9 1988

informatyka

Przegląd technik graficznych
Sterowanie przepływowe
System operacyjny IPIX

WYDAWNICTWO NOT SIGMA

Cena 250 zł

informatyka

9 1988

Nr 9

Miesięcznik Rok XXIII
Wrzesień 1988

Organ Komitetu
Naukowo-Technicznego NOT
ds. Informatyki

Materiałów nie zamówionych redakcja nie zwraca

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

RSW „PRASA-KSIĄŻKA-RUCH"
PRASOWE ZAKŁADY GRAFICZNE
ul. Dworcowa 13 85-950 BYDGOSZCZ
Zam. 3046/88. E-10
Obj. 4,0 ark. druk. Nakład 9000 egz.

ISSN 0542-9951. INDEKS 36124

Cena egzemplarza 250 zł

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

W NUMERZE Strona

W NAJBLIŻSZYCH NUMERACH:

- **Leslie Lamport prezentuje nową metodę ułatwiającą specyfikowanie systemów współbieżnych**
- **Maciej M. Sysło dokonuje przeglądu zrealizowanych modeli równoległych systemów komputerowych oraz zaprojektowanych dla nich algorytmów**
- **Henryk Gizdoń, Adam Pawlak i Włodzimierz Wrona naświetlają genezę powstania oraz przedstawiają charakterystykę języka opisu i projektowania sprzętu komputerowego VHDL**
- **Wojciech Cellary, Jerzy Kręglewski i Ruta Maćkowiak opisują system mikrokomputerowy Elwro 800**
- **Michał Kabsch omawia efektywność systemu mikrokomputerowego zarządzania bazą danych dBase III**
- **Jan Bielecki daje praktyczne wskazówki stosowania niektórych rozwiązań języka programowania Turbo C na mikrokomputerach klasy IBM PC**

WOJCIECH MOKRZYCKI
Instytut Podstaw Informatyki
Polska Akademia Nauk
Warszawa

Przegląd technik grafiki komputerowej (1)

Dominująca część współczesnej techniki komputerowej to technika cyfrowa, nienaturalna i mało komunikatywna. Kontrastuje z zachodzącymi w otaczającym nas świecie zjawiskami fizycznymi, mającymi, ze swojej natury, charakter analogowy. Zastosowanie komputera do badania otaczających nas zjawisk wymaga nadania im postaci dyskretnej. Dla człowieka, ze względu na psycho-fizjologiczne właściwości jego percepcji wzrokowej, najbardziej dogodną i pełną formą informacji jest obraz, uzupełniony ewentualnie opisem symbolicznym. Pożądane jest więc, aby komputer był w stanie przyjmować i wydawać wyniki w postaci obrazu. Jest to istotne, zwłaszcza w zastosowaniu komputera do wszelkiego rodzaju projektowania inżynierskiego, w którym głównym wynikiem jest dokumentacja w postaci map, rysunków i wykresów, a jej wykonanie jest najbardziej żmudną i pracochłonną częścią procesu projektowego.

W końcu lat pięćdziesiątych w Massachusetts Institute of Technology (USA) opracowano i dołączono do komputera (w ramach tzw. projektu SCRATCHPAD) kreślak (ang. *plotter*), tj. urządzenie do wykreślania rysunków detali maszyn, a także monitor graficzny – urządzenie do wyświetlania obrazów na ekranie lampy oscyloskopowej. Od tego czasu opracowano i dołączono do komputera wiele różnych urządzeń graficznych, zarówno do wyprowadzania, jak i wprowadzania informacji graficznej. Kreślak stał się najbardziej rozpowszechnionym graficznym urządzeniem komputera, a monitor graficzny umożliwił przejście do nowego, bardziej bezpośredniego sposobu korzystania z komputerów, tj. graficznego dialogu z komputerem. W rozwiązywaniu bowiem niektórych klas zadań nie zawsze jest możliwe dokładne określenie parametrów zadania. Bada się więc różne warianty rozwiązania, wprowadza modyfikacje i uściślenia na bieżąco, na podstawie częściowych wyników. Potrzebna jest więc technika śledzenia procesu rozwiązania zadania na bieżąco i to w sposób dogodny. Potrzebne są również środki efektywnego oddziaływania na ten proces oraz środki do „dialogu z komputerem". Powstała więc i burzliwie rozwija się nowa dyscyplina wiedzy – grafika komputerowa, obejmująca zespół środków sprzętowych i programowych, będących ogniwem dopasowującym dwa odmienne elementy systemu: dyskretny – z zasady działania i wytwarzanych wyników – komputer oraz analogowy – ze swojej natury – człowiek wraz z otaczającym go środowiskiem. Dopasowanie to polega na zamianie dyskretnej postaci informacji, generowanej przez komputer, na postać (analogowych) obrazów, rysunków i tekstów oraz – w stronę przeciwną – zamianie analogowych obrazów i rysunków na postać dyskretną, przed jej wprowadzeniem do komputera i przetwarzaniem.

Obecnie przetwarzanie informacji obrazowych przez komputer jest stosowane powszechnie. Tradycyjnie dzieli się je na trzy części:

- rozpoznawanie obrazów,
- przetwarzanie obrazów,
- grafika komputerowa.

Rozpoznawanie obrazów obejmuje metody tworzenia opisu i klasyfikacji obrazu w ramach określonych klas; polega na zamianie obrazu na abstrakcyjny opis, stanowiący zbiory liczb, symboli lub – graf połączeń pierwotników graficznych, a następnie zakwalifikowaniu obrazu do jednej z wcześniej określonych klas.

Przetwarzanie obrazów polega na wykonywaniu różnego rodzaju operacji na obrazach: usuwaniu zakłóceń na obrazie (np. po jego transmisji), kompresji informacji przed transmisją lub magazynowaniem, wyostrzanie obrazu, polepszanie kontrastu itp.

Grafika komputerowa rozumiana tradycyjnie polega na generowaniu obrazów na podstawie informacji nieobrazowej (opisu matematycznego) i prezentowaniu jej na graficznym urządzeniu wyjściowym komputera. W tym znaczeniu obejmuje ona w pewnym sensie zagadnienia odwrotne do rozpoznawania obrazów. Obejmuje wiele różnych zastosowań. Złożoność programów tworzenia obrazów zależy od rodzaju zadania. Cechą charakterystyczną grafiki komputerowej jest to, że jej programy są z reguły ukierunkowane na rozwiązywanie wąskiego kręgu zagadnień. W tym znaczeniu grafika jest zagadnieniem w pewnym sensie odwrotnym do rozpoznawania obrazów.

Wspólną, w sposób oczywisty, klasą problemów dla wymienionych trzech metod przetwarzania informacji obrazowych przez komputer jest pozyskiwanie, przechowywanie, kompresja oraz struktury danych graficznych. W innych klasach problemów są to zagadnienia uzupełniające się. Na przykład, w przetwarzaniu obrazów często wykonuje się tzw. śledzenie konturów, a w grafice – wypełnianie konturów, będące odwróceniem śledzenia.

Obecnie nazwę „grafika komputerowa" stosuje się często łącznie do tych trzech zagadnień, a tradycyjną grafikę komputerową nazywa się generacyjną grafiką komputerową.

Dr inż. WOJCIECH MOKRZYCKI ukończył w 1967 r. studia na Wydziale Automatyki i Maszyn Liczących Moskiewskiego Instytutu Energetycznego. Pracę zawodową podjął w Katedrze Maszyn Matematycznych Wojskowej Akademii Technicznej, gdzie w 1978 r. otrzymał stopień doktora nauk technicznych. W 1977 r. przeszedł do pracy w Instytucie Maszyn Matematycznych, gdzie organizował i prowadził pracownię grafiki komputerowej. W pracowni tej powstały m. in. sprzętowo niezależny system programowania grafiki komputerów PSG oraz konwersacyjny system graficzny KSG, zaimplementowane w kilku wersjach na komputerach Mera 400 i SM-4. Od 1985 r. reprezentuje PRL w roboczej grupie ds. grafiki komputerowej – organie roboczym Rady Głównych Konstruktorów maszyn SM. Od 1987 r. jest adiunktem w Instytucie Podstaw Informatyki PAN, gdzie zajmuje się metodami grafiki dyskretnej i przetwarzaniem obrazów. Jego dorobek naukowy obejmuje 74 pozycje, w tym 21 prac samodzielnych. Jest laureatem Nagrody Zespołowej I stopnia Ministra Obrony Narodowej w dziedzinie postępu naukowo-technicznego oraz autorem lub współautorem 13 patentów.

POSTAĆ DANYCH OBRAZOWYCH

Przetwarzane przez komputer obrazy dzieli się zwykle na cztery klasy. Podział ten uwzględnia przede wszystkim postać reprezentacji danych obrazowych w komputerze oraz ich przetwarzanie. Postać wizualna obrazu jest w tym wypadku drugorzędna.

- **Klasa 1** obejmuje obrazy naturalne o pełnej gradacji kontrastów i kolorów (typowe obrazy telewizyjne). W komputerze są one przedstawiane jako dwuwymiarowe macierze, zwykle kwadratowe, o rozmiarach rzędu 512 × 512 lub nawet 1024 × 1024 dwuwymiarowych elementów obrazu (tzw. dweli, komórek lub pikseli – ang. pixel) ze sprzężonym z każdym dwelem wektorem pewnych jego cech (kolor, intensywność, przenikalność itp.). Mogą być stosowane również bardziej wyszukane formy reprezentacji. Wzrok ludzki zwykle nie jest w stanie rozróżnić poziomów jaskrawości różniących się mniej niż 1%, wystarcza więc 1 bajt do zakodowania wszystkich cech piksela. Obrazy kolorowe tej klasy często jest dogodniej traktować jako macierze trójwymiarowych wektorów, w których każdy wymiar odpowiada jednemu z trzech podstawowych kolorów: czerwonemu, niebieskiemu oraz zielonemu.

- **Klasa 2** obejmuje obrazy dwuwartościowe (czarno-białe) lub kilkukolorowe. Typowym przykładem obrazów tej klasy jest strona tekstu. W komputerze są one repezentowane graficznie zwykle jako 1-bitowe macierze, a także jako „mapy", ponieważ zawierają jednoznacznie określone obszary jednego koloru.

- **Klasa 3** obejmuje obrazy konturowe. Typowym przykładem obrazów są wykresy oraz kontury obszarów. Dane w komputerze mają zwykle postać ciągu przyrostów (Δ_x, Δ_y) między kolejnymi punktami, ciągu kodów łańcuchowych, w których wektor łącząc dwa kolejne punkty jest określony jednym symbolem ze skończonego zbioru symboli (np. tzw. freemanowski kod ośmiokierunkowy), albo ciągu różnicowych kodów łańcuchowych, gdzie każdy punkt jest reprezentowany przez różnicę między dwoma kolejnymi kodami bezwzględnymi.

- **Klasa 4** obejmuje obrazy punktowe. Obrazy tej klasy są złożone z punktów i wieloboków tak od siebie oddalonych, że nie mogą być reprezentowane przez kod łańcuchowy. Same punkty są reprezentowane przez tablicę ich współrzędnych. Punkty mogą być połączone liniami prostymi lub nieskomplikowanymi krzywymi. Obrazy tej klasy są najczęściej używane w grafice komputerowej, zwłaszcza w opisie obiektów trójwymiarowych, rzutowanych na powierzchnię ekranu. Położenie obiektu określa się wówczas małą liczbą punktów. Rzuty obiektu na ekran pojawiają się jako obrazy klasy 3.

PRZEKSZTAŁCANIE OBRAZÓW

Wiele różnych problemów związanych z przetwarzaniem informacji obrazowej można określić jako przekształcenie między klasami obrazów albo jako przekształcenie obrazów w ramach tej samej klasy. Najczęściej stosuje się następujące przekształcenia:

Segmentacja jest to wydzielanie obszarów w obrazie, mających (w przybliżeniu) jednakowe cechy (takie, jak szarość, jaskrawość itp.). Nazwa ta często odnosi się do procesu, w którym poszukuje się jednolitych cech wśród fragmentów struktur bardziej złożonych. Segmentacja przekształca obrazy klasy 1 na obrazy klasy 2.

Konturowanie polega na odwzorywaniu obszaru w jego granice. Przekształca ono obrazy klasy 2 na obrazy klasy 3.

Ścienianie polega na odwzorowaniu obszaru na tzw. szkielet obrazu, będący jego szkicem. Przekształca obrazy klasy 2 na obrazy klasy 3.

Segmentacja krzywych ma na celu znalezienie tzw. punktów charakterystycznych na konturze. Punktami charakterystycznymi wielokątów są ich wierzchołki. Problemy tego typu są często omawiane w literaturze dotyczącej rozpoznawania obrazów. Segmentacja krzywych przekształca obrazy klasy 3 na obrazy klasy 4.

Interpolacja polega na przeprowadzeniu krzywej gładkiej przez dany ciąg punktów. Przekształca ona obrazy klasy 4 na obrazy klasy 3.

Aproksymacja polega na przeprowadzeniu krzywej gładkiej w pobliżu danego ciągu punktów, z minimalnym błędem średniokwadratowym. Jest przekształceniem obrazów klasy 4 na obrazy klasy 3.

Wypełnianie oraz **cieniowanie** polega na wypełnieniu (zacieniowaniu) obszaru ograniczonego przez kontur, zgodnie z przyjętą zasadą. Jest to jedno z najbardziej typowych zagadnień w grafice komputerowej oraz w przetwarzaniu obrazów. Wypełnianie (i cieniowanie) jest przekształceniem obrazów klasy 3 na obrazy klasy 2.

Rozszerzenie polega na rekonstrukcji obiektu na podstawie jego szkieletu. Jest to również przekształcenie obrazu klasy 3 na obraz klasy 2.

Filtracja obejmuje zwiększenie kontrastu, usuwanie zakłóceń wysokiej częstotliwości itp. Jest przekształceniem w ramach klasy 1 oraz klasy 2.

Filtracja dolnoprzepustowa zastosowana do obrazów kilkukolorowych powoduje uzyskanie płynnego przejścia między kolorami, tzn. poprawia estetykę obrazu. Operacje te są przekształceniem obrazów klasy 2 na obrazy klasy 1.

Rzutowanie polega na przekształceniu obiektu trójwymiarowego na obraz dwuwymiarowy lub – w szczególnym wypadku – dwuwymiarowego przekroju obiektu na jednowymiarową tablicę. Zwykle są stosowane dwa rodzaje rzutów: prostopadły i perspektywiczny. Rzuty prostopadłe, niestety, nie dają realistycznych obrazów, zwłaszcza gdy odległość obserwatora jest porównywalna z rozmiarami obiektu. Rzutowanie perspektywiczne imituje sposób, w jaki obserwator tworzy wrażenie głębi obrazu. Z rzutowaniem są związane inne bardzo ważne zagadnienia eliminowania zasłoniętych linii i powierzchni.

Rekonstrukcja polega na odtworzeniu obiektu lub jego przekroju z dwuwymiarowych rzutów.

Dobieranie i obrazowanie powierzchni jest ważnym zagadnieniem w tych zastosowaniach systemów graficznych, w których istnieje potrzeba oglądania brył z różnych stron, np. w skomputeryzowanej kartografii i geografii, gdzie występują modele terenu w postaci powierzchni opisanych wzorami matematycznymi. W przetwarzaniu i rozpoznawaniu obrazów, powierzchnie są zadawane do analizy co najmniej dwoma sposobami. W jednym traktuje się obraz jako płaszczyznę, uwzględniając poziomy jasności jako 3 współrzędną, w drugim – bierze się pod uwagę powierzchnie występujące w obrazowanym obiekcie.

Usuwanie zasłoniętych linii (grafika konturowa) i powierzchni (grafika rastrowa) w dwuwymiarowym rzucie obiektu trójwymiarowego wiąże się nie ze złożonością zadania, ale z szybkością jego rozwiązania.

Obcinanie jest rozstrzyganiem, czy prosta (lub wielobok) jest przecinana przez inny wielobok. Główne zastosowanie to sprawdzanie, czy określone obiekty mieszczą się w kadrze, oraz przy rozwiązywaniu problemu widoczności obiektów (wzajemnego przecinania się obiektów).

Nadawanie obrazom wyglądu naturalnego polega na użyciu następujących technik: teksturowanie, kreskowanie, modelowanie wielokrotnych odbić i przenikań światła, eliminowanie schodków, ząbków itp.

Zmiana układu współrzędnych obejmuje skalowanie, obroty i przesunięcia. Jest przekształceniem na obiektach w ramach klasy 3 oraz klasy 4.

[1] Autor proponuje nazywać najmniejszy element wyświetlanego obrazu terminem **dwel** (skrót od: dwuwymiarowy element). Wydaje się jednak, że rozpowszechnienie terminu piksel w tym znaczeniu jest już tak duże, że nie uda się go zastąpić inną nazwą (przyp. red.).

Ogólnie przekształcenia z klasy niższej na wyższą są dziedziną rozpoznawania obrazów, a przekształcenia z klasy wyższej na niższą – dziedziną grafiki komputerowej. Przetwarzanie obrazów zajmuje się obydwoma przekształceniami, jak również przekształceniami w ramach każdej klasy. Przykładowo, kompresja jest przekształceniem obrazów klasy 1 na obrazy klasy 2, a wyostrzanie obrazu jest przekształceniem w ramach tej samej klasy.

WPROWADZANIE OBRAZÓW DO KOMPUTERA

Obraz w postaci analogowej, aby mógł być przetwarzany przez komputer, musi zostać przekształcony na postać dyskretną – pewną macierz liczbową. Proces ten nazywa się **dyskretyzacją**, a służące do tego celu urządzenie – **dyskretyzatorem**. Dyskretyzacja składa się z dwóch procesów: próbkowania i kwantowania (porcjowania). **Próbkowanie** polega na pomiarze (analogowych) wartości wektorów cech w pewnym zbiorze punktów obrazu (w pikselach). **Kwantowanie** natomiast polega na zastąpieniu analogowych wartości wektorów cech pikseli najbliższymi im poziomami wartości dyskretnych. Liczba punktów pomiaru (a tym samym liczba pikseli) w jednostce powierzchni określa tzw. **rozdzielczość przestrzenną**, a liczba możliwych poziomów wartości wektora cech obrazu – tzw. **rozdzielczość tonową**.

Graficzne urządzenia wejściowe

Gaficzne urządzenia wejściowe [1] służą do wprowadzania do komputera obrazu bądź danych niezbędnych do wykonywania określonych operacji na obrazie bądź do prowadzenia dialogu graficznego z komputerem. Ze względu na spełniane funkcje urządzenia te można podzielić na:

- **dyskretyzatory,**
- **manipulatory,**
- **pióra,**
- **wybieraki,**
- **taksowniki.**

Do dyskretyzacji różnych klas obrazów są stosowane różne rodzaje dyskretyzatorów.

Dyskretyzatory rastrowe stosuje się do dyskretyzacji obrazów klasy 1 oraz klasy 2. Istnieją dwa typy takich dyskretyzatorów: dyskretyzatory z kamerą telewizyjną oraz dyskretyzatory z tzw. biegającą plamką, służące do prostego przeglądania obrazu. Te drugie można zaprogramować tak, aby „śledziły" granicę między dwoma obszarami obrazu, różniącymi się jaskrawością lub kolorem. Na wyjściu powstaje wówczas obraz klasy 3.

Dyskretyzatory konturowe (stołowe) służą do (ręcznego, półautomatycznego lub automatycznego) wprowadzania rysunków liniowych generując pary współrzędnych (x_i,y_i) punktów, leżących na śledzonej linii.

Dyskretyzatory obiektów są to specjalizowane urządzenia z czujnikiem ślizgającym się po powierzchni obiektu i generującym współrzędne swojego położenia.

Rysownice są rodzajem ręcznych dyskretyzatorów konturowych o niewielkich wymiarach (rzędu A3 do A4) z piórem, zwykle ultradźwiękowym lub napięciowym; zazwyczaj są stosowane do wprowadzania odręcznych szkiców lub plasowania elementów obrazu na ekranie monitora.

Manipulatory są to mechanoelektryczne przekształtniki, służące do plasowania elementów obrazu na ekranie monitora graficznego. Wyróżnia się kilka typów manipulatorów:

- **kulowy** (kula) – przekształtnik z kulą, mającą dwa stopnie swobody ruchu;
- **ramieniowy** (drążek) – przekształtnik z uchwytem, mającym dwa stopnie swobody ruchu;
- **tarczowy** (koło) – przekształtnik z tarczą lub kołem, o jednym stopniu ruchu, może być używany w parze z takim samym urządzeniem sterującym przesunięciami wzdłuż drugiej osi;
- **mysz manipulacyjna** (mysz, taczka, wózek) – przekształtnik o działaniu manipulatora kulowego lub dwóch manipulatorów tarczowych, w którym obroty kuli są uzyskiwane przez posuwanie po (dowolnej) powierzchni.

Pióra są to czujniki (detektory) używane do identyfikowania elementów obrazu na powierzchni obrazowania, (np. powierzchni monitora lub rysownicy). Wyróżnia się **pióra świetlne** (stosowane do współpracy z monitorami ekranowymi) oraz **pióra naddźwiękowe i napięciowe** (stosowane do współpracy z rysownicami).

Wybieraki są to urządzenia służące do wybierania jednej możliwości ze skończonego zbioru. Wybierakiem jest, na przykład, klawiatura funkcyjna.

Taksowniki są to urządzenia do zadawania wielkości skalarnych. Taksownikiem może być np. manipulator tarczowy albo potencjometr.

Podstawowe operacje manipulacyjne na obrazie

W graficznym dialogu z komputerem, za pośrednictwem graficznych urządzeń wejściowych, wykonuje się wiele różnych operacji. Do najważniejszych z nich należą:

- **plasowanie,** czyli wprowadzanie współrzędnych położenia elementu graficznego na powierzchni obrazowania za pomocą manipulatora lub klawiatury alfanumerycznej;
- **trasowanie,** czyli ciąg operacji powodujących podążanie znacznika za poruszającą się głowicą czujnika (np. pióra świetlnego lub lokalizatora), wyznaczającego drogę przebytą przez to urządzenie;
- **wodzenie,** tj. przesuwanie wybranego elementu obrazu wzdłuż ścieżki określonej przez poruszającą się głowicę czujnika (np. pióra świetlnego);
- **śladowanie,** tj generowanie linii ciągłej wzdłuż drogi wyznaczonej przez lokalizator lub czujnik;
- **wybieranie,** tj. identyfikowanie elementu obrazu prezentowanego na powierzchni obrazowania za pomocą pióra lub lokalizatora;
- **wygaszanie,** tj. usunięcie jednego lub większej liczby elementów obrazu bez zmiany pozostałych.

URZĄDZENIA I TECHNIKI OBRAZOWANIA

Są dwa podstawowe typy urządzeń obrazujących: graficzne monitory ekranowe oraz urządzenia do wytwarzania kopii trwałej obrazu, tj. kreślaki, drukarki graficzne i rejestratory mikrofilmowe.

Monitory ekranowe są to urządzenia wyświetlające obraz na lampie elektronopromieniowej. Stosuje się do tego celu **lampy pamięciowe** zachowujące obraz przez długi czas (ok. kilkudziesięciu minut) bez konieczności ponownego wykreślania; **lampy z odnawianiem,** w których proces tworzenia obrazu na ekranie lampy należy powtarzać z określoną częstotliwością, aby obraz był widoczny przez wymagany czas; oraz **lampy z ciemnym śladem,** w których pod wpływem strumienia elektronowego następuje przyciemnienie powierzchni ekranu zamiast jej rozjaśnienia.

Kreślaki są to konturowe urządzenia graficzne, kreślące obraz na nośniku trwałym: papierze, szkle, filmie itp. Rozróżnia się **kreślaki stołowe płaskie,** w których pióro rysuje obraz na stole ruchami wykonywanymi wyłącznie przez głowicę kreślaka, oraz **kreślaki bębnowe,** rysujące na materiale nawiniętym na bęben; głowica takiego kreślaka porusza się równolegle do osi bębna, którego obrót powoduje wykonanie rysunku wzdłuż drugiej osi układu.

