciążenia pierwszym w historii mikroprocesorem 8008 (z 1972 roku...) i należy tylko cieszyć się, że firma zrezygnowała z kontynuowania kompatybilności binarnej z 8080. Żadna zresztą z firm nie zdecydowała się na „binarną" kontynuację swoich 8-bitowych dokonań przy projektowaniu mikroprocesora z 16-bitową szyną danych. Rozwój systemów 8-bitowych zatrzymał się na etapie procesorów kompatybilnych z Z-800 (binarnie zgodny z Z-80, co spowodowało duży zamęt w kodach operacyjnych tego bardzo rozbudowanego mikroprocesora) oraz odpowiedników nowych wyrobów „w dół", tzn. do 8-bitowej szyny danych. I nic nie wskazuje na to, aby ten stan rzeczy miał się zmienić — nie widać potrzeby opracowywania nowego procesora 8-bitowego, zwłaszcza wobec niskich cen pamięci i procesorów 16-bitowych.

Mimo wszystko, Intel 8086 daje się też programować, tyle że czasami mniej wygodnie i zazwyczaj wolniej działają na nim programy, ale to już nie wina IBM — po prostu takie są te procesory. Jeśli chodzi o możliwość translacji oprogramowania — podobne działania są wykonalne także w rodzinie Ziloga. Możliwa jest maszynowa translacja programów źródłowych Z-80 na Z-8000, podobnie jak to czyni program XLT86 w rodzinie Intela. Niemal wszystkie rozkazy Z-80 dają przetłumaczyć się w stosunku 1:1 na szczególne przypadki „ortogonalnych rejestrowo" instrukcji Z-8000, a wyjątek, tzn. EXX można zastąpić zmianą przyporządkowania rejestrów na poziomie definiowania dla nich nazw logicznych lub przez ciąg operacji wymiany zawartości rejestrów.

Pragnącym programować szybko i bez zbędnych stresów mogę zaproponować próbę porównania szybkości między IBM PC/AT i wielodostępnym Systemem 8000 lub ONYXEM 8001 (którego instalacja istnieje i działa w Polsce), ewentualnie polecam czekanie na IBM RT/PC (z procesorem o zredukowanej liczbie rozkazów).

MICHAŁ CHOROSZUCHA

Wordstar 3.30

dokończenie ze str. 16

W — ciągłe przeglądanie tekstu oknem w górę (wprowadzanie kolejnych wierszy z góry); działa od chwili naciśnięcia dowolnego klawisza, z wyjątkiem klawiszy 1—9 sterujących szybkością przemieszczania;
P — przesunięcie kursora na pozycję, w jakiej znajdował się przed ostatnio wykonanym dowolnym poleceniem;
V — przesunięcie kursora na pozycję, w jakiej znajdował się przed ostatnim poszukiwaniem lub zamianą ciągu znaków.

• **Usuwanie** (ang. delete):
Y — usuwanie tekstu w wierszu na prawo od kursora włącznie;
DEL — usuwanie tekstu w wierszu na lewo od kursora wyłącznie.
• **Różnorodne** (ang. miscellaneous):
F — wyszukiwanie określonego ciągu znaków w pliku;
A — wyszukiwanie określonego ciągu znaków w pliku i zamiana na inny, określony ciąg znaków;
L — przesunięcie kursora do kolejnego ciągu wyszukiwanych lub zamienianych znaków (powtórzenie ostatniego wyszukiwania);
Q — powtarzanie polecenia następującego po tym poleceniu (np. wyszukiwanie i zamiana znaków w całym tekście).

Jones C. B.: Specyfikacje i programy (1)

INFORMATYKA 1987, nr 10, s. 1

Pierwsza część charakterystyki metody VDM w zastosowaniu do konstruowania oprogramowania.

Jones C. B.: Specifications and programs (1)

INFORMATYKA 1987, No. 10, p. 1

First part of characteristics of the VDM method and its application for software constructing.

Jones C. B.: Spezifikationen und Programme (1)

INFORMATYKA 1987, Nr. 10, S. 1

Erster Teil einer Charakteristik von VDM-Methode und ihrer Anwendung zur Softwarekonstruktion.

Staniszkis W.: Zarządzanie rozproszonymi danymi — przegląd problematyki (1)

INFORMATYKA 1987, nr 10, s. 5

Pierwsza część wprowadzenia do problematyki zarządzania rozproszonymi danymi.

Staniszkis W.: Distributed data management — problems survey (1)

INFORMATYKA 1987, No. 10, p. 5

First part of an introduction to distributed data managament problems.

Staniszkis W.: Verteilte Datenverwaltung — eine Problematikübersicht (1)

INFORMATYKA 1987, Nr. 10, S. 5

Erster Teil einer Einführung zur Problematik der verteilten Datenverwaltung.

Zielczyński P.: MULTICOMP — system rozwiązywania zadań metodą przeszukiwania drzew (2)

INFORMATYKA 1987, nr 10, s. 8

Druga część artykułu na temat języka Multicomp przeznaczonego do rozwiązywania zadań metodą przeszukiwania drzew, zawierająca opis oraz przykład zastosowania tego języka.

Zielczyński P.: MULTICOMP — a task solving system through tree search (2)

INFORMATYKA 1987, No. 10, p. 8

Second part of the paper on Multicomp language for task solving through tree search, which presents language description and an example of its application.

Zielczyński P.: MULTICOMP — ein System für Aufgabenlösung mit Anwendung von Baumsuchverfahren (2)

INFORMATYKA 1987, Nr. 10, S. 8

Zweiter Teil des Artikels über die Multicomp-Sprache für Aufgabenlösung mit Anwendung von Baumsuchverfahren, der eine Sprachebeschreibung und ein Anwendungsbeispiel umfasst.