Rejestratory mikrofilmowe są to urządzenia do bezpośredniego rejestrowania obrazu generowanego przez komputer na mikrofilmie lub mikrofiszy.

Z procesem odnawiania obrazu są związane dwa terminy: miganie[2)] i mruganie. **Miganie** jest to technika (programowa lub funkcja sprzętowa urządzenia wyświetlającego), polegająca na przemiennym wyświetlaniu i wygaszaniu wybranego elementu

[2)] Zgodnie ze słownikiem [1] Autor proponuje nazywać odpowiednią czynność migotaniem (ang. *blinking*). Ponieważ ten termin jest już zajęty w informatyce, gdyż oznacza tzw. migotanie stron (ang. *threashing*), proponujemy używanie nieco krótszej nazwy – miganie (przyp. red.).

obrazu, używana zwykle w celu przyciągania uwagi użytkownika. **Mruganie** jest niepożądanym miganiem lub pulsacją obrazu na ekranie monitora graficznego; pojawia się wówczas, gdy częstotliwość odnawiania obrazu jest za mała.

Inny, często stosowany podział urządzeń obrazowania, wynika z techniki rozwijania obrazu na powierzchni obrazowania. Zasadniczo stosuje się dwie takie techniki: konturowe rozwinięcie obrazu i rastrowe rozwinięcie obrazu.

Konturowe rozwinięcie obrazu polega na wykreślaniu elementów obrazu w kolejności podawanej przez program komputerowy, przy czym ruch elementu kreślącego obraz jest analogiczny do przesunięć pióra kreślaka stołowego. Technika ta jest podstawą działania tzw. konturowych urządzeń graficznych. Z techniką konturowego rozwinięcia obrazu jest związany termin **lista obrazowa**, oznaczający ciąg rozkazów i danych graficznych, zawartych w buforze obrazu, bezpośrednio opisujących obraz fizyczny (inaczej mówiąc, jest to opis obrazu za pomocą rozkazów graficznych urządzenia obrazującego).

Rastrowe rozwinięcie obrazu polega na punktowym kreśleniu obrazu piksela za pikselem, wzdłuż poziomych lub pionowych linii równoległych, linii radialnych bądź linii spiralnych. Technika ta jest podstawą działania rastrowych urządzeń graficznych.

Konwersja struktury obrazu jest to technika przekształcania ciągu danych opisujących obraz konturowy na obraz rastrowy. Między grafiką konturową a grafiką rastrową istnieje zasadnicza różnica.

Urządzenia grafiki konturowej tworzą obrazy klasy 3 i klasy 4. Wykreślenie obrazu wymaga utworzenia sekwencji rozkazów w rodzaju *„ustaw punkt p(x,y)", „ustaw jaskrawość", „wykreśl wektor w(Δx,Δy)"* itp. W monitorach z lampą wymagającą odnawiania obrazu, ostatnim rozkazem tej sekwencji powinien być rozkaz *„przejdź do początku ciągu"*. Każda realizacja ciągu rozkazów powoduje ponowne wykreślenie obrazu, przy czym częstotliwość odnawiania obrazu jest odwrotnie proporcjonalna do długości pętli. Istnieje górna granica długości pętli, a więc także złożoności obrazu; zależy ona wprost od szybkości działania urządzenia obrazującego.

Urządzenia grafiki rastrowej tworzą obrazy klasy 1 lub klasy 2, ale mogą również być wyposażone w bloki funkcjonalne, umożliwiające prezentowanie obrazów konturowych klasy 3 lub klasy 4. Urządzenia te noszą nazwę **rastrowych urządzeń graficznych** w odróżnieniu od **urządzeń obrazowych**, nie mających tych możliwości. Zasadniczą cechą urządzeń rastrowych jest duża pamięć – zwana buforem obrazu lub pamięcią odnawiającą – z jedną komórką dla każdego adresowalnego punktu ekranu. W pamięci tej są przechowywane parametry wszystkich elementów obrazu, a obraz jest prezentowany przez odczytywanie komórek pamięci i rozjaśnianie odpowiednich punktów ekranu.

W urządzeniach rastrowych istnieje problem eliminowania z obrazu obiektów nakładających się. Rozwiązanie tego problemu polega na podziale pamięci na strefy i obrazowanie każdego obiektu w oddzielnej strefie. Można to zrealizować dwoma sposobami: pierwszy polega na stosowaniu więcej niż jednej kopii ekranu, a drugi – na rezerwowaniu w każdej komórce pamięci obrazu kilku bitów na opisy oddzielnych obiektów.

Główną zaletą grafiki konturowej jest oddzielenie opisu obiektu od opisu ekranu. Dlatego rozdzielczość takich urządzeń jest określona przez elektronikę układów kreślących. Powszechne są, np. w monitorach, ekrany o wymiarach 4096 × 4096 jednostek rastra. Ponadto linie na obrazie są wykreślane techniką analogową, nie ma więc niekorzystnych efektów w postaci schodków.

Urządzenia rastrowe wymagają dokładnego odtworzenia każdego piksela na obrazie, na podstawie zawartości komórki pamięci odnawiającej. Dlatego zwiększenie rozdzielczości obrazu wymaga zwiększenia wielkości tej pamięci. Największe obecnie matryce mają wymiary 1024 × 1024 jednostki rastra. Kontury mogą być wyświetlane jedynie w postaci cyfrowej, co – wraz z ograniczoną rozdzielczością – daje charakterystyczny „efekt schodków". Inną wadą grafiki rastrowej jest mała liczba stref, przy podziale pamięci według adresów lub w ramach słów; umożliwia to przechowywanie tylko niewielkiej liczby obiektów.

LITERATURA

[1] Mokrzycki W., Walkiewicz L.: Słownik pojęć i terminów z dziedziny grafiki komputerowej. Informatyka, nr 2, 3, 4, 8 (1985).

Nowoczesne programowanie w języku C

dokończenie ze s. 18

LITERATURA

[1] Booch G.: Software Engineering with Ada. Benjamin-Commings, 1983
[2] Booch G.: Object-Oriented Development. IEEE Trans. on Software Engineering, Vol. SE-12, No. 2, 1986
[3] Cox B. J.: Message/Object Programming – An Evolutionary Change in Programming Technology. IEEE Software, Vol. 1, No. 1, 1984
[4] Goos G.: Nowoczesne języki wysokiego poziomu a produkcja oprogramowania. Informatyka, nr 9, 1987
[5] Kerninghan B. W., Ritchie D. M.: Język C. WNT, Warszawa, 1987
[6] Shaw M.: Abstraction Techniques in Modern Programming Languages. IEEE Software, Vol. 1, No. 4, 1984
[7] Shaw M.: Poza programowanie wielkoskalowe – kolejne wyzwania dla inżynierii oprogramowania. Informatyka, nr 7–8, 1987
[8] Stroustrup B.: The C++ Programming Language – Reference Manual. Computing Science Technical Report, No. 108, AT&T Bell Laboratories, 1984.

Znajomość systemu operacyjnego przestała już być domeną bardzo wąskiej grupy specjalistów – analityków systemu lub zespołu realizacyjnego. Obecnie napisanie nowego systemu operacyjnego wydaje się dla wielu bardzo proste. A jednak, nowe systemy pojawiają się dość rzadko, częściej – starsze są przez lata udoskonalane i rozbudowywane, lecz zachowują błędy i ograniczenia dzieciństwa.

Zrealizowanie nowego systemu operacyjnego, a dokładniej funkcjonalnie pełnego oprogramowania systemowego dla danego typu komputera, jest jednak przedsięwzięciem wymagającym znacznych nakładów finansowych oraz długiego czasu wykonania. W historii informatyki w Polsce jest kilka przykładów stworzenia całkowicie od podstaw nowego systemu operacyjnego, np. SOM dla minikomputera MERA 400 oraz kilku systemów dla potrzeb akademickich w jednostkach instalacyjnych. Komputery seryjnie produkowane w Polsce były wyposażane w oryginalne systemy opracowane za granicą: ODRA 1300 – w Executive z George'm z ICL 1900, JS-RIAD w system OS i jego mutacje z IBM 360/370, a także SM-3 i SM-4 w system RSX-11 z PDP-11.

Obecnie wraz z rozwojem rynku mikrokomputerowego – opartego głównie na mutacjach IBM PC/XT i IBM PC/AT – rozpowszechnił się system operacyjny MS-DOS (PC-DOS), którego poznanie nie jest trudne. Jednak wraz ze wzrostem mocy obliczeniowej tych mikrokomputerów, spowodowanej zastosowaniem koprocesorów, pojemniejszych pamięci operacyjnych oraz dyskowych, a także – w związku z naciskiem użytkowników na tworzenie systemów wielodostępnych i sieci lokalnych ujawniły się wady tego systemu. Proste rozszerzanie funkcji MS-DOS przez nakładki systemów (jak np. Multilink), nie przyniosło oczekiwanych rezultatów. Pozostaje więc poszukiwanie innych rozwiązań. Jedynie system klasy Unix, coraz popularniejszy na rynku światowym, spełnia większość oczekiwań. Dla mikrokomputerów typu IMB PC możliwe jest, na przykład, wykorzystanie systemów Xenix lub QNX – oferowanych już na rynku polskim.

W aktualnej sytuacji polskiego rynku oprogramowania w relacji do rynku światowego istnieją jednak podstawowe trudności w użytkowaniu tych systemów. Ograniczenia przepływu technologii z Zachodu uniemożliwiają uzyskanie licencji i całkowicie blokują dostęp do kodu źródłowego, powodując tym samym niemożliwość jego lokalnych modyfikacji i rozszerzeń. Trzeba też powiedzieć, że dalsze bazowanie tylko na oprogramowaniu zachodnim, często kopiowanym nielegalnie, prowadziłoby nieuchronnie do dalszego opóźnienia w rozwoju informatyki w Polsce.

Stąd też szczególnej uwadze polecamy Czytelnikom prezentację systemu IPIX – zrealizowanego od podstaw w kraju. Wydaje się, że jest to znaczący fakt w rozwoju naszej informatyki, jeśli oprócz przejmowania produktów zachodnich mamy dostęp do produktów własnych. Popierając te działania, udział w pracach rozwojowych, a także praktyczne zastosowania, możemy mieć nadzieję, że łatwiej będzie pokonywać kolejny dystans w pogoni za postępem. Jednocześnie informujemy Czytelników, że Polskie Towarzystwo Informatyczne wspiera rozwój tego systemu, początkowo przez niezależną ocenę jego jakości, a w przyszłości może również finansowo w realizacji nowych modułów z jego środowiska. (WI)

DANUTA KRUSZEWSKA
Instytut Podstaw Informatyki
Polska Akademia Nauk
Warszawa

System operacyjny IPIX

IPIX jest nazwą systemu operacyjnego dla komputerów typu IBM PC, opracowanego w całości przez Zespół Technologii Oprogramowania Instytutu Podstaw Informatyki Polskiej Akademii Nauk. IPIX jest zgodny ze standardem Unix System V, zalecanym przez międzynarodową grupę X/OPEN (X/OPEN Portability Guide, Issue 2). Zgodność systemu IPIX z systemem Unix polega na ich funkcjonalnej zgodności wywołań i opcji poleceń systemowych oraz na pełnej zgodności wywołań funkcji systemowych i standardowych, zdefiniowanych dla języka C. Użytkownicy systemu Unix, Xenix lub podobnego, mogą więc bez żadnego przygotowania pracować pod nadzorem systemu IPIX. Programy napisane w języku C i działające pod kontrolą tych systemów można przenosić nawet między różnymi typami komputerów – oczywiście tylko w wersjach źródłowych.

IPIX – tak jak Unix – jest systemem uniwersalnym. Jest więc bazą, na której można zbudować dowolne oprogramowanie użytkowe, np. obsługę banków danych, obliczenia numeryczne czy przetwarzanie tekstów. Każdy system zgodny z Unixem ma wiele cech charakterystycznych dla tej klasy systemów operacyjnych.

Wielodostęp

System IPIX jednocześnie obsługuje wiele stanowisk pracy – konsolę (komputer główny) i terminale. Z każdego stanowiska użytkownik ma dostęp do wszystkich zasobów systemu i pracuje tak, jakby komputer i system były wyłącznie do jego dyspozycji.

Terminalem może być komputer typu IBM PC (z dowolnym emulatorem terminala, np. **xansi**, opracowanym specjalnie dla **IPIX-a**) ale może nią również być terminal alfanumeryczny (np. VT100). Terminal jest połączony z komputerem głównym przez łącze szeregowe RS232. W

niedalekiej przyszłości IPIX będzie także pracować w sieci.

Wielozadaniowość

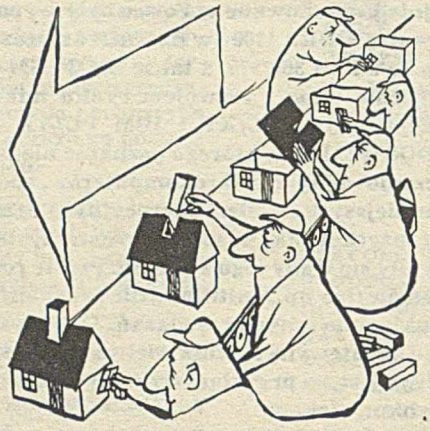
System IPIX umożliwia równoczesne działanie wielu procesorów (zadań) i pozwala wykonywać jednocześnie wiele poleceń systemowych lub użytkowych (każde z nich również może być podzielone na procesy). Wielozadaniowość pozwala na bardzo efektywne wykorzystanie sprzętu, a tym samym zwiększa intensywność jego eksploatacji.

Mechanizm procesów wykorzystano przy tzw. przetwarzaniu w tle – IPIX umożliwia równoczesną pracę wsadową i konwersacyjną. Użytkownik może, na przykład, zainicjować w tle drukowanie wyników i uruchomić kompilator języka C dla kilku modułów programu, a jednocześnie redagować dokument (zanim poprzednie zadania się zakończą).

Mechanizm potokowy

System IPIX oferuje tzw. mechanizm potoków. Potok jest specjalnym rodzajem pliku, umiejscowionym na ogół w pamięci operacyjnej komputera. Jego cechą charakterystyczną jest to, że dane napływające są zapisywane zawsze na końcu pliku, a dane przeczytane – zawsze od początku – są usuwane z pliku.

Potoki najczęściej są wykorzystywane przy tworzeniu nowych poleceń z już istniejących. Taki potok poleceń jest ciągiem, w którym każde poprzednie polecenie przekazuje swoje wyniki bezpośrednio jako dane do następnego polecenia.



Hierarchiczny system plików

System plików IPIX-a jest drzewem o jednym korzeniu. Hierarchiczna struktura skorowidzów umożliwia tematyczne grupowanie plików. Jedną z właściwości tego systemu jest możliwość tworzenia dowiązań do zbiorów danych: jeden zbiór danych (program, tekst itp.) może być reprezentowany przez wiele plików o różnych lub tych samych nazwach w różnych skorowidzach.

Przez system plików IPIX umożliwia użytkownikom jednolitą obsługę wszystkich zasobów systemu: zbiorów danych, urządzeń wejścia i wyjścia, a nawet generatora dźwięków i pamięci operacyjnej.

Ochrony zasobów systemu i użytkowników

Ochronę zasobów systemu IPIX zrealizowano na podstawie mechanizmu prawa dostępu do plików. Superużytkownik (zarządca systemu) ma wszelkie uprawnienia dostępu do wszystkich zasobów systemu. Każdy inny użytkownik musi przedstawić się systemowi (przez podanie hasła) i od tej pory staje się właścicielem swoich plików oraz ma dostęp do tych plików systemowych, które udostępnił mu sam system lub jego zarządca. IPIX obsługuje również grupy użytkowników i tzw. resztę (potencjalni nowi użytkownicy).

Prawa dostępu określają możliwość odczytywania, zapisywania lub wykonywania pliku przez jego właściciela, grupę i resztę użytkowników. W ten sposób, przez niepowołanym dostępem można ochronić plik, skorowidz lub całe poddrzewo systemu plików.

Dlaczego powstał IPIX

To pytanie, na różnych imprezach i przez różnych ludzi, jest zadawane najczęściej. Przyzwyczailiśmy się już do argumentów w rodzaju:

„Chcecie konkurować z MS DOS-em? Nigdy nie zrobicie tak bogatego oprogramowania aplikacyjnego, jakie jest w Polsce dostępne!"

„Przecież jest w Polsce dostępny Xenix – system tej samej klasy – i na dodatek jest również dostępne zachodnie oprogramowanie aplikacyjne, lepiej zrobilibyście jakąś bazę danych!".

A my zwykle odpowiadamy tak.

Po pierwsze nie chcemy konkurować z MS-DOS-em!

●System ten ma rzeczywiście bogate oprogramowanie, ale jest systemem przeznaczonym dla komputerów osobistych (dla inżyniera, ale nie dla całej fabryki; dla sekretarki, ale nie dla całego biura; dla grafika, ale nie dla całego wydawnictwa); nie nadaje się do obsługi naprawdę dużych baz danych w naprawdę dużym przedsiębiorstwie (nawet gdy komputery są połączone w sieć).

●MS-DOS jest nieefektywny, bo „marnuje" możliwości sprzętu, tzn.:
– działa jednozadaniowo, jeśli program czeka na dane z dysku, to w tym czasie procesor odpoczywa, a mógłby wykonywać inne obliczenia (Intel 8086 – ok. miliona operacji na sekundę);
– „widzi" co najwyżej 640 KB pamięci operacyjnej, dlatego stosowanie go na PC/AT z pamięcią 2 MB jest marnotrawstwem pieniędzy (wbrew pozorom pamięć jest droga);
– bez sztuczek „widzi" tylko 32 MB dysku Winchester;
– wszystkie najbardziej znane pakiety, np. graficzne, działają poza systemem (bezpośrednio sięgają do możliwości sprzętu).

●Kolejne wersje MS-DOS-a coraz bardziej przypominają system Unix. Na dodatek, system ten jest w wielu wypadkach użytkowany prawem kaduka – ogromna większość jego użytkowników nie kupiła go oficjalnie.

Po drugie, gdy zaczynaliśmy prace nad systemem IPIX, system Xenix nie był jeszcze tak rozpowszechniony w Polsce, jak dziś.

Mgr DANUTA KRUSZEWSKA ukończyła w 1976 roku studia na Wydziale Matematyki i Mechaniki Uniwersytetu Warszawskiego (kierunek Informatyka). Pracuje w Instytucie Podstaw Informatyki PAN w zespole zajmującym się technologią tworzenia oprogramowania, który opracował i rozwija system operacyjny IPIX. Jest współautorką tłumaczenia książki pt. „Język C" (B. Kerninghan, D. Ritchie, „The C Programming Language") wydanej przez WNT w 1987 r.

Oprócz mechanizmu uprawnień system zapewnia możliwość blokowania dostępu do danych pliku (nawet do jednego bajtu), to znaczy pozwala określić: co, przez kogo i w jaki sposób jest blokowane oraz sprawdzić, czy określony obszar w pliku jest zablokowany i jaki jest rodzaj tej blokady.

Konwersacyjny interpretator poleceń SHELL

Komunikacja użytkownika z systemem odbywa się za pośrednictwem interpretatora poleceń **Shell** – jego zadaniem jest wykonywanie poleceń użytkownika. Do niego także należy (na życzenie użytkownika) definiowanie lub zmiana kierunku przesyłania danych, łączenie poleceń w grupy lub potoki, wykonywanie poleceń w tle itp.

Na ogół Shell wykonuje polecenia w trybie konwersacyjnym, ale umożliwia też użycie proceduralnego języka programowania rozbudowanych plików poleceń wsadowych (skrypty Shella), wykorzystujących inne, już istniejące polecenia. Korzystanie z języka Shella eliminuje w wielu wypadkach konieczność pisania nowych programów.

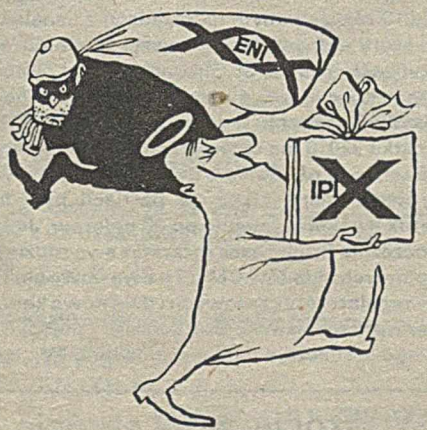
Zestaw poleceń systemowych i użytkowych

System IPIX zapewnia szereg podstawowych usług, których wywołania są na-

Prace nad systemem IPIX są prowadzone w Instytucie Podstaw Informatyki PAN od 1984 roku. Wersja przeznaczona dla komputerów typu IBM PC/XT powstała dzięki zamówieniu Krakowskiej Fabryki Aparatów Pomiarowych MERA-KFAP, która w przeważającej części sfinansowała (ważne!) prace nad systemem przeznaczonym dla produkowanego przez siebie komputera KRAK-86. Wynik tych prac – IPIX – jest rozprowadzany przez MERA-KFAP razem z ich komputerem! IPIX jest również oferowany przez IPI PAN jako samodzielny produkt rynkowy dla użytkowników i producentów innych komputerów zgodnych z IBM PC.

Na samym początku pracy założyliśmy, że musi ona być w całości wykonana w Polsce i że nie będzie polegała na „wypruwaniu metek" (zwrot świetnie ujmujący tendencje na polskim rynku). Pomijając jednak prawne i etyczne aspekty naszej decyzji (w Polsce nie ma skutecznej ochrony prawnej ani oprogramowania zachodniego, ani krajowego), chcemy zwrócić uwagę na jej aspekt czysto pragmatyczny. Dzięki tej pracy jesteśmy zespołem ekspertów panujących nad systemem pod każdym względem. Możemy zatem:

- z powodzeniem instalować IPIX-a na komputerach niekoniecznie dokładnie zgodnych z IBM-owskim pierwowzorem (instalacja systemu Xenix na nietypowym sprzęcie czasami jest w ogóle niemożliwa, a często wymaga uruchomienia własnych programów obsługi niektórych urządzeń zewnętrznych);
- natychmiast reagować na uwagi użytkowników, dotyczące jakości systemu, tzn. usuwać błędy, zmieniać stosowane algorytmy itp. (nie ma przecież programu bez wad, Xenix też ma błędy);
- rozwijać, a przede wszystkim wspomagać rozwijanie systemu w dowolnie wybranych kierunkach zastosowań; IPIX nie musi być uniwersalny; IPIX (lub tylko jego jądro) może być częścią systemu aplikacyjnego zrobionego „pod klucz", a algorytmy IPIX-a możemy dostosować do szczególnych wymagań lub dla zwiększenia efektywności i aplikacji.



Mieszkamy i pracujemy w Polsce – można do nas zatelefonować lub przyjechać, wyjaśnimy wątpliwości, pomożemy (np. w postaci konsultacji czy kursu) w początkowym okresie eksploatacji systemu.

Ostatnia, ale chyba najważniejsza zaleta – możemy przenosić system na inne komputery, niekoniecznie oparte na procesorze Intel. Co więcej – oprogramowanie dla IPIX-a, które do tego czasu powstanie, będzie można bez istotnych problemów przenieść na ten nowy komputer (oczywiście w wersji źródłowej). Ta możliwość stanie się niezwykle istotna w chwili, gdy polski przemysł elektroniczny naprawdę będzie produkował polskie komputery i będzie chciał nimi handlować poza granicami kraju.

Uczestniczymy z IPIX-em w różnych wystawach, demonstrujemy jego możliwości na różnych konferencjach, prowadzimy rozmowy handlowe i dotyczące współpracy z różnymi instytucjami – zarówno państwowymi, jak i prywatnymi, w kraju i za granicą. Udało się nam sprzedać kilka egzemplarzy systemu. Głęboko wierzymy, że nowy komputer Elwro 801 AT najlepiej będzie wykorzystany przez system IPIX. A jednak nadal słyszymy to pytanie: **Dlaczego powstał IPIX?**

W pracach nad systemem IPIX uczestniczyli lub uczestniczą następujący członkowie Zespołu Technologii Oprogramowania: doc. dr Jan Borowiec – Szef Zespołu, mgr inż. Elżbieta Jezierska-Ziemkiewicz, mgr Danuta Kruszewska, mgr Marek Kruszewski, mgr Stanisław Kruszewski, mgr Dariusz Kupiecki, dr Andrzej Łukasiewicz, mgr Władysław Majerski, mgr inż. Mirosław Malicki, mgr Tadeusz Paprzycki, mgr inż. Jacek Surma, dr Jan Walasek – Zastępca Szefa, mgr inż. Tadeusz Wilczek, mgr Julian Winiewski, mgr inż. Andrzej Ziemkiewicz.

Historia Unixa

Z systemami operacyjnymi klasy Unix (IBM/IX, Xenix, QNX) większość informatyków w Polsce zetknęła się w momencie, kiedy MS-DOS miał już ugruntowaną pozycję. Stąd częste przekonanie, że Unix jest produktem późniejszym od MS-DOS-u, co z kolei jest źródłem licznych nieporozumień, ponieważ chronologicznie było dokładnie na odwrót.

Wszysko zaczęło się w 1968 roku, kiedy to w Bell Laboratories (w USA) spotkali się Ken Thompson i Dennis Ritchie. Thompson przybył z Berkeley, gdzie rozwijano wówczas system operacyjny SDS930, zaś Ritchie – z Harvardu, gdzie zajmował się matematyką stosowaną. Grupa badawcza, w której skład weszli, zwróciła się do kierownictwa o środki na odpowiedni komputer do prowadzonych prac, jednakże takich środków nie otrzymała. Po dyskusjach Ritchie i Thompson postanowili wykorzystać dostępny minikomputer PDP-7 i stworzyć dla niego odpowiedni system operacyjny. Był to minikomputer o architekturze zbliżonej do współczesnych mikrokomputerów, ale jego oprogramowanie składało się jedynie z asemblera i programu ładującego. W 1969 roku system operacyjny we wstępnej wersji był gotowy. Miał hierarchiczny system plików, możliwość powoływania zadań współbieżnych (w terminologii UNIXa – procesów) i obsługiwał tylko jednego użytkownika.

Nazwę Unix wymyślił Kerninghan w 1970 roku. Miała to być nazwa ironiczna, utworzona przez analogię do nazwy wielodostępnego systemu Multics, choć jeszcze w tym samym roku Unix potrafił obsługiwać dwóch użytkowników. Początki Unixa były więc bardzo skromne, ale nawet w tej pierwotnej wersji cechowała go niezwykła wprost elegancja rozwiązań – owo specyficzne połączenie prostoty i wyrafinowania, mocy i zwartości. Do dziś Unix jest oceniany w kategoriach tyleż użytkowych co, estetycznych.

System od razu zdobył uznanie innych pracowników Bell Laboratories. Został wykorzystany do różnych prac, przede wszystkim do przetwarzania tekstów.

Dalsza historia Unixa jest następująca:

1971 – przeniesienie na komputer nieco większej mocy, PDP 11/20,

1972 – włączenie mechanizmu potoków,

1973 – Thompson skończył opracowywanie języka C i system został przepisany na ten język,

1975 – pierwsza wersja licencjonowana;

W jakiś czas potem, wykorzystując programy źródłowe w języku C, przeniesiono system na komputer Interdata 8/32, który ma architekturę tak odmienną od serii PDP, iż nie było wątpliwości, że Unix jest systemem naprawdę przenośnym. Potem dokonano licznych innych przeniesień na komputery różnych typów i generacji. Ich podstawą były programy źródłowe w języku C udostępnione przez AT&T. Poza tym powstały liczne systemy klasy Unix lub podobne, realizowane od podstaw. Obecnie istnieje ponad 100 takich systemów. Sama nazwa Unix jest zastrzeżona dla systemów dostarczanych przez AT&T.

1979 – pierwsza licencjonowana wersja handlowa,

1983 – Unix System V, opublikowana norma systemu, ustalona przez AT&T,

1984 – Ritchie i Thompson dostają za Unix doroczną nagrodę Turinga za rok 1983 (przyznawaną przez Association for Computing Machinery).

Od 1981 roku na historię UNIX-a nakłada się historia mikrokomputera IBM PC. Firma IBM zwróciła się do firmy Microsoft o skonstruowanie odpowiedniego systemu operacyjnego dla tego mikrokomputera. Po rozmowach szef firmy Microsoft – Bill Gates – przyjął zamówienie i powstała pierwsza wersja systemu MS-DOS (przez IBM nazywanego PC-DOS). Jednocześnie jednak Gates podjął decyzję, że przyszłym systemem dla IBM PC będzie system klasy Unix. Zwrócił się więc do AT&T i kupił wersję źródłową Unixa; trochę później powstał system Xenix. Nie jest jasne, dlaczego Gates wówczas nie przyjął, że MS-DOS będzie podzbiorem przyszłego Xenixa – choćby bardzo ograniczonym. Tymczasem rozwinęły się komputery IBM PC i odpowiednio firma Microsoft rozwinęła MS-DOS, zresztą przez kolejne włączanie różnych mechanizmów Xenixa. MS-DOS stał się sukcesem rynkowym, ale też hamował popyt na Xenixa. Sam MS-DOS nie może przez kolejne rozszerzenia przekształcić się w Xenix, gdyż nie pozwalają na to wcześniej przyjęte ustalenia.

Przełom nastąpił wówczas, gdy pojawił się PC/AT – mikrokomputer, który może być w pełni wykorzystany przez Xenix lub system tej klasy. Pojawiły się jednak trudności z innej strony. Microsoft włączył do swojego Xenixa kilka istotnych rozszerzeń (blokowanie rekordów, pamięć współdzielona, semafory itd.). Firma AT&T włączyła te rozszerzenia do swojej normy Unix System V, ale zaimplementowała je w odmienny sposób. Biorąc pod uwagę potęgę AT&T, nie było to wygodne dla Microsoftu. Ostatecznie Gates zdecydował, że należy doprowadzić do zgodności Xenixa z Unixem (System V) i taką wersję opracowała firma współpracująca z Microsoftem – Santa Cruz Operation.

W 1986 roku powstała organizacja X/OPEN, grupująca znaczące przedsiębiorstwa komputerowe, której celem jest zalecanie istniejących norm w dziedzinie oprogramowania. Jako normę systemu operacyjnego zaleca się Unix System V, a dla języka C – normę ANSI. Inne zalecenia odnoszą się do baz danych, operacji ekranowych itd. Unix jest pierwszym systemem objętym normalizacją w tak szerokim zakresie. **(DK)**

zwane poleceniami systemowymi. Należą do nich m. in. polecenia obsługujące system plików, wspomagające powstawanie nowych poleceń, zarządzające wykonywaniem uruchomionych zadań itp.

Część usług systemu jest realizowana przez programy użytkowe odgrywające rolę typowych programów usługowych, np. ekranowy edytor tekstów (tworzenie plików tekstowych), obsługa plików tekstowych (porównywanie, sortowanie itp.) czy obsługa plików w formacie systemu MS-DOS. Przez polecenie tego rodzaju rozumie się program (lub skrypt Shella) napisany przez użytkownika. Z punktu widzenia systemu IPIX nie ma żadnej różnicy w traktowaniu poleceń systemowych i programów użytkowych.

W systemie IPIX takich poleceń jest ok. 100 (standard zawiera ok. 120 opisów poleceń, przy czym niektóre polecenia nie są obowiązkowe).

Kompilator języka C

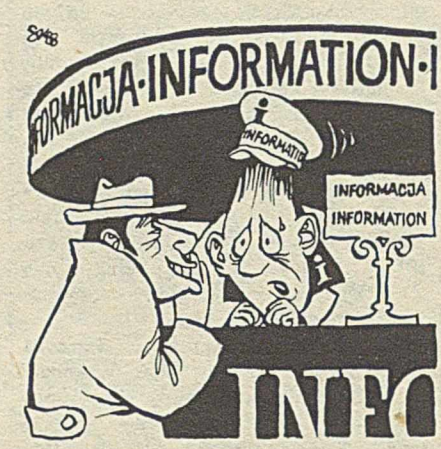
Głównym językiem programowania w systemie IPIX jest język C. Jego kompilator akceptuje pełny język C według raportu: B. W. Kerninghan, D. M. Ritchie, The C Programming Language, Prentice-Hall, 1982 (polskie tłumaczenie: Język C, WNT, 1987). Kompilator tworzy program według jednego z czterech modeli pamięci. Standardowa biblioteka funkcji wejścia i wyjścia (dla każdego modelu pamięci) jest całkowicie zgodna ze standardem Unix System V.

* * *

Jednocześnie w kilku ośrodkach prowadzi się intensywne prace nad rozwojem systemu IPIX. Celem tych prac jest rozszerzenie i wzbogacenie możliwości systemu. Wród nich na szczególną uwagę zasługują:

- nowa dedykowana wersja IPIX dla komputerów opartych na procesorze Intel 80386,
- kompilator języka Pascal,
- oryginalny asembler o składni języka wysokiego poziomu,
- pakiety funkcji obsługujących ekran i urządzenia graficzne,
- pakiety funkcji wspomagających zarządzanie bazą danych (ISAM),
- język SQL dla tworzenia relacyjnych baz danych.

Ponadto, z planowanych prac warto wymienić opracowania kompilatorów innych nowoczesnych języków programowania, jak Loglan i Prolog oraz narzędzi do pracy w sieci.



Rysunki SZYMON KOBYLIŃSKI

W skrócie ● W skrócie ● W skrócie ● W skrócie

□ Zespoły badawcze firmy IBM i AT&T mają pracować w ośrodku badawczym IBM znajdującym się w La Gaude we Francji. Ich celem jest określenie zapotrzebowania rynku europejskiego na inteligentne usługi sieciowe, obejmujące również sieci prywatne.

*

□ Oddział systemów testujących (Sentry Test Systems Division) przedsiębiorstwa Schumberger Ltd. oferuje nowy tester kostek pamięci dynamicznej CMOS. Tester ten, o nazwie S90, może badać jednocześnie 32 kostki z częstotliwością 50 MHz (dotychczasowe testery mogły kontrolować maksymalnie 16 kostek równolegle). Jest on przystosowany do układów o submikronowej strukturze, takich jak kostki pamięci statycznej o pojemności 256 K bitów i dynamicznej o pojemności 1 M bitu. Ma cztery głowice pomiarowe, z których każda może sondować 8 mikroukładów dynamicznych lub cztery statyczne. W zależności od konfiguracji cena systemu waha się od 250 do 600 tysięcy dolarów, co oznacza 19 375 dolarów za jedno stanowisko testujące, a więc taniej niż w dotychczasowych testerach.

*

□ Firma Texas Instruments Inc. ogłosiła parametry swego procesora sygnałów cyfrowych – TM9320C10. Wykonuje on 6,4 miliona rozkazów na sekundę, a jego część związana z układami CMOS działa z maksymalną częstotliwością zegara 25,6 MHz i jest w pełni zgodna ze standardowym procesorem 32010 realizowanym na układach z kanałem n, który jest o 25% wolniejszy. Mimo dużej szybkości kostka CMOS ma typowy pobór prądu tylko 35 mA, a więc pobór mocy wynosi tylko piątą część tego, co w układzie 32010. Kostka 32010 ma czterdziestokońcówkową, plastikową obudowę z dwurzędowymi wyprowadzeniami (DIL). W partiach po 100 sztuk jest sprzedawana po 60 dolarów. Jednocześnie firma Texas rozszerza swą rodzinę liniowych układów CMOS o dwa nowe układy regulatorów czasowych i dwa nowe komparatory różnicowe. Oba działają z pojedynczego zasilacza o napięciu zaledwie 1V.

W skrócie ● W skrócie ● W skrócie ● W skrócie

WŁADYSŁAW M. TURSKI
Instytut Informatyki
Uniwersytet Warszawski

POLSKIE TOWARZYSTWO INFORMATYCZNE

Kanoniczny krok procesu programowania (2)

Rozważania w pierwszej części artykułu były dość abstrakcyjne, zilustrujemy je teraz w miarę konkretnym przykładem.

PRZYKŁAD

Zadanie polega na zrealizowaniu relacji równoważności (w przykładzie wykorzystano wyprowadzenie algorytmu Fishera-Gallera opisane przez Corella [2]).

Na pewnym skończonym zbiorze **Z** elementów ponumerowanych od **1** do **n** określono relację równoważności, **equiv. E** jest zbiorem kolekcji zawierających pary elementów, o których wiadomo, że należą do relacji **equiv.**

Specyfikacja zadania obejmuje język specyfikacji:

INIT: → **E**
ENTER: **E** × **Z** × **Z** → **E**
EQUIV: **E** × **Z** × **Z** → {true, false}

i, j, k, l – zmienne typu **Z** (ich wartościami są numery elementów)
e – zmienna typu **E** (wartościami **e** są kolekcje par równoważnych) i aksjomaty:

EQUIV (INIT, i, j) ⟨=⟩ (i = j)
EQUIV (ENTER (e, i, j), k, l) ⟨=⟩ (EQUIV (e, k, l) (A1)
v (EQUIV (e, i, k) EQUIV (e, j, l))
v (EQUIV (e, i, l) ∧ EQUIV (e, j,k)) (A2)

Aksjomaty pozalogiczne (A1) i (A2) łącznie z rachunkiem predykatów pierwszego rzędu wyznaczają teorię relacji **equiv.** Posługując się tymi aksjomatami, można np. udowodnić, że relacja ta jest symetryczna:

EQUIV (i, j) ⟨=⟩ EQUIV (j, i)

zwrotna:

EQUIV (i, i) = true

i przechodnia:

EQIV(i,j) EQUIV(j,k)⇒EQUIV(i,k).

Zauważmy, że jeśli explicite nie uznaliśmy żadnych par za równoważne (tj. nie wykonaliśmy operacji **ENTER**), to jedynymi parami spełniającymi **EQUIV (i, j)** są pary **i = j.** Jeśli natomiast konkretna kolekcja zawiera np. pary **(1, 2)** i **(2, 7)** to spełnione są warunki:

EQUIV (2, 1) – na mocy symetrii
EQUIV (1, 7) – na mocy przechodniości
EQUIV (1, 2) – na mocy umieszczenia w **E**
EQUIV (2, 7) – na mocy umieszczenie w **E**

Tyle o teorii (specyfikacji) **S.**

Za nowy język **J1** weźmiemy język zawierający **INIT** i **ENTER** z **J0** oraz nową funkcję:

REP: E × Z → Z

a za aksjomaty pozalogiczne przyjmujemy:

REP (INIT, i) = i (A11)
REP (ENTER (e, i, j), k) =
if REP (e, j) = REP (e, k)
then REP (e,i) (A12)
else REP (e, k)

Użycie w zapisie aksjomatu (A12) notacji **if... then... else** nie zmienia charakteru logiki, jest to czysto stenograficzny skrót notacyjny.

Dla ułatwienia interpretacji **S** w **S1** rozszerzamy język **J1** o predykat **EQUIV** i dodajmy definicję:

EQUIV (e, i, j) ⟨=⟩ (REP (e,i) = REP (e, j))

Interpretację ustalamy jako tożsamościowe przekształcenie symboli **J0** na symbole rozszerzonego **J1**, tj.

I (INIT) = INIT
I (ENTER) = ENTER
I (EQUIV) = EQUIV.

Należy teraz sprawdzić spełnienie warunku **(W)**[1], tj. udowodnić, że (A1) i (A2) – po zinterpretowaniu! – wynikają z (A11) i (A12). Dowód (A1) jest banalnie prosty:

(A1) : EQUIV (INIT, i, j) ⟨=⟩ (i = j)
{interpretując} :(REP (INIT, i) = REP (INIT, j) ⟨=⟩ (i = j)
{na mocy (A11)}: (i = j) ⟨=⟩ (i = j) QED

Dowód (A2) jest sporo dłuższy, ale i on nie przedstawia żadnych istotnych trudności.

Następny krok polega na przejściu od języka zbiorów (którym posługiwaliśmy się dotąd) do języka drzew, pozwalającego nadać zbiorowi par równoważnych strukturę przydatną do algorytmicznego przetwarzania. Składnię tego języka definiujemy tak:

LINK: **E** × **Z** × **Z** → **E**
FATHER: **E** × **Z** → **Z**
ROOT: **E** × **Z** → {true, false}
INIT: → **E**

pozalogiczne aksjomaty mają zaś postać:

ROOT (INIT, i) = true (A21)
ROOT (LINK (e, i, j), j) = false (A22)
ROOT (LINK (e, i, j), k) ∧ (j ≠ k) ⟨=⟩ ROOT (e, k) (A23)
FATHER (LINK (e, i, j), k) = **if** j = k **then** i **else** FATHER (e, k) (A24)

Aby wygodnie zinterpretować symbole języka **J1** rozszerzamy **J2** o definicje:

ENTER (e, i, j) ⟨=⟩ **if** REP (e, i) = REP (e, j) **then** e
else LINK (e, REP (e, i), REP (e, j))
REP (e, i) ⟨=⟩ [t: = i;
while ROOT (e, t) = false
do t: = FATHER (e, t) **od**;
result t]

W drugiej definicji użyliśmy aparatu wykraczającego poza rachunek predykatów pierwszego rzędu: logika **L2** musi być na tyle silna, by można było w niej wnioskować o własnościach wyrażeń zawierających konstrukcje programowe. Korzystając z takiej właśnie logiki musimy udowodnić, że w podanej interpretacji języka **J1** aksjomaty (A11) i (A12) wynikają z (A21)-(A22). Dowody te można znaleźć w cytowanej pracy [2]. (Na pełną realizację algorytmu Fischera-Gallera składa się jeszcze jeden krok, w którym drzewa zastępuje się tablicami: nie wnosi on jednak nic nowego do ilustracji metody postępowania). [Koniec przykładu]

[1] Z pierwszej części niniejszego artykułu

UWAGI OGÓLNE

Wyidealizowany przebieg procesu konstruowania oprogramowania zgodnie z wyłożoną (w zarysach) metodą można przeto przedstawić tak:

$$S0 \rightarrow S1 \rightarrow \ldots \rightarrow Si \rightarrow S(i+1) \rightarrow \ldots \rightarrow Sn$$
$$\downarrow P1 \quad \downarrow Pi \; \downarrow P(i+1) \quad \downarrow Pn$$
$$S1' \quad Si' \; S(i+1)' \quad Sn'$$

gdzie **Si, i = 0,1,...,n,** są kolejnymi teoriami (specyfikacjami) poziomów językowych, a **Pi, i = 1,2,...,n** – złożeniami rozszerzeń realizujących **S(i – 1)'** na poziomie **Si (S0' = S0).**

Warto zwrócić uwagę na dwie sprawy o pierwszorzędnym znaczeniu pragmatycznym:
1) ponieważ wykonanie każdego kroku wymaga przeprowadzenia dowodów spełnienia warunków (**W**), kroki są tym łatwiejsze, im mniej się zmienia z kroku na krok;
2) natura wszystkich kroków jest taka sama.
Można więc spodziewać się, że postępowanie zgodne z opisaną metodą będzie liczyć bardzo wiele kroków jednakowych co do schematu postępowania i natury zobowiązań dowodowych.

Na tym właśnie polega główna różnica między opisywaną i wszystkimi poprzednio znanymi metodami. Zamiast kilku wyodrębnionych etapów, różniących się swym charakterem, występuje tu ciąg wielu jednorodnych kroków. Praktycznie zysk ze stosowania nowej metody polega przede wszystkim na tym, że instrumentarium narzędzi pomocniczych nie musi być zróżnicowane (inne dla każdego etapu). Przyjmując, że jakość i kompletność instrumentarium są proporcjonalne do poniesionych na nie nakładów, bez trudu dostrzegamy wielką szansę lepszego wyposażenia programistów pracujących wedle opisanej metody: przy takich samych nakładach instrumentarium wspomagające wykonywanie kroków kanonicznych będzie tyle razy lepsze, ile różnych etapów zostało wyeliminowane. (Lub odwrotnie, podobnej jakości instrumentarium pomocnicze można osiągnąć tylekroć taniej!)

Na wstępie wspomnieliśmy, że wszystkie znane problemy budowy oprogramowania można przedstawić jako kombinacje kroków kanonicznych i operacji wycofywania się. Przedstawiony powyżej schemat postępowania nazwaliśmy wyidealizowanym dlatego, że nie występuje w nim operacja wycofywania. Skąd bierze się potrzeba wycofywania? Ma ona dwa źródła:
1) ukończywszy pewien krok dochodzimy do wniosku, że nie możemy ustalić rozsądnego następnego poziomu językowego (np. na żadnym poziomie, który można osiągnąć z bieżącego, nie potrafimy zrealizować bieżącej specyfikacji); nasze postępowanie konstrukcyjne zaszło w ślepy zaułek;
2) klient zgłasza modyfikację specyfikacji, której to modyfikacji nie potrafimy wyrazić jako rozszerzenia bieżącej specyfikacji (możliwość takiego dezyderatu wynika stąd, że w porządnie kierowanym procesie programistycznym sprawdzamy słuszność każdej lub przynajmniej prawie każdej specyfikacji pośredniej).

W obydwu wypadkach istnieje radykalne lekarstwo: powrót do specyfikacji pierwotnej i rozpoczęcie całego postępowania od początku. Aczkolwiek na pewno skuteczne, jest to jednak lekarstwo bardzo drogie, gdyż powoduje zmarnowanie całej dotychczas wykonanej pracy. Dlatego też oszczędniej będzie wycofywać się krok po kroku, aż do napotkania pierwszego takiego poziomu językowego, na którym można będzie zaradzić powstałemu kłopotowi. Samo wycofywanie się krok po kroku nie przedstawia żadnych trudności, naturalnie jeśli zawczasu zadbaliśmy o stworzenie odpowiednich po temu warunków (np. zapisując całą historię danego projektu). Ocena, czy wycofaliśmy się dostatecznie daleko, zależy od natury problemu. W pierwszym z wymienionych wypadków ocena ta jest kwestią umiejętności programisty (podobnie jak wszystkie decyzje co do wyboru poziomów językowych!). W drugim, możemy polegać na tym, czy żądaną modyfikację można wyrazić na tym poziomie, do którego już się wycofaliśmy (ewentualnie, czy dany poziom da się niesprzecznie odpowiednio poszerzyć).

Po nawrocie i ustaleniu poziomu językowego, od którego zaczynamy nową próbę dotarcia do zadanego poziomu ostatecznego, kolejne kroki mogą (ale nie muszą) powtarzać decyzje co do wyboru poziomów językowych. Ponieważ należy się liczyć z tym, że konieczność wycofywania się wystąpi wielokrotnie, rzeczywisty proces konstruowania oprogramowania odbiega od wyidealizowanego, liniowego schematu przedstawionego powyżej. Faktycznie mamy do czynienia z dwoma ciągami kroków kanonicznych: z historycznym następstwem kroków rzeczywiście wykonanych (jeśli rezygnujemy z reprezentowania nawrotów ciąg ten przypomina drzewo o gałęziach nierównej długości – jeden liść jest ostateczną realizacją, pozostałe odpowiadają krańcom ślepych zaułków) oraz z czysto liniową sekwencją kroków, które doprowadziły do ostatecznej realizacji (jest to, oczywiście, jedna z gałęzi drzewa).