Bielecki J.: Podstawy grafiki w języku Turbo Pascal (3)

INFORMATYKA 1987, nr 10, s. 11

Trzecia część artykułu na temat grafiki w języku Turbo Pascal, zawierająca omówienie zasad oraz przykłady zarządzania oknami i grafiką żółwia.

Bielecki J.: Graphics essentials in Turbo Pascal (3)

INFORMATYKA 1987, No. 10, p. 11

Third part of the paper on graphics in Turbo Pascal, which contains discussion of principals and examples for windows and turtle graphics management.

Bielecki J.: Grafikgrundlagen in Turbo Pascal (3)

INFORMATYKA 1987, Nr. 10, S. 11

Dritter Teil des Artikels über Grafik in Turbo Pascal, der eine Besprechung von Grundlagen und Beispiele der Fenster- und Schildkrötegrafikverwaltung umfasst.

Ossowska S.: Wordstar 3.30 — zasady działania i sposób użytkowania (2)

INFORMATYKA 1987, nr 10, s. 15

Druga część charakterystki zasad działania i sposobu użytkowania pakietu Wordstar 3.30.

Ossowska S.: Wordstar 3.30 — operating principles and handling (2)

INFORMATYKA 1987, No. 10, p. 15

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

Ossowska S.: Wordstar 3.30 — Betriebsgrundsätze und Handhabung (2)

INFORMATYKA 1987, Nr. 10, S. 15

Zweiter Teil einer Charakteristik von Grundsätzen und Handhabung des Wordstar 3.30-Programmpakets.

Bielecki J.: Język C — programowanie operacji wejścia-wyjścia

INFORMATYKA 1987, nr 10, s. 17

Omówienie funkcji przeznaczonych do programowania operacji wejścia-wyjścia w języku C.

Bielecki J.: C language — I/O operations programming

INFORMATYKA 1987, No. 10, p. 17

Discussion of functions for I/O operations programming in C language.

Bielecki J.: C-Sprache — Programmierung von Ein- und Ausgabeoperationen

INFORMATYKA 1987, Nr. 10, S. 17

Eine Besprechung von Funktionen für Programmierung der Ein- und Ausgabeoperationen in C-Sprache.

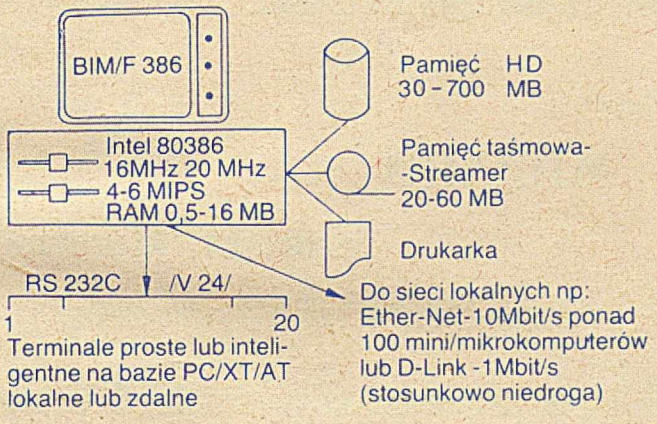
PZ globo®

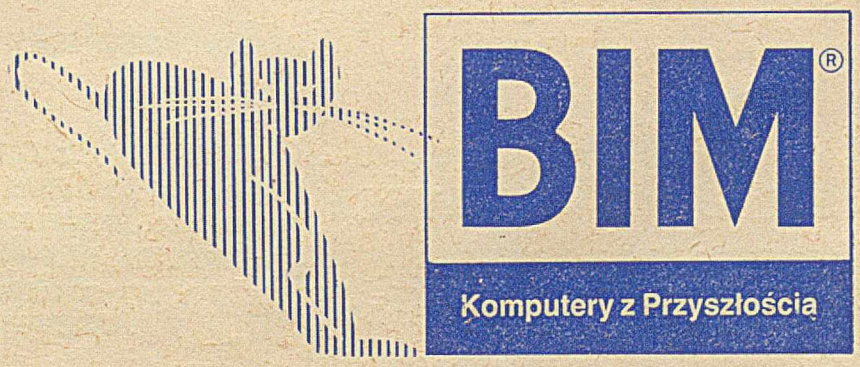
Licencjonowany producent firmy BIM
Koplin 73 200 Choszczno Tel. 7550 Telex 0445413

Oferuje nowoczesne, niezawodne, bardzo wydajne:

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

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





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

EO/614/87

KONIEC TWOICH PROBLEMÓW

Nareszcie wszystkie potrzebne Ci dane dotrą na czas
w przejrzystej formie tabel i wykresów.

Pakiet oprogramowania na mikrokomputery 16-bitowe

MEGA — BANK

to nowe MEGA możliwości jakie otwierają się przed
Twoim przedsiębiorstwem.

Wyobraź sobie

MILIARD

rekordów, które możesz
zapełnić według własnych potrzeb i uznania.

Miliard kooperantów, miliard pracowników, miliard produktów
— z tym wszystkim nasz MEGA-BANK poradzi sobie bez trudu.

System jest łatwy w obsłudze i opracowany w języku polskim.

Gwarantujemy satysfakcję!

COMPUTER STUDIO KAJKOWSCY
PROFESJONALNE OPROGRAMOWANIE MIKROKOMPUTERÓW
ul. Balladyny 3B, 81-524 Gdynia, tel.: 29-00-18, 24-01-50

BO/74/87