Do różnych celów praktycznych potrzebne są oba ślady procesu konstruowania oprogramowania. Dokumentacja użytkowa powinna, na przykład, zawierać opis ciągu kroków prowadzących do ostatecznej realizacji, natomiast dokumentacja techniczna, przeznaczona dla dalszej pielęgnacji oprogramowania – całe drzewo, gdyż nigdy nie wiadomo, do jakiego poziomu trzeba będzie wrócić, aby zaspokoić przyszłe żądania klienta.

Ponieważ – podkreślmy to jeszcze raz – metoda kroków kanonicznych wymaga wykonania bardzo wielu drobnych kroków, a stosowanie operacji wycofywania się bardzo komplikuje strukturę następstwa kroków, wydaje się zupełnie niemożliwe opanowanie całości procesu bez odpowiednio silnych środków wspomagających, których przeznaczeniem jest archiwizowanie poszczególnych kroków. Nie może to być byle jakie archiwum, powinno ono maksymalnie ułatwić rekonstrukcję dowolnego minionego stanu procesu konstruowania oprogramowania.

Tak więc widzimy, że stosowanie niesłychanie prostej (ideowo!) metody kroków kanonicznych wymaga bardzo silnego środowiska programistycznego, na które składają się dwa, prawie niezależne elementy: instrumentarium wspomagające wykonywanie jednego kroku (przeniesienia specyfikacji z jednego poziomu językowego na drugi) i baza danych (archiwum wykonanych kroków) wraz z odpowiednio dobranymi mechanizmami „nawigacji", pozwalającymi rekonstruować minione stany procesu konstruowania. Obydwie składowe środowiska są całkowicie uniwersalne, nie zależą od semantyki rozwiązywanego zadania, obydwie też, niestety, są bardzo skomplikowane i wymagają bardzo silnych środków obliczeniowych. Nie należy się łudzić, że środowiska tego rodzaju będą dostępne w sprzedaży wysyłkowej, ani że będą mogły funkcjonować na komputerach z wąską magistralą albo wolną pamięcią niesekwencyjną o pojemności kilkunastu megabajtów.

Powstaje wobec tego pytanie, czy cała metoda ma sens? Odpowiedź zależy od tego, co jest przedmiotem naszych zainteresowań.

Po pierwsze, opisana metoda ma pewne znaczenie poznawcze, gdyż redukuje całość procesu konstruowania oprogramowania, od wstępnej specyfikacji do ostatecznej implementacji i następującej po niej pielęgnacji funkcjonującego systemu, do kombinacji jednorodnych kroków. Pozwala to lepiej poznać naturę atomów działalności programistycznej.

Po drugie, aczkolwiek mało użyteczna dla amatorów i nawet profesjonalistów korzystających z „gotowców", opisana metoda ma sporo zalet dla konstruktorów dużych jednostkowych systemów budowanych od podstaw.

(Mimo wszystkich zalet standaryzacji i powielanego oprogramowania użytkowego, zapotrzebowanie na jednostkowe oprogramowanie stanowi – i zapewne będzie stanowić – dominujący aspekt rynku oprogramowania: każdy nowy port lotniczy, każda duża korporacja, każdy bank i każdy rodzaj usług informacyjnych wymagają własnego, unikatowego oprogramowania, a są to systemy, których ceny kontraktowe wynoszą dziesiątki milionów dolarów za sztukę.)

ZASTOSOWANIA METODY

Koncepcja budowy oprogramowania metodą kroków kanonicznych legła u podstaw dwu poważnych przedsięwzięć programistycznych realizowanych przez londyńską firmę software'ową IST (Imperial Software Technology). Są to projekty **ISTAR** i **GENESIS.**

Projekt GENESIS [1], częściowo finansowany przez EWG-owski program badawczy Esprit i prowadzony wspólnie z laboratoriami Philipsa w Holandii, dotyczy mechanizmów jednego kroku przekształceń formalnych między dwoma poziomami lingwistycznymi. Obejmuje on pokaźny zestaw narzędzi programistycznych ułatwiających prezentowanie specyfikacji i ich rozszerzeń (różnego rodzaju edytory i weryfikatory syntaktyczne, a także programy wspomagające badanie niesprzeczności), dowodzenie poprawności realizacji itp. Cechą charakterystyczną projektu GENESIS jest brak uprzywilejowanej logiki: narzędzia programistyczne są tak zaprojektowane, by mogły być stosowane z różnymi logikami (konkretną logikę zadaje się jak „parametr"), nawet wtedy, gdy logika poziomu językowego specyfikacji jest inna niż logika systemu językowego realizacji. Prototypowa wersja GENESIS została oddana do użytku latem 1987 r.

Projekt ISTAR [3] został zrealizowany na zamówienie British Telecom – niegdyś państwowego, dziś zdenacjonalizowanego brytyjskiego koncernu telekomunikacyjnego. Pierwsza wersja została uruchomiona w centrum programistycznym British Telecom na przełomie lat 1985–1986, po czym firma IST podjęła sprzedaż ISTAR-ów na wolnym rynku, klientom tak różnym, jak: jeden z największych światowych producentów układów scalonych, wielkie przedsiębiorstwo poszukiwań geologicznych, koncern lotniczy i ministerstwo spraw zagranicznych pewnego niewielkiego, ale bardzo aktywnego państwa. Rozrzut tematyczny instytucji kupujących ISTAR-y świadczy o uniwersalności systemu, inaczej powiedziawszy, dowodzi jego całkowitej neutralności względem semantyki konkretnego oprogramowania. Trzeba jeszcze dodać, że ISTAR jest dostarczany zarówno w wersjach indywidualnych (jako części wyposażenia osobnych stanowisk pracy), jak też jako system wielodostępny, działający na kilku, a nawet kilkudziesięciu stanowiskach pracy (do pięćdziesięciu). (Dla ciekawych: wersja wielodostępna kosztuje 0,5 miliona funtów +50 tys. za każde obsługiwane stanowisko, oczywiście bez kosztu sprzętu; istnieje też wielodostępna wersja edukacyjna – za 300 tys. funtów, bez względu na liczbę stanowisk).

ISTAR jest zespolonym środowiskiem programistycznym przeznaczonym do wspomagania pracy programistów realizujących duże zadania z dziedziny konstruowania oprogramowania. Jednocześnie ISTAR wspomaga kierowników zespołów programistycznych w zakresie planowania pracy, bieżącego nadzoru i kontroli, kosztorysowania, fakturowania i sprawozdawczości. Umiejętne korzystanie z ISTAR-a pozwala też znacznie uprościć prace pomocnicze, w tym – właściwe dokumentowanie przedsięwzięć programistycznych. Całą menedżerską i dokumentacyjną stronę ISTAR-a zmuszeni jesteśmy tutaj pominąć, aczkolwiek pomysłowe wykorzystanie odnośnych możliwości (stworzonych przez podstawowy model krokowy) niewątpliwie przyczyniło się do handlowego sukcesu tego środowiska.

Podstawową jednostką działalności programisty korzystającego z ISTAR-a jest krok. Na wykonanie każdego kroku zawierany jest kontrakt pomiędzy zleceniodawcą i zleceniobiorcą (por. [4]). Kontrakt zawiera opis zadania kroku i formułuje kryteria jego odbioru, ustalając jednocześnie, kto ma dokonać odbioru (rolę tę zazwyczaj odgrywa sam zleceniodawca, ale może on wskazać dowolnego innego „agenta"). Kontrakt wskazuje też przewidywaną datę ukończenia kroku, tj. spełnienia kryteriów odbioru wykonanego zadania (kontrakt zawiera też inne informacje wykorzystywane przez menedżerskie programy ISTAR-a).

Zleceniobiorca może przystąpić do wykonywania kroku albo rozpisać zadanie na dwa lub więcej kroków. W tym wypadku staje się on zleceniodawcą, zawierającym kontrakty na wykonanie tych kroków. W trakcie wykonywania kroku zleceniobiorca zawsze może sformułować „podzadanie", które zleci – jako odrębny krok kontraktowy – do wykonania innemu zleceniobiorcy. (Nic zresztą nie stoi na przeszkodzie, by zawierać kontrakty na podzadania ze sobą samym: ISTAR wymaga tylko, by każdy krok miał ściśle określonego wykonawcę, by na jego realizację był zawarty legalny kontrakt i by przewidywane terminy realizacji spełniały naturalne zależności ścieżek krytycznych).

Przystępując do wykonywania kroku, zleceniobiorca pobiera świeży egzemplarz jednego z wielu warsztatów ISTAR-a. Warsztatem (ang. *workbench*) nazywa się zespół powiązanych ze sobą środków programistycznych. Warsztat pascalowy, na przykład, składa się z edytora syntaktycznego, kompilatora, programów wspomagających analizę wykonywania programów w Pascalu, optymalizatorów kodu wynikowego, edytora wydruków (ang. *pretty printing*) itd. Obecna wersja ISTAR-a zawiera warsztaty Pascala, Chilla, Ady, Prologu, SDL-u, Cobolu, kilku baz danych oraz VDM-u; liczba warsztatów jest stale zwiększana. Kontrakt na wykonanie kroku może ograniczyć swobodę wyboru warsztatu; również wybór warsztatu może nałożyć odpowiednie ograniczenia na kontrakty zawierane na ewentualne podzadania.

Informacja o wybranym warsztacie jest na trwałe związana z krokiem; każdy nawrót do tego samego kroku automatycznie otwiera ten warsztat i to w stanie, w jakim był, kiedy ostatni raz pracowano nad danym krokiem.

Praca nad realizacją kroku może zająć jedną lub wiele sesji; przerywając pracę tj. kończąc sesję, programista zamyka warsztat, wznawiając pracę – otwiera go, a ISTAR dba o to, by warsztat był w takim samym stanie jak na końcu poprzedniej sesji.

Całość informacji generowanej w trakcie kroku jest gromadzona przez archiwum ISTAR-a, wykonawca kroku może zażądać zniszczenia części (a nawet całości) tej informacji, ale kontrakt może tę swobodę silnie ograniczyć. W żadnym zaś wypadku nie można zniszczyć skrupulatnie odnotowywanej informacji o zleceniach niszczenia informacji. Kontrakt może też przyznać zleceniodawcy prawo niszczenia informacji wygenerowanej w czasie pracy zleceniobiorcy. Podobne reguły dotyczą praw inspekcji infomacji powstającej w trakcie kroku.

Po ukończeniu kroku jego wyniki są poddawane odbiorowi. Po pomyślnym odbiorze wyniki stają się własnością zleceniobiorcy, ale archiwum ISTAR-a nadal przechowuje informację o zakończonym kroku, pozwalającą na jego pełną rekonstrukcję.

Powyższe, bardzo pobieżne wyliczenie niektórych głównych funcji ISTAR-a (pominięto, na przykład, nader rozbudowane środki obrazowania globalnego postępu prac i wszelkiego rodzaju transformacje międzywarsztatowe) pozwala zorientować się co do zakresu usług środowiska. Warto dodać, że ISTAR istnieje w postaci metasystemu, co pozwala generować ISTAR-y na różne komputery i ich konfiguracje.

LITERATURA

[1] Bustany A. et al.: Overall Requirements for GENESIS (project 1041). IST, London, March 1986

[2] Correl C.H.: Proving Programs Correct Through Refinement. Acta Informatica, Vol. 9, pp. 121–132, 1978

[3] ISTAR: Integrated Project Support Environment – Technical Overview, IST, London, August 1985

[4] Turski W. M.: Software Quality Control – Techniques and Users' Criteria. Pp. 37–43, Constructing Quality Software (P. G. Hibbard, S. A. Schuman, Eds.), North-Holland, Amsterdam, 1978.

Wymagania stawiane bazom danych w biurach

dokończenie ze s. 14

LITERATURA

[1] Chou H.-T. et al.: Design and Implementation of the Winconsin Storage System. Software – Practice and Experience, Vol. 15, No. 10, pp. 943–962, 1985

[2] Dittrich K. R. et al.: An Event/Trigger Mechanism to Enforce Complex Consistency Constraints in Design Databases. ACM SIGMOD RECORD, Vol. 15, No. 3, pp. 22–26, 1986

[3] Haerder T., Reuter A.: Architektur von Datenbanksystemen für Non-Standard-Anwendungen. Pp. 253–286, Blaser A., Pistor P. Eds.), Datenbanksysteme für Büro, Technik und Wissenschaft. GI-Fachtagung, Karlsruhe, Informatik-Fachbericht 94, Springer Verlag, 1985.

FRANK FOERSTERLING
Instytut Informatyki i Techniki Obliczeniowej
Akademia Nauk NRD
Berlin

Wymagania stawiane bazom danych w biurach (2)

W pierwszej części artykułu skoncentrowano się na podstawowych zagadnieniach dotyczących modelowania biura i biurowych systemów informacyjnych (BSI). Druga część jest poświęcona głównemu składnikowi BSI, tj. bazie danych, oraz wielonośnikowym BSI z bazą danych.

WYMAGANIA SZCZEGÓŁOWE

Poniżej przedstawiono wymagania na struktury danych, operacje, tzw. wyzwalacze, transakcje i wymagania czasowe.

Struktury

Typy danych. W systemach zarządzania bazą danych przeważnie są określone tylko elementarne typy danych, takie jak liczby całkowite, wartości logiczne i napisy. Ich wadą są ograniczenia nakładane na długość. Przykładowo, w systemie dBase maksymalna długość pól przeznaczonych na napisy wynosi 255 znaków. Zastosowania biurowe wymagają zaniechania ograniczeń długości. Niesformatowany (tj. bez wprowadzonej struktury) dokument tekstowy może bowiem zajmować tysiące i więcej bajtów. W systemie wielonośnikowym system zarządzania bazą danych musi być zdolny do posługiwania się nowymi (niesformatowanymi) typami danych, takimi jak IMAGE, AUDIO i TEXT, o jeszcze większych wymaganiach pamięciowych. Obraz w postaci rastrowej może zajmować nawet 4 mln bajtów.

Obiekty. Tradycyjne systemy zarządzania bazami danych są nastawione na operowanie strukturami rekordów. W niektórych modelach danych (np. sieciowym) występują wprawdzie struktury łącznikowe wiążące rekordy, nie sposób jednak przedstawić użytkownikowi tak powiązanych rekordów jako pewnej jednostki semantycznej. Nowoczesne zastosowania – takie jak biurowe – wymagają wzmocnienia semantycznych środków wyrazu. Wprowadza się obiekty o bardzo złożonej strukturze. Wiele obiektów, występujących w praktyce biura, jest budowanych z innych obiektów; na przykład, opracowanie składa się z rozdziałów, które z kolei są złożone z akapitów.

Schematy. W tradycyjnym systemie zarządzania bazą danych, schemat logiczny musi być określony w fazie projektowania bazy danych – za pomocą języka opisu danych. Przyjęcie statycznych schematów, typowe dla dzisiejszych systemów zarządzania bazami danych, nie jest możliwe w systemie biurowym, ponieważ biuro jest bardzo podatne na zmiany. Istnieją dwie dynamiczne cechy typów obiektów, występujących w schematach:

- Może zmieniać się opis struktury typu obiektów. Normalnym zdarzeniem w pracy biura jest dołączenie nowego rozdziału, rozbicie rozdziału na dwa nowe rozdziały lub opatrzenie notą otrzymanego listu. Wszystkie te operacje wiążą się ze zmianą opisu pewnych typów obiektów.
- Może nastąpić całkowite usunięcie opisu jakiegoś typu obiektu lub wprowadzenie opisu zupełnie nowego typu (dla BSI takie zmiany są charakterystyczne i bardzo częste). Schematy logiczne, tworzone na użytek BSI, różnią się od schematów przeznaczonych do innych zastosowań również pod względem ilościowym. Zwykle schematy logiczne dla BSI zawierają dużą liczbę opisów typów obiektów, którym odpowiada niewielka liczba wystąpień tych obiektów. W tradycyjnych zastosowaniach sytuacja jest na ogół odwrotna.

Złożoność. Obiekty biurowe są w dużym stopniu wzajemnie powiązane. Dokumenty są gromadzone w teczkach, a teczki w kartotekach; dokumenty zawierają wiele odwołań do innych odnośnych dokumentów itd. Dobrych metod opisu związków, zachodzących między obiektami, dostarczają mechanizmy tworzenia abstrakcji. Wyróżnia się trzy takie mechanizmy.

- Uogólnienie jest stosowane do zdefiniowania typu obiektu **N** jako podtypu pewnego innego typu obiektu **M**, w ten sposób, że **N** może dziedziczyć właściwości od **M**. Na przykład, może istnieć ogólny nagłówek dokumentu, którego właściwości (takie, jak Autor, Znak Firmowy, Data) są dziedziczone przez specjalne nagłówki, zależne od zastosowania.
- Mechanizm agregacji służy do łączenia rozmaitych typów obiektów w jeden typ. Za pomocą tej metody abstrakcji głos, obraz i tekst mogą być połączone w dokument wielonośnikowy.
- Asocjacja odzwierciedla związki między obiektami nie należącymi do żadnej z powyższych hierarchii abstrakcyjnych. Mechanizm ten jest istotny, na przykład w modelowaniu odnośników prowadzących od jednego dokumentu do innych.

Szablony. Szablony są konstrukcjami służącymi do reprezentowania obiektów. Ideę szablonu można porównać z zewnętrznym poziomem architektury systemu zarządzania bazą danych przyjętej przez ANSI/SPARC. Obiekty biurowe mają tendencję do grupowania wielkiej liczby danych. Niektóre z obiektów danych przechowują informacje wrażliwe. Szablony spełniają następujące zadania:

- Udostępniają obiekty lub fragmenty obiektów jedynie osobom upoważnionym. Realizują koncepcję biura opartą na rolach. Tradycyjne systemy zarządzania bazami danych opierają się na autoryzacji imiennej. System biurowy powinien opierać się na autoryzacji funkcyjnej, tj. takiej, w której upoważnienie dla pracownika biura zależy od roli, jaką odgrywa w biurze.
- Przedstawiają obiekty użytkownikowi bądź urządzeniom wyjściowym, w skoordynowany sposób, stosownie do zdolności percepcyjnej użytkownika lub parametrów technicznych urządzenia.
- Przekształcają informację wyjściową na pożądaną postać (tabela, wykres itp.).

Dla jednego obiektu przechowywanego w bazie danych może istnieć kilkanaście szablonów.

Operacje

Tradycyjny system zarządzania bazą danych wprowadza rozróżnienie między językiem definiowania danych, stosowanym na etapie projektowania bazy danych do konstruowania schematów (typów danych), a językiem manipulowania danymi, przeznaczonym do operowania obiektami.

Ponieważ w biurowym systemie zarządzania bazą danych schemat ma charakter dynamiczny, ścisłego rozgraniczenia między językiem definiowania danych a językiem manipulowania danymi nie da się utrzymać. Język definiowania danych będzie używany także po ukończeniu fazy projektowania bazy danych. Ograniczenie możliwości jego użycia do kręgu upoważnionych użytkowników musi zostać zniesione.

Muszą istnieć operatory konstruowania złożonych obiektów i selekcji obiektów składowych. Wymagane są również operatory rozpoznawania struktury.

W skład biurowego systemu zarządzania bazą danych powinny wejść mocne środki do formułowania zapytań, podobne do tych, jakie są właściwe relacyjnym systemom zarządzania bazami danych. Niezbędne jest jednak pewne rozszerzenie. Stwierdzono, że użytkownicy w biurze rzadko są w stanie sformułować swoje zapytanie ad hoc w taki sposób, by wyszukana informacja odpowiadała ich życzeniom. Z tego powodu ważne jest zaimplementowanie mechanizmu „wyboru i przeglądania" (ang. selecting and browsing). W trakcie selekcji obiektów informacyjnych użytkownik może je przeglądać i decydować, co z nimi zrobić. Jeśli wybrany obiekt nie odpowiada życzeniom użytkownika, to może on przerwać proces selekcji i przeformułować kryteria wyboru.

System zarządzania bazą danych w biurze powinien ułatwiać przeszukiwanie tekstów niesformatowanych. Operatory do realizacji takich zadań różnią się od operatorów (warunków) przeszukiwania z wykorzystaniem cech. Znanych jest kilka podejść do przeszukiwania tekstów niesformatowanych, na przykład specyfikacja napisów związanych przez operatory albo metody statystyczne.

Wyzwalacze

Jednym z ważnych zadań systemu zarządzania bazą danych jest zagwarantowanie poprawności przechowywanych obiektów i relacji między nimi. Poprawność rozpatruje się od dwóch stron – od strony warunków integralności i od strony warunków spójności. Warunki integralności określają charakterystyki każdej jednostki danych, niezależnie od jakichkolwiek innych jednostek danych w bazie danych. Warunki spójności natomiast określają relacje zachodzące między dwiema lub więcej jednostkami danych w bazie danych.

System zarządzania bazą danych w biurze narzuca większe wymagania na poprawność niż system tradycyjny. Musi być znaleziony mechanizm rozwiązania następujących dwóch problemów. Po pierwsze, w biurze występuje wielka liczba warunków poprawności. W niektórych wypadkach wiążą one wiele obiektów. Po drugie, dynamiczny charakter biura wymusza dynamiczne zachowywanie warunków poprawności. Łatwe powinno więc być dodanie, zmiana lub odrzucenie warunku.

Odpowiednią metodą rozwiązania powyższych problemów wydaje się być mechanizm wyzwalaczy (ang. triggers). Wyzwalacz jest parą:

⟨**Z: zdarzenie, A: akcja**⟩,

gdzie: **Z** jest wskaźnikiem sygnalizującym zajście specyficznej sytuacji, a **A** jest zbiorem operacji, jakie powinny być wykonane na bazie danych.

Zdarzenia mogą zostać uwolnione, na przykład wskutek wykonania operacji na bazie danych albo wskutek przeterminowania. Część wyzwalacza, opisująca akcję, składa się z następujących podczęści:
– część warunku; warunki mogą być formułowane jako warunki początkowe, testowane przed wykonaniem operacji na bazie danych, albo warunki końcowe, testujące poprawność wykonanej operacji;
– część operacyjna; składa się z dwóch niezależnych operacji, z których jedna jest wykonywana wówczas, gdy warunek jest prawdziwy, druga – gdy warunek jest fałszywy.

W [2] proponuje się prosty, lecz skuteczny mechanizm wyzwalaczy. Związany z tym mechanizmem sprzęg użytkownika zawiera cztery komponenty:
- definicji, obliczania i usunięcia warunku,
- definicji, uwolnienia i usunięcia zdarzenia,
- definicji i usunięcia akcji,
- definicji, aktywacji, deaktywacji i usunięcia wyzwalacza.

Transakcje

Pojęcie transakcji pozostaje w ścisłym związku z pojęciem poprawności. Transakcję stanowi zbiór operacji na bazie danych, przeprowadzających bazę z jednego stanu poprawnego do innego poprawnego stanu. W tradycyjnych systemach zarządzania bazami danych zazwyczaj są wykonywane transakcje krótkie, dzięki czemu można je wykonywać seryjnie.

Transakcje biurowe zachowują się podobnie jak transakcje w procesie inżynierskim. Przez dłuższy czas mogą występować transakcje nie zakończone. Aby podtrzymać równoległy charakter długich transakcji biurowych, system zarządzania bazą danych powinien dostarczać zupełnie nowego mechanizmu transakcji. System zarządzania bazą danych w biurze ma status usługowy. Ze względu na rozproszone środowisko BSI może zdarzyć się tak, że transakcję trzeba będzie wykonać w obrębie jednego stanowiska roboczego (por. cz. 1 tego artykułu). W tej sytuacji zadanie systemu zarządzania bazą danych ogranicza się jedynie do zbadania poprawności obiektów zwróconych ze stanowiska roboczego.

Czas

Dla biurowych systemów zarządzania bazami danych istotne jest modelowanie uwzględniające zmiany w czasie. Pojęcia: wersji, wariantu i historii dotyczą wszystkich procesów w pracy biura. Rozróżnia się dwa aspekty modelowania uwzględniającego upływ czasu:
- Ujęcie statyczne zmierza do uchwycenia bieżącego stanu bazy danych. Aby uwzględnić aspekty zależne czasowo albo dołącza się atrybut opisujący czas do tych atrybutów, dla których ważne jest prowadzenie zapisu w czasie, albo zdejmuje się pewną liczbę obrazów stanu danych (ang. snapshots, migawki) i wykonuje się badania dotyczące czasu na odpowiednim podzbiorze tych obrazów.
- Ujęcie dynamiczne koncentruje się na przejściach między stanami. Istotny jest czas przejścia, dla którego określa się warunki początkowe i końcowe (np. przez wyzwalacze). Tradycyjne systemy zarządzania bazami danych są zazwyczaj bierne, w tym sensie, że tylko reagują na żądania dotyczące wyszukiwania informacji i manipulowania danymi. W ujęciu dynamicznym biurowy system zarządzania bazą danych zmienia się z biernego na aktywny.

ARCHITEKTURA SYSTEMU ZARZĄDZANIA BAZĄ DANYCH DLA BIURA

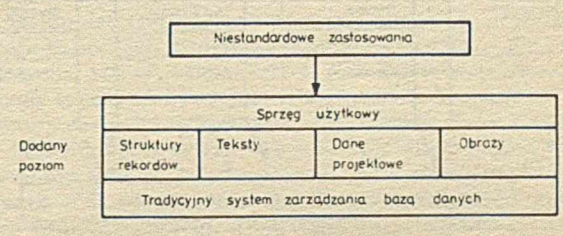
Powyżej dokonano przeglądu najważniejszych wymagań, stawianych przed bazą danych, wynikających z zastosowania w biurowych systemach informacyjnych. Analiza tych wymagań, w połączeniu z analizą tradycyjnych systemów zarządzania bazami danych wskazuje, że dzisiejsze systemy zarządzania nie wspierają dostatecznie ich zastosowań w biurowym systemie informacyjnym. Obecnie zostaną przedstawione rozwiązania umożliwiające przezwyciężenie niedostatków tych systemów.

Podejście konstrukcyjne

Tradycyjny system zarządzania bazą danych nie odpowiada wymaganiom stawianym przez biuro. W [3] proponuje się cztery warianty konstrukcyjne dla tak zwanych niestandardowych systemów zarządzania bazami danych, do których zaliczają się również systemy biurowe.

1. Architektura dodanego poziomu (rys. 1)

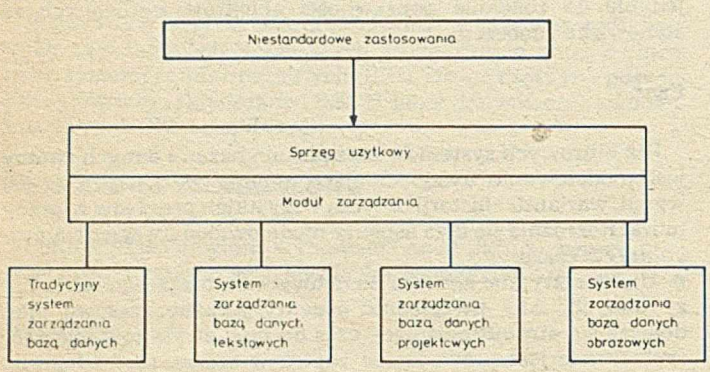
W podejściu tym do tradycyjnego systemu zarządzania bazą danych dodaje się nowy, silny poziom, na którym są realizowane wszystkie wymagania niestandardowych zastosowań. Zaletą tego podejścia jest redukcja kosztów implementacji, a wadą



Rys. 1. Architektura dodanego poziomu

– dramatyczne obniżenie wydajności, spowodowane niedostatecznym wspieraniem operacji na tym poziomie przez niższe poziomy.

2. Architektura kombinowana (rys. 2)
Architektura kombinowana łączy tradycyjny system zarządzania bazą danych z niezależnymi systemami specjalnymi przez moduł zarządzania i wspólny moduł komunikacji z użytkownikiem. Zaletą jest możliwość włączenia do systemu już istniejących podsystemów; wady – to możliwa wysoka redundancja danych fizycznych w podsystemach i złożony moduł zarządzania, organizujący transformację danych oraz kontrolę poprawności.

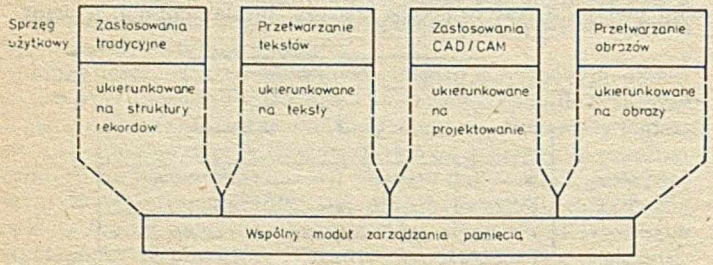


Rys. 2. Architektura kombinowana

3. Architektura rozszerzona (rys. 3)
Architektura rozszerzona ma umożliwiać konsekwentną, poziomą rozbudowę tradycyjnego systemu zarządzania bazą danych, przez dodawanie nowych rodzajów reprezentacji i rozszerzenie składowych systemu przeznaczonych do obsługi specjalnych struktur i operacji. Jest to podejście „otwarte", gdyż nie jest możliwe wbudowanie w składową wszystkich wyobrażalnych wymagań. Wymagania stawiane przez nowe zastosowania powodują dokonywanie kolejnych rozszerzeń.

Rys. 3. Architektura rozszerzona

4. Architektura rdzeniowa (rys. 4)
W podejściu tym proponuje się podział systemu zarządzania bazą danych na dwie składowe: niezależny od zastosowań rdzeń i jedną lub więcej warstw zależnych od zastosowań. Wszystkie warstwy zależne od zastosowań operują tym samym rdzeniem, czyli tym samym systemem pamięci i tymi samymi obiektami. Zaletą jest względna stabilność rdzenia i zwarte, nie obciążone zbędnymi narzutami warstwy zależne od zastosowań. Zminimalizowana jest redundancja danych. Wadą tego podejścia jest konieczność dokonania nowej implementacji całego systemu i transformacji obiektów z już istniejących systemów.



Rys. 4. Architektura rdzeniowa

Składowe rdzenia

Poziom rdzenia (rdzeń) niestandardowego systemu zarządzania bazą danych (także systemu dla biur) musi wspierać żądania z wyższych poziomów, ukierunkowanych na zastosowania. Podział systemu pamięci na warstwy funkcjonalne wydaje się być dobrą metodą konstrukcji elastycznego modułu obsługi pamięci. Poniżej przedstawiono krótki opis niektórych warstw funkcjonalnych systemu pamięci.

● **Warstwa fizycznych operacji wejścia-wyjścia**
Warstwa ta zapewnia realizację operacji przesłań fizycznych między pamięcią operacyjną a pomocniczą (dyskiem). Wprowadza ona blokową organizację obszarów dyskowych oraz zarządza listą wolnych bloków. Na użytek warstw położonych bezpośrednio wyżej są zaimplementowane takie funkcje, jak odczyt i zapis bloku. Zazwyczaj warstwa ta jest częścią systemu operacyjnego, zarządzającego pracą komputera głównego.

● **Warstwa zarządzania buforami**
Warstwa ta zarządza pamięcią operacyjną. Aby zminimalizować liczbę operacji dostępu do dysku, w skład każdego z systemów zarządzania bazą danych wchodzi jego własny zarządca buforów dla bloków sprowadzanych z dysku. Często strategia wymiany bloków polega na przetrzymywaniu większości użytych bloków w pamięci operacyjnej. Użyteczne wydaje się wprowadzenie mechanizmu umożliwiającego jawne wskazanie bloków, które mają być obecne w pamięci operacyjnej. Na użytek warstw następnych muszą być więc określone takie operacje, jak „odczytaj_blok", „zatrzymaj_blok" i „zwolnij_blok". Strategia wymiany bloków automatycznie wykonuje operację „zapisz_blok".

● **Warstwa struktur pamięci**
Warstwa ta jest odpowiedzialna za zarządzanie obiektami ustrukturowanymi. Muszą w niej być zaimplementowane struktury rekordów z tradycyjnych zastosowań. Tu także powinny zostać zaimplementowane długie pola danych na użytek danych niesformatowanych (takich jak biurowe obiekty informacyjne TEXT, IMAGE i AUDIO), a także drzewa (np. dla B_* – wskaźników) i struktury sieciowe. Wszystkie struktury złożone są konstruowane z rekordów i długich pól danych. Poniżej tej warstwy jednostką dostępu jest blok, a powyżej – rekord lub część długiego pola danych.

● **Warstwa dostępu**
W warstwie tej operacje dostępu do struktur pamięci są implementowane jako operacje różnego typu przeglądania. Forma przeglądania zależy od struktury pamięciowej, jakiej dotyczy. Metoda przeglądania pozycji długiego pola danych dostarcza dogodnych środków odczytania całej pozycji (taką formę przeglądania czasami nazywa się portalem). Przeglądanie umożliwia formułowanie warunków przeszukiwania jako wyrażeń boolowskich (w normalnej postaci koniunkcyjnej).

W obecnym stanie badań trudno orzec, gdzie kończy się rdzeń, a zaczyna poziom zależny od zastosowań. Muszą wystąpić warstwy modelowania upływu czasu, formułowania zapytań, zarządzania wyzwalaczami, kontroli transakcji itd. Pierwsze publikacje i implementacje wskazują, że powyższe cztery warstwy są pomocne w implementacji systemu zarządzania bazą danych przeznaczonego dla różnych zastosowań [1, 3].

* * *

Artykuł ten stanowi przyczynek do analizy wymagań stawianych przez biura. Opisano w nim podejścia występujące w modelowaniu biur; wprowadzono klasyfikację modeli w celu wszechstronnego scharakteryzowania wymagań stawianych systemowi zarządzania bazą danych dla biurowego systemu informacyjnego; podano przegląd możliwych rozwiązań w architekturze baz danych, wraz z krótką analizą ich zalet i wad. Autor opowiada się za architekturą rdzeniową i implementacją warstwową.

dokończenie na s. 11

Tłumaczył i opracował:
A. RADOMSKI

RYSZARD RYBUS
Instytut Informatyki
Politechnika Warszawska

Nowoczesne programowanie w języku C

Stałe poszerzanie obszarów zastosowań informatyki powoduje przede wszystkim zwiększone zapotrzebowanie na oprogramowanie. Programiści za pomocą dostępnych im środków nie są w stanie zaspokoić tego zapotrzebowania. Przewaga popytu nad podażą oraz stałe zmniejszanie się kosztów sprzętu powodują znaczny wzrost kosztów oprogramowania (do około 80% wartości) systemów informacyjnych [1]. Niezadowalająca jest również niezawodność dostarczanego oprogramowania, co ma podstawowe znaczenie w zastosowaniach wojskowych, badaniach kosmosu, medycynie oraz obliczeniach inżynierskich. Taka sytuacja nosi często nazwę „kryzysu oprogramowania".

Od 1968 r. dziedzina wiedzy zajmująca się problemami związanymi z produkcją oprogramowania zyskała miano inżynierii oprogramowania. Zajmuje się ona poszukiwaniem praktycznych zasad, metod i narzędzi, umożliwiających tworzenie niezawodnego oprogramowania przy minimalizacji jego kosztów. Jednym z kierunków badań inżynierii oprogramowania są metodyki projektowania programów. Ich zastosowanie jest efektywne, jeśli istnieją odpowiednie narzędzia wspomagające. Ideałem byłoby, gdyby projekt programu był zapisany w notacji nadającej się do dalszego maszynowego przetwarzania. Nadaje to kierunek rozwoju językom programowania, które w zamierzeniu wspierają rozpowszechnione metodyki projektowania lub ich klasę [4]. Obecnie coraz bardziej rozpowszechnione są języki wspomagające metodyki, oparte na pojęciu abstrakcyjnych typów danych [1, 6]. Najbardziej znanymi przedstawicielami tej klasy języków są: Ada, Modula 2 i Chill.

W dalszej części artykułu przedstawiono zasady inżynierii oprogramowania, stanowiące podstawy nowoczesnych konstrukcji we wspomnianych językach, a także możliwość efektywnego stosowania tych zasad przy programowaniu w języku C.

ZASADY INŻYNIERII OPROGRAMOWANIA

Celem inżynierii oprogramowania jest określenie warunków do podnoszenia jakości oprogramowania i obniżania kosztów jego wytwarzania [7]. Obszar zainteresowania tej dyscypliny obejmuje więc badania nad procesem powstawania programu oraz problemy związane z zarządzaniem jego produkcją. Na podstawie dotychczasowych osiągnięć można sformułować zbiór prostych zasad pozwalających osiągnąć wspomniane wyżej cele inżynierii oprogramowania. W artykule omówiono zasady, które dotyczą głównie fragmentu procesu tworzenia programu obejmującego projektowanie i kodowanie.

Abstrakcja

Główną przyczyną kryzysu w rozwoju oprogramowania jest wzrost złożoności programów. Podstawowym narzędziem intelektualnym, umożliwiającym zapanowanie nad złożonością rozwiązywanego problemu, jest postrzeganie abstrakcyjne. Abstrakcja jest to uproszczony opis systemu, który zwraca uwagę na pewne szczegóły lub właściwości, pomijając inne, nieistotne w dawnej chwili [6]. Abstrakcja nie jest nową koncepcją; jest ona powszechnie stosowana w naszym codziennym patrzeniu na świat.

W odniesieniu do oprogramowania postrzeganie abstrakcyjne pozwala na zdefiniowanie hierarchii modeli (abstrakcji), w której modele niższego poziomu dostarczają szczegółów objaśniających modele z wyższego poziomu. Zasadę abstrakcji stosuje się zarówno do danych, jak i do algorytmów (operacji) składających się na rozwiązanie problemu. Jako przykład rozważę problem obsługi pamięci dyskowej w systemie zarządzania bazą danych [1]. Pamięć dyskowa jest tu widzana jako zbiór plików. Pomija się jednocześnie fakt, że pliki są rozmieszczone na nośniku podzielonym na ścieżki i sektory, oraz że nośnik ten wymaga dodatkowo odpowiedniego sterowania sygnałami elektronicznymi. Są to abstrakcje niższych poziomów, bez których oczywiście nie byłoby możliwe tworzenie plików, ale które w rozpatrywanym problemie nie są istotne (ich obsługą najczęściej zajmują się odpowiednie fragmenty systemu operacyjnego). Z plikami są związane operacje **OPEN, CLOSE, READ, WRITE.** Stosując te operacje pomija się również takie szczegóły, jak wybór ścieżki, znalezienie sektora lub sprawdzenie sumy kontrolnej.

Podsumowując, stosowanie abstrakcji pozwala podzielić problem na mniejsze fragmenty, które staną się następnie częściami składowymi tworzonego oprogramowania. Dzięki temu rozwiązywaniem podproblemów o mniejszej złożoności mogą zająć się różni programiści lub jeden twórca może traktować je w dużym stopniu w sposób rozłączny.

Przesłanianie informacji

Przesłanianie informacji polega na udostępnianiu przez składnik programu tylko tych informacji, które są niezbędne do jego prawidłowego użycia. Pozostałe informacje dotyczące szczegółów realizacji założonych funkcji powinny być na zewnątrz niewidoczne. Przesłanianie informacji przez redukcję wzajemnych zależności między składnikami oprogramowania ma korzystny wpływ na niezawodność i łatwość modyfikowania całego systemu. Wzrost niezawodności wynika z ograniczenia możliwości nieprzewidzianej interakcji między składnikami systemu, gdyż połączenia między nimi są ściśle zdefiniowane. Ukrycie szczegółów implementacyjnych umożliwia zmianę reprezentacji danych i algorytmów bez wpływu na pozostałą część systemu pod warunkiem, że nie ulega modyfikacji sposób dostępu do zasobów zmienianego składnika. Wracając do przykładu z pamięcią dyskową, pożądane jest ukrycie fizycznej organizacji plików na dysku przed modułem realizującym operacje logiczne na plikach. Dzięki temu zamiana jednostki dyskowej, powodująca reorganizację dysku, nie ma wpływu na moduł operacji logicznych znajdujących się na wyższym poziomie abstrakcji.

TYPY ABSTRAKCYJNE W ADZIE

Opisane zasady inżynierii oprogramowania stały się podstawą różnych metodyk projektowania, które powstawały od początku lat siedemdziesiątych. Szczególnego znaczenia nabrała metodyka oparta na abstrakcyjnych typach danych [6] i bezpośrednio po niej następujące projektowanie obiektowe [2]. Jako podstawowe kryterium podziału problemu stosuje się tu dane, a nie przepływ sterowania, jak w rozwiązaniach wcześniejszych. Ideą abstrakcyjnych typów danych jest umieszczenie w jednym module struktury danych i związanych z nią operacji. Pozwala to traktować taki moduł jako typ rozszerzający zbiór typów w języku programowania. Z zasady przesłaniania informacji wyniknęła potrzeba oddzielenia części definiującej właściwości modułu od części implementującej. Wagę wspomnianych metodyk podkreśla fakt, iż znalazły one bezpośrednie wsparcie w nowoczesnych

językach programowania, powstałych na przełomie lat siedemdziesiątych i osiemdziesiątych. Konstrukcje językowe realizujące omawiane idee zostaną krótko omówione na przykładzie języka Ada.

```
package STACK is
   type STACK_TYPE (LENGTH : NATURAL) is limited private;
   procedure CLEAR (ST : in out STACK_TYPE);
   procedure PUSH (VALUE : in INTEGER; ON : in out STACK_TYPE);
   procedure POP (VALUE : out INTEGER; FROM : in out STACK_TYPE);
   function EMPTY (ST : in STACK_TYPE) is BOOLEAN;
   function FULL (ST : in STACK_TYPE) is BOOLEAN;
private
   type STACK_CONTENTS is array (INTEGER range <>) of INTEGER;
   type STACK_TYPE (LENGTH : NATURAL) is
      record
         ELEMENTS : STACK_CONTENTS(1 .. LENGTH);
      TOP : INTEGER range 0..LENGTH;
      end;
end STACK;
```

Wydruk 1. Specyfikacja pakietu w Adzie

Głównymi konstrukcjami służącymi do implementowania abstrakcyjnych typów danych w Adzie są pakiety i typy prywatne. Pakiet składa się z dwóch części: specyfikacji i ciała. Specyfikacja pakietu formalnie definiuje abstrakcyjny typ danych oraz jego połączenie z obiektami dla niego zewnętrznymi. Pakiet udostępnia zwykle typy danych oraz procedury do operowania tymi danymi. Ciało stanowi implementację założonych funkcji pakietu, której szczegóły są ukryte przed innymi składnikami systemu. Zawiera więc procedury: dostępne z zewnątrz, zmienne i lokalne. Reprezentacja wewnętrzna abstrakcyjnego typu danych może być ukryta przez uczynienie jej prywatną. Przykładem, którym będę się posługiwać w dalszej części artykułu, jest stos. Jedną z wielu możliwych realizacji stosu liczb całkowitych zapisaną w Adzie przedstawiono na wydruku 1. Zaprezentowana specyfikacja pakietu **STACK** pozwala na używanie w programie dowolnej liczby stosów. Definicja stosu jest następująca:

STACK_1: STACK.STACK_TYPE;

```
generic
   type ELT_TYPE is private;
   LENGTH : NATURAL;
package STACK is
   type STACK_TYPE is limited private;
   procedure CLEAR (ST : in out STACK_TYPE);
   procedure PUSH (VALUE : in ELT_TYPE; ON : in out STACK_TYPE);
   procedure POP (VALUE : out ELT_TYPE; FROM : in out STACK_TYPE);
   function EMPTY (ST : in STACK_TYPE) is BOOLEAN;
   function FULL (ST : in STACK_TYPE) is BOOLEAN;
private
   type STACK_CONTENTS is array (1 .. LENGTH) of ELT_TYPE;
   type STACK_TYPE (LENGTH : NATURAL) is
      record
         ELEMENTS : STACK_CONTENTS(1 .. LENGTH);
      TOP : INTEGER range 0..LENGTH;
      end;
end STACK;
```

Wydruk 2. Specyfikacja jednostki rodzajowej w Adzie

Ada zawiera konstrukcje umożliwiające tworzenie jednostek rodzajowych, które pozwalają na wyrażanie bardziej ogólnych abstrakcji niż abstrakcyjne typy danych. Jednostki rodzajowe jako parametry akceptują inne abstrakcje, na przykład, typy danych lub podprogramu. Rozważmy problem definiowania typów danych w zastosowaniu używającym różnych rodzajów stosów: stos liczb całkowitych, liczb rzeczywistych i stos elementów typu zdefiniowanego przez użytkownika. We wcześniejszych językach programowania konieczna byłaby oddzielna implementacja każdego z tych trzech typów stosów. Fragmenty programu dla każdego typu byłyby bardzo podobne. Inną wadą tradycyjnej implementacji, oprócz powielania kodu, są kłopoty z pielęgnacją różnych wersji stosu. Jednostki rodzajowe stanowią rozwiązanie przedstawionego problemu. Są one wzorcami, na podstawie których powstają egzemplarze odpowiadające wartościom parametrów aktualnych jednostki. Specyfikację pakietu rodzajowego obsługującego stos przedstawiono na wydruku 2. Na tej podstawie można zdefiniować w programie pakiety implementujące stosy o różnych typach elementów, przykładowo:

package INT_STACK **is new** STACK(INTEGER,20);
package FLOAT_STACK **is new** STACK(FLOAT,30);

NOWOCZESNE PROGRAMOWANIE W JĘZYKU C

Język C w zamierzeniu autorów miał być językiem do realizacji oprogramowania systemowego. Przemawiały za tym takie jego cechy, jak: łatwy dostęp do specyfikacji cech sprzętu, efektywność kodu wynikowego i elastyczność. Konsekwencją tej ostatniej cechy jest odstąpienie od obowiązkowej kontroli zgodności typów, co może powodować duże trudności z opanowaniem języka przez niezbyt wprawnych programistów. Mimo to popularność języka C stale rośnie.

Standard języka C nie zawiera konstrukcji odpowiadających omówionym elementom Ady. W ostatnich latach powstają mutacje idące w tym właśnie kierunku, takie jak C++ [8] lub Objective – C [3]. Poniżej przedstawiono reguły umożliwiające wprowadzenie idei inżynierii oprogramowania do praktyki programisty w standardowym języku C. Powodzenie stosowania tych reguł zależy wyłącznie od zdyscyplinowania programisty. Wszystkie przykłady w języku C prezentowane w niniejszym artykule zostały uruchomione przy użyciu kompilatora Turbo C.

Abstrakcyjne typy danych

Język C dopuszcza, aby program składał się z fragmentów rozmieszczonych w różnych plikach. Pliki zawierające części programu w języku C mogą być kompilowane niezależnie. Umożliwia to podzielenie dużego programu na mniejsze fragmenty. Pliki mogą być wykorzystane do zrealizowania idei abstrakcyjnych typów danych, gdyż odpowiednie konstrukcje językowe pozwalają na ukrywanie lub udostępnianie obiektów w nich zawartych. Z każdym obiektem w języku C jest związana klasa pamięci, która determinuje jego okres istnienia (ang. *life time*) i zakres widoczności [5]. Dla nas interesujący jest problem widoczności.

Widoczność zmiennych zdefiniowanych[1)] wewnątrz bloku, czyli również wewnątrz funkcji, jest ograniczona do tego właśnie bloku. Inaczej jest ze zmiennymi zdefiniowanymi na zewnątrz funkcji. Domyślną klasą takich zmiennych jest klasa zewnętrzna. Zmienne zewnętrzne mogą być używane w funkcjach znajdujących się w innych plikach, przy czym przed odwołaniem do zmiennej zewnętrznej zdefiniowanej w innym pliku, musi być ona zadeklarowana przy użyciu słowa kluczowego **extern**. Poprzedzenie definicji zmiennej słowem kluczowym **static** powoduje zmianę klasy pamięci na statyczną. Widoczność zmiennych statecznych jest ograniczona do pliku, w którym są one zdefiniowane. Reguły widoczności funkcji są podobne jak dla zmiennych leżących na zewnątrz funkcji. Słowo kluczowe **static** uniemożliwia jej użycie w innym pliku. Użycie funkcji zdefiniowanej w innym module nie zawsze wymaga jawnej deklaracji. Deklaracja jest konieczna, jeśli typ funkcji jest różny od domyślnego typu **int**. Nowsze wersje kompilatorów dopuszczają również pełną deklarację funkcji, która oprócz nazwy i typu funkcji zawiera typy parametrów formalnych. Pozwala to kompilatorowi sprawdzić poprawność wywołania funkcji, dzięki czemu można uniknąć wielu przykrych błędów.

W języku C nie ma możliwości ukrycia szczegółów reprezentacji typu deklarowanego przez programistę. Nowe nazwy typów (ale nie nowe typy danych) można tworzyć przy użyciu deklaracji **typedef**, której stosowanie poprawia czytelność całego programu. Przy przekazywaniu typów między modułami mechanizm ten pozwala uniknąć uciążliwego przepisywania skomplikowanych często deklaracji typów. W ten sposób nie zachęca do odwoływania się do szczegółów reprezentacji typu danych.

Wspomniano wcześniej o potrzebie rozdzielenia specyfikacji właściwości modułu do ich implementacji. W języku C specyfikacja i implementacja modułu powinny znajdować się w oddzielnych plikach. Informacje zawarte w pliku specyfikacji muszą być udostępniane innym modułom na etapie kompilacji. W C można to osiągnąć za pomocą dyrektywy preprocesora **include**, służącej do dołączania pliku tekstowego do pliku zawierającego

[1)] Definicja obiektu jest rozumiana jako specyfikacja jego właściwości i alokacja pamięci. Deklaracja natomiast służy tylko do określenia właściwości obiektu.

dyrektywę. Plik specyfikacji zawiera deklaracje obiektów udostępnianych na zewnątrz modułu, a mianowicie:

- definicje stałych,
- deklaracje obiektów zewnętrznych, niezbędne do dalszych deklaracji,
- deklaracje typów,
- deklaracje zmiennych zewnętrznych,
- pełne deklaracje funkcji widocznych dla innych modułów.

Kolejność wymienionych deklaracji powinna być stała, wpływa to bowiem na poprawę czytelności programu. Plik specyfikacji musi być dołączony do plików implementujących inne moduły, umożliwiając w ten sposób korzystanie z zasobów w nim zadeklarowanych. W pliku implementującym oprócz definicji obiektów udostępnianych na zewnątrz znajdują się definicje obiektów lokalnych. Wszystkie obiekty lokalne powinny mieć klasę statyczną. Polecana struktura pliku implementacyjnego jest następująca:

- definicje stałych,
- deklaracje obiektów zewnętrznych (pliki specyfikacji innych modułów),
- deklaracje używanych funkcji bibliotecznych,
- deklaracje typów,
- definicje zmiennych globalnych, zewnętrznych i statycznych.
- definicje funkcji zewnętrznych i statycznych.

Rzeczywisty program, zrealizowany zgodnie z metodyką opartą na abstrakcyjnych typach danych, może składać się z dużej

```
/* stack.h - plik specyfikacji modulu stos liczb calkowitych */

/*= DEFINICJE STALYCH =*/

#define MAX_SIZE    20   /* maksymalna dlugosc stosu */

/*= DEKLARACJE TYPOW  =*/

typedef int STACK_CONTENTS [MAX_SIZE];
typedef struct {
     STACK_CONTENTS elements; /* zawartosc stosu */
     int top;                 /* wskaznik szczytu stosu */
     } STACK_TYPE;
typedef STACK_TYPE * STACK_PTR;

/*= DEKLARACJE FUNKCJI =*/

/* Clear - zerowanie stosu */
void Clear (STACK_PTR st);
     /* Parametry:
        st - wskaznik na stos */

/* Push - wstawianie elementu na stos */
void Push (int value, STACK_PTR on);
     /* Parametry:
        value - wartosc skladowanego elementu
        on - wskaznik na stos */

/* Pop - pobranie elementu ze stosu */
void Pop (int * value, STACK_PTR from);
     /* Parametry:
        value - wskaznik na zwracana wartosc elementu
        on - wskaznik na stos */
```

```
/* stack.c - plik implementacji modulu stos  */


/*= DEFINICJE STALYCH =*/

#define MAX_SIZE    20   /* maksymalna dlugosc stosu */

/*= DEKLARACJE TYPOW  =*/

typedef int STACK_CONTENTS [MAX_SIZE];
typedef struct {
     STACK_CONTENTS elements; /* zawartosc stosu */
     int top;                 /* wskaznik szczytu stosu */
     } STACK_TYPE;
typedef STACK_TYPE * STACK_PTR;

/*= DEFINICJE FUNKCJI =*/

void Clear (STACK_PTR st)
{
    st->top = 0;
}

void Push (int value,STACK_PTR on)

{
    on->elements[on->top++] = value;
}

void Pop (int * value,STACK_PTR from)
{
    *value = from->elements[--from->top];
}
```

```
/* use_stack.c - program wykorzystujacy modul stosu */

/*= DEKLARACJE OBIEKTOW ZEWNETRZNYCH =*/

#include "stack.h"

/*= DEFINICJE ZMIENNYCH =*/

static STACK_TYPE stack_1,stack_2;

/*= DEFINICJE FUNKCJI =*/

main()
{
    int elt;

     Clear(&stack_1);
     Clear(&stack_2);
     Push(1,&stack_1);
     Pop(&elt,&stack_1);
     printf("\nsi %d",elt);
     exit(0);
}
```

Wydruk 3. Moduł stosu w języku C

```
/* gen_stack.h - plik specyfikacji jednostki rodzajowej dla stosu */

/*= DEKLARACJE TYPOW  =*/

typedef void * STACK_TYPE;

/*= DEKLARACJE FUNKCJI =*/

/* CreateStack - utworzenie nowego stosu */
void CreateStack (int size,int len,STACK_TYPE * new_st);
     /* Parametry :
          size - wielkosc elementu stosu,
          len  - dlugosc stosu
          new_st - wskaznik na nowo utworzony stos */

/* Wstawienie elementu na stos */
void Push (void * value , STACK_TYPE st);
     /* Parametry :
          value - wskaznik na element wstawiany na stos
          st - wskaznik na stos */

/* Pobranie elementu ze stosu */
void Pop (void * value, STACK_TYPE st);
     /* Parametry:
          value - wskaznik na element wstawiany na stos
          st - wskaznik na stos */
```

```
/* gen_stack.c - plik implementujacy jednostke rodzajowa stos */

/*= DEKLARACJE TYPOW  =*/

typedef struct {
     int elt_size;
     int max_len;
     int top;
     } STACK_DESC;

/*= DEFINICJE FUNKCJI =*/

void CreateStack (int size,int len,STACK_DESC ** new_st)
{
    *new_st=(STACK_DESC *)malloc(len*size+sizeof(STACK_DESC));
     (*new_st)->elt_size=size;
     (*new_st)->max_len=len;
     (*new_st)->top = 0;
}

void Push (void * value,STACK_DESC * st)
{
    memmove((char *)(st+1)+st->elt_size*st->top,value,st->elt_size);
     st->top++;
}

void Pop (void * value,STACK_DESC * st)
{
    st->top--;
     memmove(value,(char *)(st+1)+st->elt_size*st->top,st->elt_size);
}
```

```
/* use_gen.c - program wykorzystujacy jednostke rodzajowa stosu */

/*= DEKLARACJE OBIEKTOW ZEWNETRZNYCH =*/

#include "gen_stack.h"

/*= DEFINICJE ZMIENNYCH =*/

static STACK_TYPE i_stack,f_stack;

main()
{
    int i_elt;
     float f_elt;

     CreateStack(sizeof(int),15,&i_stack);
     CreateStack(sizeof(float),25,&f_stack);
     i_elt=1;
     Push(&i_elt,i_stack);
     f_elt=21.1;
     Push(&f_elt,f_stack);
     Pop(&i_elt,i_stack);
     Pop(&f_elt,f_stack);
}
```

Wydruk 4. Moduł stosu jako jednostka rodzajowa

liczby modułów, umieszczonych w różnych plikach, oddzielnie kompilowanych. Zapewnienie, że do programu wynikowego są włączane aktualne wersje poszczególnych modułów, stanowi duży kłopot przy dużej liczbie plików. Do większości kompilatorów C są dołączane programy usługowe, zwyczajowo nazywane **MAKE**, rozwiązujące ten problem. Programy te umożliwiają określenie wzajemnych zależności między plikami, istotnych ze względu na kolejność kompilacji. Na przykład, modyfikacja pliku specyfikacji używanego przez moduł powinna powodować jego ponowną kompilację, a ta z kolei ponowne łączenie wszystkich modułów składowych w program wynikowy. Program **MAKE** zapewnia otrzymanie najbardziej aktualnej wersji programu, gdyż opiera się na porównywaniu czasu utworzenia plików pozostających we wzajemnych zależnościach.

Zastosowanie omówionych elementów języka C przy realizacji modułu stosu zaprezentowano na wydruku 3. W tym i następnym przykładzie pominięto problemy związane z obsługą błędów wynikających z braku pamięci lub nieprawidłowej kolejności wywołań procedur modułu.

Jednostki rodzajowe

Jednostki rodzajowe są konstrukcją, która odbiega bardzo daleko od filozofii języka C. Jednakże elastyczność tego języka umożliwia tworzenie modułów, które mają podstawową zaletę jednostek rodzajowych, jaką jest wielokrotne wykorzystywanie raz napisanego kodu. Na wydruku 4 przedstawiono realizację stosu w języku C, jako jednostki rodzajowej. Zaproponowane rozwiązanie dopuszcza używanie dowolnej liczby stosów różnych typów. Opracowywanie tego rodzaju modułów wymaga od programisty dużego doświadczenia. Wynika to z intensywnego stosowania dość skomplikowanych konstrukcji wskaźnikowych, które są najtrudniejszym do opanowania elementem języka C. Również korzystanie z takich modułów wymaga dużej uwagi i dyscypliny programisty. Język nie zawiera żadnych konstrukcji sprawdzających poprawność wywołania funkcji z parametrami przekazywanymi przez adres. W programie przykładowym jest możliwe następujące wywołanie:

```
Push(&f_elt,int_stack);
```

Tworzenie modułów mających cechy jednostek rodzajowych jest opłacalne jedynie w wypadku typów danych często wykorzystywanych w różnych zastosowaniach, na przykład kolejki, listy lub drzewa.

Obsługa wyjątków

Od nowoczesnych języków wymaga się mechanizmów zapewniających niezawodność oprogramowania. Istotna jest możliwość reagowania na sytuacje niespodziewane, które nie mają związku z rozwiązywanym problemem. Sytuacje takie noszą nazwę wyjątków. Przykładem zdarzenia tego rodzaju jest dzielenie przez zero lub zewnętrzna próba przerwania programu przez operatora. W językach takich jak Ada programista może związać z fragmentem programu segment obsługi wyjątku. Wystąpienie wyjątku w chronionym bloku programu powoduje przekazanie sterowania do segmentu obsługi.

Wyjątki mogą być również zastosowane do obsługi zwykłych sytuacji uznawanych za niepoprawne, które są przez programistę przewidywane. Sytuacje takie występują praktycznie w każdym programie i są najczęściej związane z niepoprawnym formatem danych wejściowych. Często spotykana jest struktura programu, w której rozpoznanie błędu na dowolnym poziomie hierarchii wywołań modułów powoduje powrót do poziomu programu głównego. Sytuacja taka występuje, na przykład, w programach sterowanych dyrektywami, w momencie wykrycia błędnego formatu lub wartości parametrów dyrektywy. W tradycyjnych językach programowania każdy moduł w hierarchii znajdujący błąd udostępnia znacznik błędu, który musi być sprawdzany w kolejnych modułach wywołujących. Jeśli liczba modułów w hierarchii jest duża, to oprogramowanie wszystkich sytuacji błędnych staje się zadaniem dość uciążliwym. W Adzie tego rodzaju problem może być rozwiązany przez wprowadzenie wyjątku zdefiniowanego przez programistę i sygnalizowanie jego wystąpienia odpowiednią instrukcją.

```
/* use_exp.c - program wykorzystujacy modul stosu z obsluga bledow */

/*= DEKLARACJE OBIEKTOW ZEWNETRZNYCH =*/

#include "exp_stack.h"

/*= PROCEDURY BIBLIOTECZNE =*/

#include <stdio.h>
#include <signal.h>
#include <setjmp.h>

/*= DEFINICJE ZMIENNYCH =*/

static STACK_TYPE stack_1,stack_2;
static jmp_buf env; /* bufor na stan programu */

/*= DEFINICJE FUNKCJI =*/


int err_handler(int i)   /* procedura obslugi sygnalu */
{
    ssignal(i,err_handler);  /*  odtworzenie  powiazania  procedury  z
sygnalem */
     printf("\nError %d",i);
     longjmp(env,0);
}


main()
{
    int elt;

     Clear(&stack_1);
     Clear(&stack_2);
     /* zwiazanie procedury obslugi wyjatku z sygnalem */
     ssignal(STACK_ERR,err_handler);
     /* zachowanie adresu powrotu z obslugi bledu */
     setjmp(env);
     while (1) {    /* petla programu glownego */
        Push(1,&stack_1);
        Pop(&elt,&stack_1);
        Pop(&elt,&stack_1); /* bledna operacja na stosie */
     }
}
```

Wydruk 5. Obsługa wyjątków w języku C

Język C nie zawiera konstrukcji umożliwiających obsługę wyjątków. W bibliotekach dostarczanych przez producentów kompilatorów znajduje się natomiast grupa definicji stałych i funkcji służących do obsługi sygnałów (ang. *signal*). Są one opisane w pliku **signal.h.** Funkcja tam zawarta pozwala związać z pewnymi zdarzeniami procedury, które są wywoływane przy odbiorze sygnału zajścia zdarzenia. Sygnały mogą być zdefiniowane przez programistę lub też być związane z przerwaniami zwykle obsługiwanymi przez system operacyjny. Aby rozwiązać problem przejścia do modułu głównego po wykryciu wyjątku, konieczny jest mechanizm przekazywania sterowania między funkcjami, bez pośrednictwa normalnego mechanizmu wykorzystującego stos. Umożliwiają to dwie funkcje biblioteczne **setjmp** i **longjmp.** Pierwsza z nich zachowuje aktualny stan programu w miejscu jej wywołania. Druga natomiast umożliwia przekazanie sterowania do miejsca, w którym była wywołana funkcja **setjmp.** Na wydruku 5 przedstawiono program, w którym obsługa sytuacji błędnych została zrealizowana za pomocą sygnałów programowych i omówionych funkcji bibliotecznych. Program ten korzysta ze zmodyfikowanego modułu stosu liczb całkowitych z wydruku 3. W pliku specyfikacji i pliku implementacji tego modułu zdefiniowano stałą **STACK_ERR,** która określa numer sygnału związanego z błędną operacją na stosie. Na przykład, przepełnienie stosu w funkcji **Push** jest sygnalizowane przez instrukcję w postaci:

```
if (on->to = = MAX_SIZE)
    gsignal (STACK_ERR);
```

Podobnie jest obsługiwana próba pobrania elementu z pustego stosu w funkcji **Pop.**

* * *

W artykule przedstawiono konstrukcje języka C, które przy odpowiedniej dyscyplinie i doświadczeniu programisty pozwalają na stosowanie nowoczesnych zasad programowania. Metodyki oparte na abstrakcyjnych typach danych umożliwiają podział programu na moduły, których połączenie jest ściśle zdefiniowane, a szczegóły implementacyjne są ukryte przed obiektami zewnętrznymi. Zyskuje się przy tym na czytelności programu, jego niezawodności i łatwości pielęgnacji. Jednostki rodzajowe są środkiem pozwalającym na wielokrotne używanie raz napisanego kodu. Sygnały programowe stanowią efektywne rozwiązanie problemu obsługi sytuacji błędnych.

dokończenie na s. 4

ROMAN PODRAZA
Instytut Informatyki
Politechnika Warszawska

Sterowanie przepływowe (1)

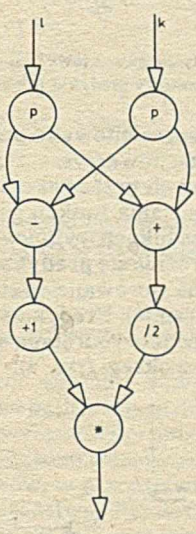
Opis modelu

Od lat siedemdziesiątych obserwuje się dynamiczny rozwój prac nad systemami wieloprocesorowymi i przetwarzaniem równoległym. Wynika to przede wszystkim ze stale zwiększającego się zapotrzebowania na wzrost mocy obliczeniowej i szybkości przetwarzania komputerów. Rozwój nowoczesnych technologii, zwłaszcza VLSI, doprowadził do uzyskania dość tanich układów o niewielkich rozmiarach i potencjalnie olbrzymich możliwościach obliczeniowych. Natomiast czas przetwarzania danego zadania można zmniejszyć przez równoczesne (równoległe, współbieżne) wykonywanie niezależnych części (instrukcji, modułów, procedur) tego zadania.

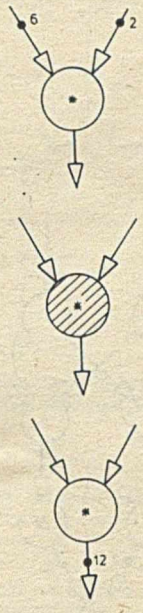
Prace nad przetwarzaniem równoległym i systemami wieloprocesorowymi obejmują różnorodne zagadnienia sprzętowe, programowe i systemowe. Wśród wielu propozycji systemów wieloprocesorowych prostotą i elegancją wyróżniają się **systemy sterowania przepływowego** [13] (ang. *dataflow systems*). Są one oparte na modelu sterowania przepływowego [5, 10], umożliwiającym przetwarzanie równoległe na poziomie operacji.

ZASADY FUNKCJONOWANIA SYSTEMU STEROWANIA PRZEPŁYWOWEGO

W modelu sterowania przepływowego program jest przedstawiany w postaci graficznej. Graf sterowania przepływowego (rys. 1) składa się z węzłów reprezentujących operacje. Węzły są połączone łukami, które definiują porządek (częściowy) wykonania operacji. Na łukach mogą występować znaczniki, służące do przenoszenia wartości argumentów między operacjami. Obliczanie jest realizowane przez „odpalanie" (ang. *firing*) węzłów. Węzeł jest przygotowany, jeżeli na każdym z jego łuków wejściowych znajduje się znacznik. Węzeł przygotowany może odpalić. Odpalenie polega na wchłonięciu znaczników z łuków wejściowych węzła, wykonaniu operacji (reprezentowanej przez węzeł) na wartościach argumentów dostarczonych przez znaczniki i wyemitowaniu po jednym znaczniku na każdy łuk wyjściowy węzła. Wartość związana ze znacznikiem na łuku wyjściowym węzła jest rezultatem operacji wykonanej na wartościach dostarczonych przez znaczniki z łuków wejściowych. Na rysunku 2 zaprezentowano przykładowe odpalenie węzła.



Rys. 1. Program obliczania sumy kolejnych liczb całkowitych od k do l według wzoru: k+(k+1)+(k+2)+...+(l−1)+l=(k+l)*(l−k+1)/2, Symbole wewnątrz węzłów oznaczają operacje reprezentowane przez węzły. Węzły oznaczane literą p służą do powielania (kopiowania) argumentów



Rys. 2. Odpalanie węzła reprezentującego operację mnożenia

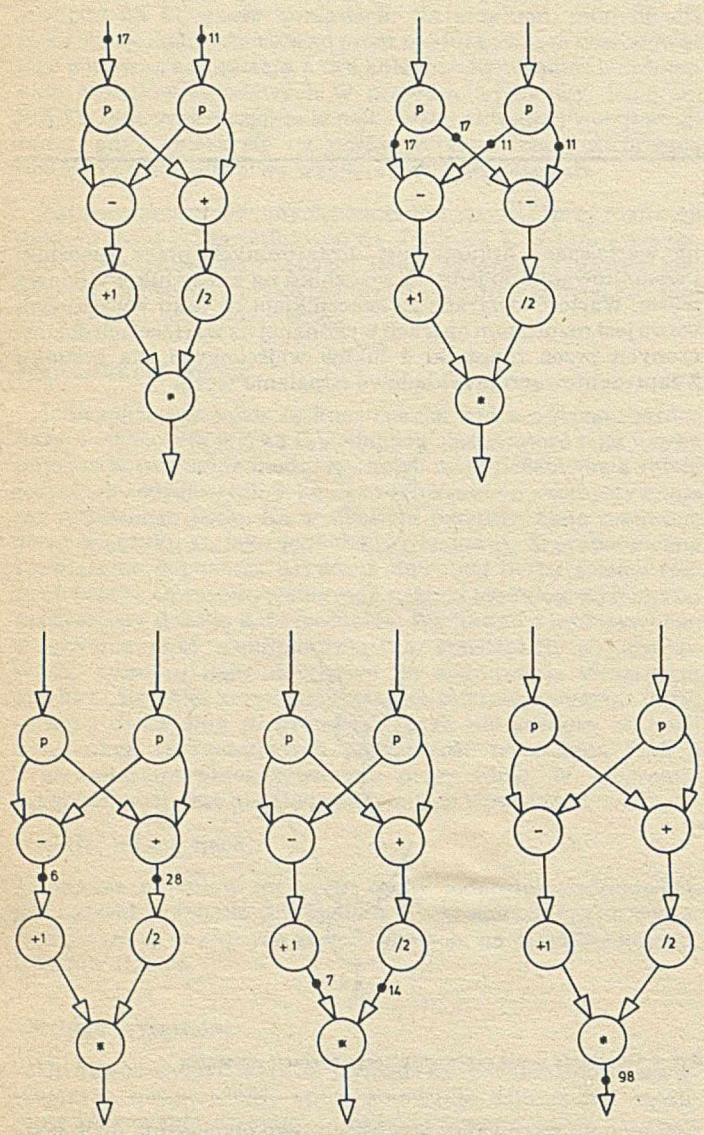
W modelu sterowania przepływowego obowiązują dwie podstawowe zasady, określające reguły przebiegu obliczeń.

- **Operacja jest wykonywana wtedy i tylko wtedy, gdy wszystkie jej argumenty są dostępne.**
- **Wykonanie operacji jest zastosowaniem funkcji reprezentowanej przez węzeł do wartości dostarczonych przez znaczniki; nie powstają żadne efekty uboczne w wyniku odpalenia węzła.**

Jedynym warunkiem wykonania operacji jest dostępność argumentów. Wykonanie operacji jest lokalne – zjawiska zachodzące podczas odpalania węzła dotyczą jedynie łuków związanych z odpalającym węzłem. Odpalaniu węzłów towarzyszy znikanie znaczników z łuków wejściowych i pojawianie się ich na łukach wyjściowych, co z kolei powoduje przygotowanie następnych węzłów; w ten sposób jest wykonywany program. W modelu zakłada się, że odpalenie jest momentalne, to znaczy, że czas jego trwania jest zerowy. Pozwala to na przedstawienie przebiegu obliczeń za pomocą tzw. ujęć migawkowych (ang. *snapshot*). Postęp wykonania programu jest obserwowany przez rozmieszczenie znaczników na łukach grafu programu (rys. 3).

W implementacji modelu sterowania przepływowego nie ma centralnego sterowania. Każda operacja może być wykonana przez dowolny procesor, jeśli tylko ma ona zgromadzone wszystkie argumenty. W praktyce dostępność argumentów nie wystarcza do natychmiastowego wykonania operacji – potrzebny jest

również jakiś wolny element przetwarzający (nie zajęty wykonywaniem innych operacji). Przy ograniczonej liczbie procesorów może wystąpić sytuacja, w której jest wiecej operacji gotowych do wykonania niż wynosi liczba procesorów. Wówczas program jest wykonywany na tyle szybko, na ile pozwalają mu możliwości sprzętowe.



Rys. 3. Ujęcia migawkowe przebiegu obliczenia sumy liczb całkowitych z przedziału ⟨11;17⟩ w grafie z rys. 1; zakłada się, że liczba dostępnych procesorów jest większa lub równa 2

Zdefiniowany program może być zastosowany równocześnie dla kilku kompletów danych wejściowych. Oznacza to, że zanim zostaną uzyskane wyniki odpowiadające pierwszemu kompletowi danych wejściowych, na łuki wejściowe grafu można wprowadzić kolejny komplet itd. Takie podejście jest uzasadnione wówczas, gdy liczba elementów przetwarzających jest większa niż stopień równoległości wykonywanego programu i system nie jest efektywnie wykorzystany. W takiej sytuacji zwiększenie obciążenia systemu poprawi jego przepustowość, ale prawdopodobnie zwiększy czas przetwarzania pojedynczego programu. A bardzo często głównym celem tworzenia różnych systemów wieloprocesorowych jest zminimalizowanie czasu przetwarzania, nawet kosztem spadku efektywności wykorzystania sprzętu.

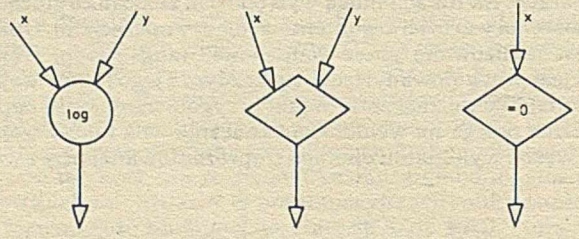
Równoczesne przetwarzanie kilku kompletów danych wymaga wprowadzenia mechanizmu chroniącego przed wymieszaniem się znaczników, należących do różnych procesów obliczeniowych. Węzły nie mogą rozróżnić, do jakiego procesu obliczeniowego należą znaczniki, a łuki służą jedynie do przekazywania argumentów między węzłami (nie porządkują znaczników). Dlatego znaczniki można „kolorować". Każdy komplet znaczników z danymi wejściowymi z różnych procesów obliczeniowych jest oznakowany innym kolorem. Do reguły odpalania dodaje się zastrzeżenie, że węzeł może zostać przygotowany przez komplet znaczników tego samego koloru. Odpalenie węzła zachowuje kolor. Tak więc znaczniki z różnych procesów obliczeniowych są odseparowane w grafie przez cały czas wykonywania operacji programu. Kolorowanie znaczników służy do określania ich kontekstu – w podanym wypadku tym kontekstem jest przynależność do określonego procesu obliczeniowego.

Przy okazji rozważania sposobu zwiększania przepustowości systemu sterowania przepływowego warto wspomnieć o wieloprogramowaniu. Nie stwarza ono żadnych problemów ze względu na brak centralnego sterowania w systemie. Węzły z kilku rozłącznych grafów (różnych programów) są przygotowywane i odpalane zgodnie z regułami modelu. Nie ma żadnej różnicy między wykonaniem jednego programu (logicznie) złożonego z kilku rozłącznych grafów, a wykonaniem kilku programów. W celu optymalnego wykorzystania procesorów systemu sterowania przepływowego suma stopnia równoległości programów (tj. liczby niezależnych instrukcji równocześnie gotowych do wykonania) powinna być równa liczbie procesorów w systemie. Stwierdzenie powyższe oddaje ideę dostosowania równoległości zawartej w algorytmie do możliwości sprzętowych. W praktyce stopień równoległości programu (jednego lub kilku) powinien być jednak większy niż liczba elementów przetwarzających, aby w trakcie wykonywania operacji można było realizować czynności związane z przygotowaniem kolejnych operacji do wykonania. Wynika to z niezerowych czasów odpalania węzłów, przesyłania znaczników między węzłami oraz wyboru węzłów do odpalania.

ELEMENTY PROGRAMOWANIA

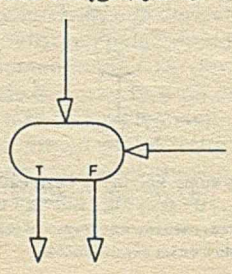
Do podstawowych konstrukcji programowych w językach dostosowanych do modelu sterowania przepływowego (np. w LAPSE [7]) należą:

- funkcja,
- wyrażenie warunkowe: **if..then..else**
- pętla,
- definicja operacji (węzła) i jej makrorozwinięcie,
- rekurencja.



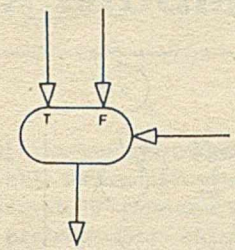
Rys. 4. Węzły reprezentujące dwuargumentową funkcję log x y, dwuargumentowy predykat [x>y] i jednoargumentowy predykat [x = 0]

Funkcja może być reprezentowana przez pojedynczy węzeł. Odpaleniu takiego węzła towarzyszy zastosowanie funkcji do argumentów dostarczonych przez znaczniki na łukach wejściowych. Rezultaty zastosowania funkcji są wartościami znaczników pojawiających się na łukach wyjściowych. W grafie sterowania przepływowego wyróżnia się **predykaty**, tj. funkcje o wartościach logicznych. Są one stosowane do sterowania przebiegiem obliczeń w grafie (programie). Predykaty w grafie przedstawia się w postaci rombów, co je wyodrębnia spośród innych funkcji reprezentowanych przez okręgi (rys. 4).



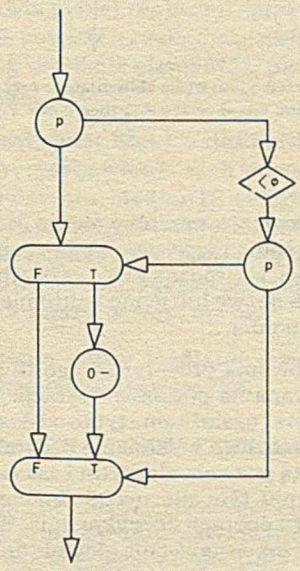
Rys. 5. Dystrybutor

Obliczenia warunkowe if..then..else są realizowane za pomocą węzłów zwanych: dystrybutorem (ang. *distributor*) i selektorem (ang. *selector*). Do odpalenia dystrybutora (rys. 5) jest potrzebny znacznik z dowolną wartością na pionowym łuku wejściowym i znacznik z wartością logiczną na poziomym łuku wejściowym, zwanym łukiem sterującym. Odpalenie dystrybutora powoduje skopiowanie wartości znacznika z pionowego łuku wejściowego na jeden z łuków wyjściowych po wchłonięciu znaczników z łuków wejściowych. Wybór łuku wyjściowego zależy od wartości logicznej z łuku sterującego. Z łukami wyjściowymi są związane oznaczenia **T** i **F** odpowiadające wartościom logicznym odpowiednio „prawda" i „fałsz".



Rys. 6. Selektor

Z dystrybutorem współpracuje węzeł zwany selektorem (rys. 6) Jego działanie jest w pewnym sensie odwrotne do działania dystrybutora. Na łuk wyjściowy jest kopiowany znacznik z jednego z pionowych łuków wejściowych, zgodnie z wartością logiczną znacznika na łuku sterującym (poziomym). Warto nadmienić, że do odpalenia selektora wystarczy, aby znacznik znajdował się tylko na jednym, wybranym pionowym łuku wejściowym (oraz oczywiście na łuku sterującym).



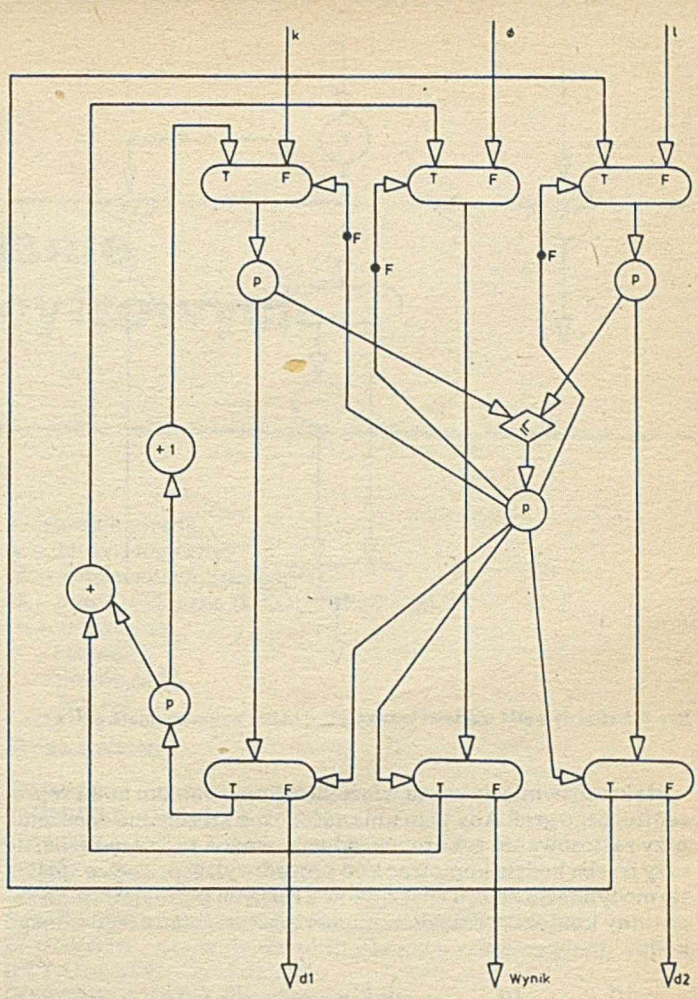
Rys. 7. Graf wyrażenia: if x<0 then −x else x

Na rysunku 7 przedstawiono zastosowanie dystrybutora i selektora do realizacji wyrażenia warunkowego:

if x<0 **then** -x **else** x

odpowiadającego obliczaniu wartości bezwzględnej liczby.

Dystrybutor i selektor znajdują również zastosowanie do tworzenia **pętli.** Przykład pętli służącej do obliczania sumy liczb naturalnych od **k** do **l** (wyłącznie) przedstawiono na rys. 8. Dla przejrzystości nie umieszczono węzłów służących do modyfikowania kolorów znaczników, w przedstawionym przykładzie nie ma bowiem niebezpieczeństwa wymieszania się znaczników z różnych kroków iteracji. W ogólnym wypadku jest to możliwe, dlatego też po każdym z selektorów, organizujących obliczenia w pętli, powinien znajdować się węzeł nadający odpowiedni kolor znacznikom w kolejnych krokach iteracji. Kolor ten



Rys. 8. Graf programu obliczania sumy liczb całkowitych od k do l (włącznie) w pętli. W języku LAPSE program odpowiadający temu grafowi ma postać:

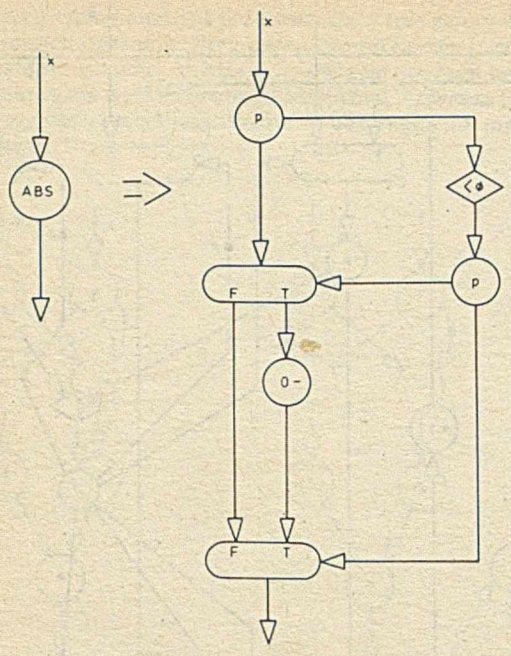
```
iteration suma (x, y, pom);
while old x =< old y
do{x, y, pom} := {old x + 1, old y, old pom + old x} od;
begin
{d1, d2, wynik} := suma (k, l, 0)
end;
```

W języku LAPSE słowo kluczowe old oznacza wartość poprzedniego kroku iteracji, a nawiasy klamrowe grupują wartości w rekordy

powinien jednoznacznie definiować kontekst. Nie należy stosować zwykłej inkrementacji albo dekrementacji jakiejś wartości koloru związanej ze znacznikiem, ponieważ mogą występować pętle zagnieżdżone jedna w drugiej. Łuki dystrybutorów wyprowadzające znaczniki z rezultatami obliczeń iteracyjnych powinny wskazywać węzły odtwarzające kolor znaczników, jaki miały one przed wejściem do pętli. Opisy technik generowania kolorów znaczników można znaleźć w [1, 8].

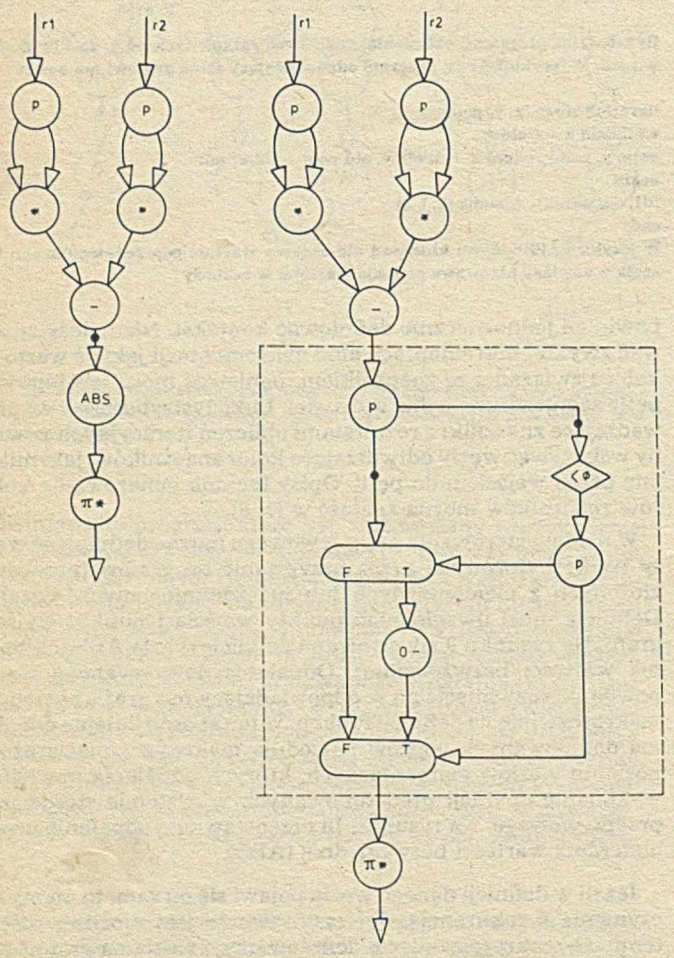
W modelu sterowania przepływowego można definiować **węzły funkcji złożonych** przez przypisanie im grafów (procedur) złożonych z elementarnych lub już zdefiniowanych węzłów. Definicja musi uwzględniać punkty wejścia i punkty wyjścia grafu. Na rysunku 9 zilustrowano definicję węzła ABS (obliczania wartości bezwzględnej). Odpalenie definiowanego węzła powoduje rozwinięcie go w odpowiadający mu graf – następuje makrorozwinięcie definicji funkcji. W ten sposób kolejne odpalania definiowanych węzłów powodują makrorozwinięcia aż do poziomu węzłów elementarnych, których odpalenia powodują wykonanie operacji predefiniowanych w systemie sterowania przepływowego. Na rysunku 10 przedstawiono odpalenie węzła obliczania wartości bezwzględnej (ABS).

Jeżeli w definicji danego węzła pojawi się on sam, to mamy do czynienia z rekurencją. Jej zastosowanie jest możliwe dzięki temu, że makrorozwinięcie definiowanego węzła na graf następuje dopiero w chwili odpalenia tego węzła. Na rysunku 11 przedstawiono graf obliczania n-tego wyrazu ciągu Fibonacciego.

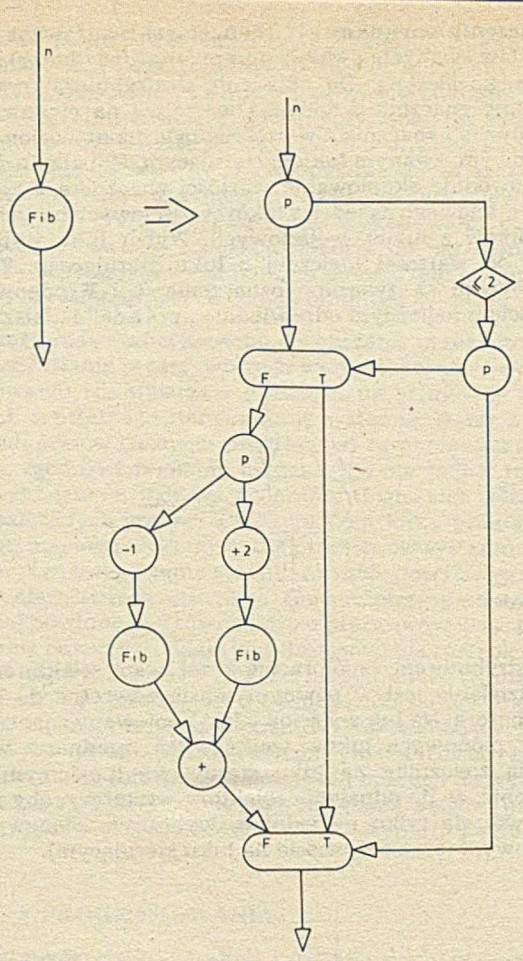


Rys. 9. Definicja węzła wartości bezwzględnej (ABS) wykorzystująca graf z rys. 7

Makrorozwinięcie węzła wiąże się z kopiowaniem kodu reprezentującego graf. Aby tego uniknąć (co ma szczególne znaczenie przy zastosowaniu rekurencji, gdy nie można z góry określić, ile razy trzeba będzie kopiować kod procedury), w praktyce stosuje się modyfikację koloru znaczników z różnych poziomów rekurencji (inny kontekst), dzięki czemu można korzystać z tego samego kodu.



Rys. 10. Makrorozwinięcie węzła ABS (zdefiniowanego na rys. 9) w trakcie odpalenia; Prezentowany graf ilustruje program obliczenia powierzchni pierścieni o promieniach r1 i r2 (bez określania, który z nich jest wewnętrzny)



Rys. 11. Definicja (rekurencyjna) węzła obliczającego n-ty wyraz ciągu Fibonacciego zgodnie z programem: Fib(n) := **if** n = ⟨2 **then** n **else** Fib(n − 1) + Fib(n − 2)

PROGRAM

W modelu sterowania przepływowego każdy program jest funkcją. Nie występuje pojęcie zmiennej; zamiast niego stosuje się pojęcie wartości. Nie można kilku różnych wartości identyfikować przez jedną nazwę lub jedno położenie w pamięci, jak dla modelu von Neumanna.

W modelu sterowania przepływowego równoległość programu jest zdefiniowana na poziomie operacji. Wykonanie każdej operacji jest lokalne i uzależnione tylko od dostępu argumentów. Dzięki temu jest możliwe wykonanie programu z maksymalną szybkością, na jaką pozwala algorytm i zasoby sprzętowe systemu. W stwierdzeniu tym nie uwzględniono jednak losowego charakteru wyboru operacji do wykonania. Wybór ten może nie być optymalny, przez co nawet dla idealnie dobranego programu dla danego systemu, w trakcie wykonania mogą wystąpić chwilowe niedopasowania stopnia równoległości programu i sprzętu, co w danym przebiegu obliczeń powoduje spadek maksymalnej efektywności. Kolejne wykonanie tego samego programu może mieć już przebieg idealny pod względem efektywności. Wspomniane stwierdzenie pomija również konieczność uwzględnienia czasów przesłań znaczników i kojarzenia argumentów, które w rzeczywistości są najbardziej istotnymi czynnikami ograniczającymi szybkość przetwarzania w systemie sterowania przepływowego.

* * *

Model sterowania przepływowego w przedstawionej postaci ma ograniczone zastosowania i nadaje się do konstruowania procesora obliczeń funkcjonalnych dla systemu o zastosowaniach uniwersalnych. W pracach nad systemami sterowania przepływowego podejmuje się badania nad możliwością stosowania wartości strukturalnych, realizacją operacji wejścia-wyjścia, obliczeniami niedeterministycznymi, zarządzaniem zasobami. Zagadnienia te nie dają się łatwo ująć w samym modelu.

dokończenie na s. 26

JERZY FRANCZYK
Instytut Elektroniki
Politechnika Śląska
Gliwice

Komputerowe wspomaganie projektowania układów cyfrowych w języku Modlan

Gwałtowny rozwój urządzeń cyfrowych w ostatnich latach, a zwłaszcza opanowanie technologii produkcji układów LSI oraz VLSI, zawierających 10^6 elementów na jednej strukturze, spowodował, iż projektowanie układów i systemów metodami tradycyjnymi staje się praktycznie niemożliwe ze względu na ich złożoność i wielofunkcyjność. Konieczne staje się komputerowe wspomaganie projektowania – CAD (ang. *computer aided design*). Do realizacji CAD projektant musi posiadać podstawowe narzędzie, jakim jest język opisu sprzętu (ang. *computer hardware description language*).

Istniejące języki opisu sprzętu – LOGOS, CAP, PINLAN, APL DS, PHPL i inne – umożliwiają opisanie sprzętu na określonym poziomie (lub kilku poziomach). Żaden z nich jednak nie umożliwia pełnej realizacji sprzętowej – od określenia funkcji, jakie powinien realizować układ czy system, aż do opisu układu na poziomie elementów podstawowych, jak bramki, przerzutniki itp.

W 1983 roku w Instytucie Elektroniki Politechniki Śląskiej opracowano nowy, pozbawiony tych niedogodności język *Modlan*, oparty na Pascalu. Modlan [3] umożliwia opis projektowanego sprzętu na trzech poziomach: strukturalnym, funkcjonalnym i behawioralnym. Poziomy te są blokami (modułami) języka i umożliwiają uszczegółowianie opisu, zwane dalej przechodzeniem na niższy poziom opisu.

W celu zrozumienia istoty Modlanu trzeba zapoznać się z podstawowymi pojęciami tego języka.

PODSTAWOWE POJĘCIA I OZNACZENIA MODLANU

Moduł – jest to fragment układu cyfrowego, który ma dokładnie określone granice w sensie strukturalnym. Struktura wewnętrzna modułu nie jest znana projektantowi. Cała informacja o działaniu modułu jest przekazywana przez jego terminale.
Terminal – stanowi jedyną drogę wymiany informacji między wnętrzem modułu a jego otoczeniem. Przy sprzętowej interpretacji modułu terminal stanowi jego końcówkę.
Połączenie – przenosi informację między terminalami modułów. Może być interpretowane jako ścieżka obwodu drukowanego, przewód itp.
Modlan jest w zasadzie językiem opisu sprzętu, dopuszczalne jest jednak definiowanie modułów programowych. Terminal i połączenie mają wówczas inną programową interpretację.

Podstawowym oznaczeniem Modlanu jest **symbol.** Symbolem może być: identyfikator, liczba, symbol specjalny lub słowo kluczowe.

Symbole specjalne:
; – koniec instrukcji,
+ – suma arytmetyczna,
* – iloczyn arytmetyczny,
/ – iloraz,
– – różnica,
$ – suma logiczna,
& – iloczyn logiczny,
ⓐ – konkatenacja (sklejenie),
– suma wyłączna (EXCLUSIVE OR),
:= – połączenie,
= – równość,
<> – nierówność,
| – lub itd.

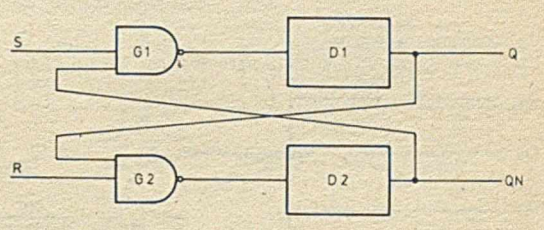
Słowa kodowe:

SMODULE – opis strukturalny,
FMODULE – opis funkcjonalny,
CMODULE – opis behawioralny,
BEGIN – rozpoczęcie opisu,
END – zakończenie opisu,
PARBEGIN – otwarcie nawiasu dla akcji równoległej,
SEQBEGIN – otwarcie nawiasu dla akcji sekwencyjnej,
DATAGRAPH – graf danych,
CONTROLGRAPH – graf sterowania,
DELAY – opóźnienie (czas propagacji) itd.

BLOK OPISU STRUKTURALNEGO

Opis strukturalny jest najbardziej szczegółowym poziomem opisu sprzętu, jeżeli jako moduły zostaną przyjęte elementy podstawowe (bramki **AND, NAND, OR, NOR, INV, XOR, BUFE**), z możliwością określenia rodzaju używanych bramek np.: **ANDOC** – bramka **AND** z otwartym kolektorem, **BUFEO** – bramka trójstanowa z niskim aktywnym sygnałem sterowania itp.

Istnieje również możliwość opisu strukturalnego na wyższym poziomie, jeżeli jako moduły zostaną przyjęte uprzednio zdeklarowane (na poziomie strukturalnych, funkcjonalnym lub behawioralnym) elementy, układy lub nawet całe systemy. Opis strukturalny składa się z definicji modułu, w której są zawarte również terminale (w kolejności: wejściowe, wyjściowe, dwukierunkowe), deklaracji elementów (nazwa-numer) i opisu połączeń.



Rys. 1. Przerzutnik S-R

```
SMODULE SRFF (S,R:Q,QN);
% Definicja modułu (wejścia:wyjścia)
BEGIN
DELAY (1-2,2-3)D1,D2;
% Deklaracja modułu opóźnień: 1–2 umownych jednostek czasu
% na zboczu narastającym i 2–3 na zboczu opadającym
```

```
NAND 2 G1,G2;
% Deklaracja modułów podstawowych – bramki NAND
% dwuwejściowe
G1: = S,D2;
G2: = R,D1;
D1: = G1;
D2: = G2;
Q: = D1;
QN: = D2;
% Deklaracja połączeń
END SRFF
```

Wydruk 1

Model opisu strukturalnego podano na przykładzie opisu najprostszego, asynchronicznego przerzutnika (rys. 1).

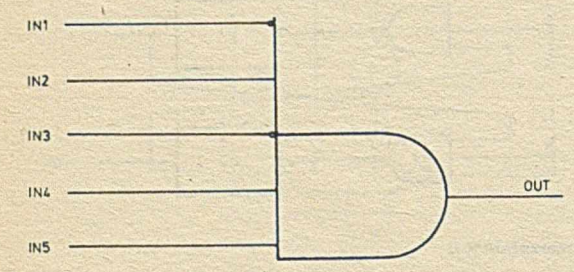
Czasy propagacji bramek **G1** i **G2** przedstawiono jako nie istniejące w rzeczywistości bloki opóźnień **D1** i **D2**. Wszystko, co znajduje się po znaku **%** stanowi komentarz i jest ignorowane przez komputer podczas realizacji programu, natomiast jest umieszczane w wydruku.

BLOK OPISU FUNKCJONALNEGO

W opisie funkcjonalnym projektanta nie interesuje struktura wewnętrzna modułu. Moduł jest definiowany w kategoriach funkcji przez niego realizowanych, zależności czasowych między sygnałami wejściowymi i wyjściowymi oraz transformacji jego stanów wewnętrznych. Blok opisu funkcjonalnego składa się z definicji struktur danych modułu oraz opisu funkcji realizowanych przez moduł. Podstawowymi obiektami opisu funkcjonalnego są rejestry, pamięci, pary rejestrów (np. konkatenacja rejestrów **H** i **L** mikroprocesora 8080 na parę **HL**), zmienne numeryczne oraz opóźnienia przypisane terminalom wyjściowym i dwukierunkowym.

Opis funkcjonalny stanowi instrukcję pojedynczą lub złożoną – szeregową (sekwencyjną) lub równoległą. Równoległa instrukcja złożona umożliwia nieproceduralny zapis operacji realizowanych równolegle. Składa się ona z list instrukcji realizowanych równolegle, przy czym każda lista może być poprzedzona etykietą warunkową, umożliwiającą sterowanie realizacją operacji równoległych.

Jeżeli wartość etykiety jest prawdziwa **TRUE**, to odpowiadająca jej lista operacji jest wykonywana. Jeżeli aktywne są dwie etykiety (lub więcej), to odpowiadające im listy są realizowane równolegle. Przesłania równoległe są realizowane w ten sposób, że po wykonaniu listy (list) operacji obiekty przesłania są uaktualniane na nowe wartości. Każda lista operacji musi być ograniczona nawiasami: **PARBEGIN** – rozpoczęcie i **PAREND** – zakończenie. Sekwencyjne instrukcje złożone są to instrukcje, w których wykonanie następnej listy operacji zależy od wyników realizacji poprzedniej listy operacji. Listy operacji w sekwencyjnych instrukcjach złożonych oddziela się nawiasami: **SEQBEGIN** – rozpoczęcie i **SEQEND** – zakończenie.



Rys. 2. Pięciowejściowa bramka AND z zanegowanym pierwszym i trzecim wejściem

Jako przykład najprostszego opisu funkcjonalnego podano opis nietypowej pięciowejściowej bramki **AND** o pierwszym i trzecim wejściu zanegowanym (rys. 2).

```
FMODULE AND5 (IN[5] : OUT);
% Definicja struktury danych modułu – 5 wejść IN i jedno
% wyjście OUT
BEGIN
OUT← 'IN1&IN2&'IN3&IN4&IN5;
%Opis funkcji realizowanej przez układ: na wyjście OUT jest
%podawany iloczyn sygnałów na wejściach 2, 4 i 5
%oraz zanegowanych sygnałów na wejściach 1 i 3
END;
```

Przykładem bardziej skomplikowanego (na wyższym poziomie) opisu jest fragment bloku funkcjonalnego mikroprocesora 8080 na poziomie instrukcji.

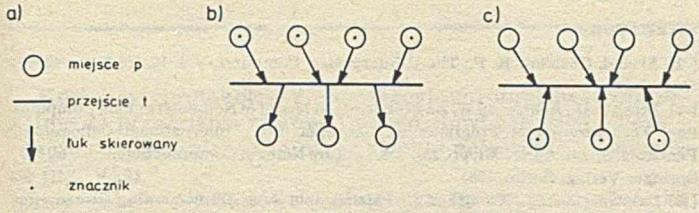
```
FMODULE COMP 8080 (INT :: % Wejście przerwania
                   D [8]); % Ośmiobitowa, dwukierunkowa
                           % magistrala danych
BEGIN
MEMORY: MEM (65536, 8), RB[8, 8], RBB [4, 16],
        FLAG [4, 1]
        % MEM – pamięć zewnętrzna (do 65KB, 8 bitów)
        % RB – rejestry procesora RB 0 = B..., RB 7 = A
        % RBB – pary rejestrów BC, DE, HL oraz SP
REGISTER : PC [16], SP[16], TEMP [16], ABUF [16], I [8],
           SRC [8], AK [8], ARG [8], INTE, HALT, ZERO,
           CARRY, PARITY, PR, SIGN, ACARRY;
COMMON CARRIER: RB [0]ⓐRB [1] = RBB [0], % Para BC
                RB [2]ⓐBR [3] = RBB [1], % Para DE
                RB [4]ⓐRB [5] = RBB [2],% Para HL
                SP = RBB [3]; % Wskaźnik stosu
COMMON CARRIER: FLAG [0] = ZERO, FLAG [1] = CARRY,
                FLAG [2] = PARITY, FLAG [3] SIGN;
                % Wskaźniki stanu (Flags)
MACRO: A = RB [7], N2B = MEM [PC + 2] MEM[PC + 1]
SEQBEGIN
IF INTE INT THEN PARBEGIN
                 I←D; INTE←0; HALT←0
                 PAREND
           ELSE IF HALT THEN→FIN
                        ELSE I←MEM [PC]
ON I [7-6]
CASE 00B DO
ON I 2-0
CASE 000B DO
CASE 001B DO                                        %NOP
  SEQBEGIN
  IF I [3] THEN→DAD
  RBB [I (5-4)]←MEM PC +2ⓐMEM [PC + 1]              %LXI
  PC←PC + 3; ←FIN;
DAD: CARRY|ⓐHL←HL + RBB [I[5-4]];                   %DAD
  →FNB
  SEQEND
CASE 010 DO
  ON I [5-3]
  CASE 0,2 DO MEM [RBB[I [4]]]← A                   %STAX
  CASE 1,3 DO MEM [RBB[I [4]]]                      %LDAX

END
CASE 01B DO SEQBEGIN

            ON I [5-0]                              %MOV
            CASE110110B DO HALT←1                   %HLT
            END


END:
FNB: PC←PC + 1
FIN: SEQEND
END COMP 8080 DESCRIPTION BLOCK;
```

Wydruk 2

SIECI PETRIEGO

Aby zrozumieć istotę opisu behawioralnego trzeba poznać najbardziej elementarne pojęcia i zasady działania sieci Petriego

(SP). Sieć Petriego jest czwórką **SP = (P,T,A,Mo)**, gdzie: **P** – jest skończonym, niepustym zbiorem miejsc (ang. *places*) p_i dla $1 \leqslant i \leqslant n$; **T** – jest skończonym, niepustym zbiorem przejść (ang. *transitions*) t_i dla $1 \leqslant i \leqslant n$; **A** – jest skończonym zbiorem skierowanych krawędzi, łączących miejsca z przejściami i przejścia z miejscami (łączenie przejścia z przejściem i miejsca z miejscem jest zabronione); **Mo** – jest funkcją określającą tzw. znakowanie początkowe sieci Petriego, tj. przyporządkowującą każdemu miejscu pewną liczbę znaczników (rys. 3).



Rys. 3. Sieci Petriego: a) elementy, b) przejście przed wzbudzeniem, c) przejście po wzbudzeniu

Podstawowa zasada działania sieci Petriego jest następująca. Dla każdego przejścia jest określany zbiór miejsc wejściowych i zbiór miejsc wyjściowych. Przejście może zostać wzbudzone, jeżeli w każdym jego miejscu wejściowym znajduje się co najmniej jeden znacznik. Wzbudzenie przejścia powoduje usunięcie z każdego miejsca wejściowego po jednym znaczniku i umieszczenie w każdym miejscu wyjściowym również po jednym znaczniku.

Ze względu na swoją abstrakcyjność sieć Petriego umożliwia przedstawienie wszelkiego rodzaju akcji lub łańcuchów akcji, których realizacja zależy od występowania określonych warunków. Akcjami tymi mogą być procesy sterowania, reakcje chemiczne, procesy produkcyjne itp. Bliższe wiadomości na temat sieci Petriego Czytelnik może znaleźć w literaturze, zarówno krajowej, jak i zagranicznej [1,2,4].

BLOK OPISU BEHAWIORALNEGO

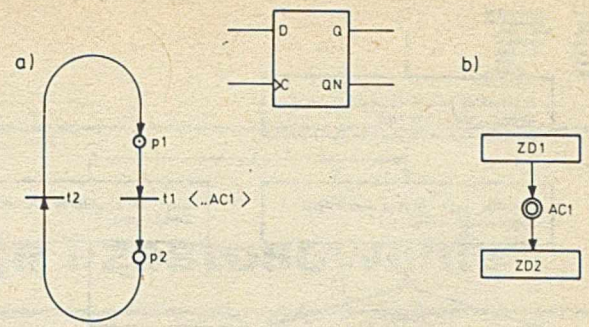
Opis behawioralny modułu (systemu cyfrowego) jest realizowany przy użyciu modułu rozdzielonego sterowania i operacji wykonywanych oraz interpretacji. Sterowanie w opisie behawioralnym jest przedstawione w postaci grafu sterowania (ang. *control graph*), który stanowi czasową synchronizowaną bezpieczną sieć Petriego wolnego wyboru **CSBSPWW.** Przejścia w grafie sterowania są interpretowane jako poszczególne instrukcje programu sterowania projektowanym modułem, a ich wzbudzenie – powodujące przenoszenie znaczników z miejsc wejściowych do wyjściowych – obrazuje dynamikę układu przez wykonanie kolejnych instrukcji.

Operacje wykonawcze są przedstawione w postaci grafu danych **DG** (ang. *data graph*), który jest trójką **DG = (AC,ZD,AD)**, gdzie: **AC** – jest skończonym niepustym zbiorem aktywacji, **ZD** – jest skończonym niepustym zbiorem zbiorów danych, **AD** – jest skończonym zbiorem łuków skierowanych łączących zbiory danych z aktywacjami i aktywacje ze zbiorami danych, obrazującym kierunek przepływu danych. Interpretacja stanowi opis **aktywacji** i warunków wzbudzeń przejść za pomocą słów kodowych i symboli specjalnych.

Jako przykład bloku behawioralnego podano opis przedstawionego na rys. 4 synchronicznego przerzutnika zatrzaskowego **D** (ang. *latch*) o aktywnym wysokim poziomie sygnału zegarowego **C** (ang. *clock*).

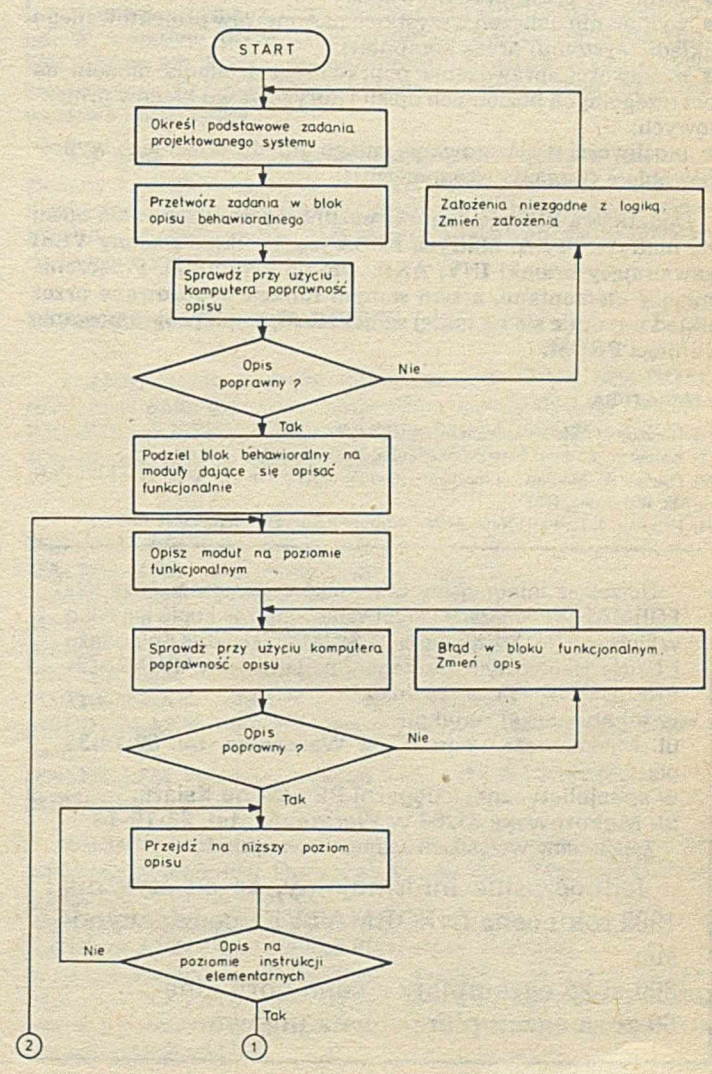
INTERPRETATION % Interpretacja
ACTIVATION AC1
C←1
END
END INTERP.AC1

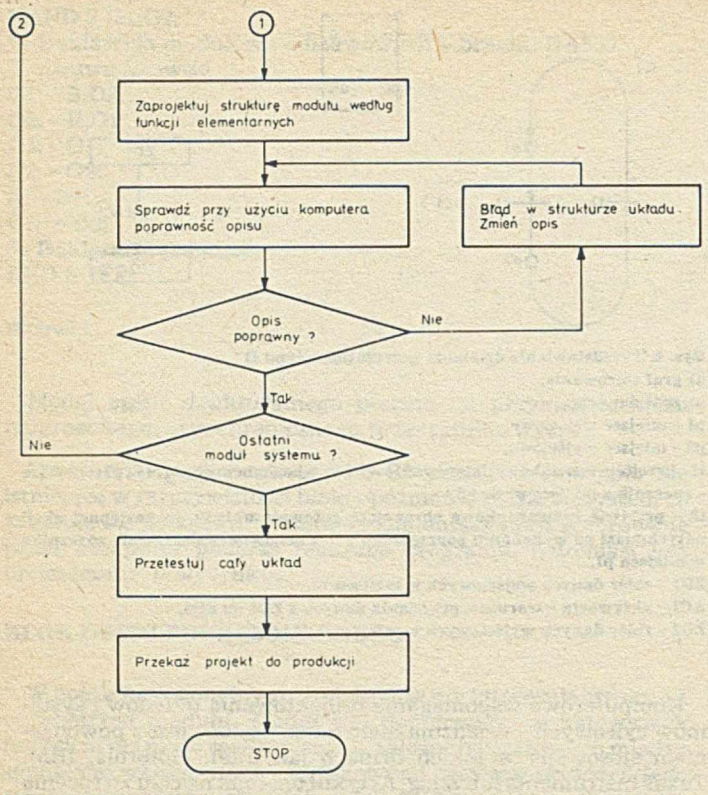
Kolejność czynności wykonywanych przez projektanta podczas projektowania i kontroli poprawności działania układu na poszczególnych poziomach opisu przedstawiono na rys. 5.



Rys. 4. Przedstawienie działania przerzutnika typu D
a) graf sterowania,
b) graf danych;
p1 – miejsce wejściowe,
p2 – miejsce wyjściowe,
t1 – przejście warunkowe – sterowanie wysokim poziomem sygnału C, przenoszące – znacznik z miejsca p1 do p2,
t2 – przejście bezwarunkowe obrazujące gotowość układu do następnej akcji natychmiast po wykonaniu poprzedniej, przez ponowne umieszczenie znacznika w miejscu p1,
ZD1 – zbiór danych wejściowych w miejscu D,
AC1 – aktywacja – warunek przesłania danych z ZD1 do ZD2,
ZD2 – zbiór danych wyjściowych w miejscach Q i QN

Komputerowe wspomaganie projektowania układów i systemów cyfrowych – dziedzina mało u nas znana – jest z powodzeniem stosowana w takich firmach jak Intel, Motorola, IBM, Texas Instruments lub Zilog. Artykuł ten miał na celu zwrócenie uwagi konstruktorów sprzętu cyfrowego na tę metodę pracy przynoszącą ogromne oszczędności czasowe i ekonomiczne. Korzyści wynikające ze stosowania komputerowego wspomagania projektowania polegają na:





Rys. 5. Algorytm pracy konstruktora

- uproszczeniu procesu projektowania;
- skróceniu czasu projektowania;
- wykonaniu obliczeń wszystkich parametrów projektowanego układu (systemu) przez komputer;
- możliwości sprawdzenia poprawności działania modelu na poszczególnych poziomach opisu i korygowaniu błędów projektowych;
- możliwości przetestowania całego układu przed jego wypróbowaniem (względy ekonomiczne!).

Dodatkową korzyść stanowi możliwość odwzorowania opisu na matryce **FPLA**. Matryca **FPLA** jest to układ scalony **VLSI** zawierający bramki **INV, AND, OR** i przerzutniki. Połączenie między elementami, a tym samym funkcje realizowane przez układ uzyskuje się na takiej samej zasadzie jak programowanie pamięci **PROM.**

LITERATURA

[1] Gondzio M.: Sieci Pertiego. Informatyka, nr 1,2, 1987
[2] Kotow W. E.: Sieti Petri. Wyd. Nauka, Moskwa, 1984
[3] Pawlak A.: Modlan – a hardware module description language. Prace IPI PAN, nr 532, Warszawa, 1983
[4] Peterson J. L.: Petri Nets. ACM Computing Surveys, September 1977.

Sterowanie przepływowe

dokończenie ze s. 22

Niektóre z nich można rozwiązać w nieco innych wersjach modelu sterowania przepływowego [2, 5, 9] lub jego modyfikacjach [11, 12], inne są rozwiązywane na poziomie implementacji, bez odwoływania się do ogólnych zasad modelu [3, 4, 6]. Trudności w jednolitym rozwiązaniu tych problemów ograniczają rozwój systemów sterowania przepływowego jako wieloprocesorowych systemów powszechnego zastosowania.

LITERATURA

[1] Arvind, Gostelow K. P.: The U-Interpreter. Computer, Vol. 15, No 2, pp. 42–49, 1982
[2] Brock J. D., Ackreman W. B.: Scenarios – A Model of Non-determinate Computation. Formalization of Programming Concepts. Proc. International Colloquium, Peniscola, Spain, April 1981, pp, 252–259, Lecture Notes in Computer Science, Vol. 107, Springer-Verlag, Berlin, 1981
[3] Caluwaerts L. J., Debacker J., Peperstraete J. A: Implementing Streams on a Dataflow Computer System with Paged Memory. Proc. 40th Ann. Symp. on Computer Architecture. SIGARCH Newsletter, Vol 11, No. 3, pp. 76–83, 1983
[4] Catto A. J., Gurd J.: Nondeterministic Dataflow Programming. Proc. Sixth ACM European Regional Conference, Westbury House, pp. 435–444, April 1981
[5] Davis A. L., Keller R. M.: Data Flow Program Graphs. Computer, Vol. 15, No. 2, pp. 26–41, 1982
[6] Dennis J. B.: First Version of a Data Flow Procedure Language. Lecture Notes in Computer Science, Vol. 19, pp. 362–376, Springer-Verlag, Berlin, 1974
[7] Gurd J., Glauert J., Kirkham C. C.: Generation of Dataflow Graphical Object Code for the Lapse Programming Language. CONPAR' 81, Lecture Notes in Computer Science. Vol. 111, pp. 155–168, Springer-Verlag, Berlin 1981
[8] Gurd J., Watson I., Glauert J.: Multilayered Data Flow Computer Architecture. Department of Computer Science, University of Manchester, 1980
[9] Jennings S. F., Oldehoeft A. E.: An Analysis of Program Execution on a Recursive Stream-oriented Data Flow Architecture. The Journal of Systems and Software, Vol. 3, No. 2, pp. 147–154, June 1983
[10] Podraza R.: Model sterowania przepływowego. Podstawy Sterowania, T. 15, Z. 3, s. 153–170, 1985
[11] Podraza R.: Niedeterminizm w modelu sterowania przepływowego. Podstawy Sterowania, T. 16, Z. 2, s. 167–184, 1986
[12] Treleaven P. C., Hopkins R. P., Rautenbach R. W.: Combining Data Flow and Control Flow Computing. Computer Journal, Vol. 25, No. 2, pp. 207–217, 1982
[13] Treleaven P. C.: Decentralized Computer Architectures for VLSI. VLSI Architecture. B. Randall, P. C. Treleaven Eds. pp. 348–380, Prentice-Hall, Englewood Cliffs (NJ), 1983.

KERMIT

dokończenie ze s. 28

Przykładowo, polecenia przesłania plików z IBM PC przez KERMIT do komputera macierzystego (np. SM-4) mają następującą postać:

⟩KER R:	polecenie odbioru wydane do SM-4
⟩CTRL –]	
C:	powrót do pracy z PC
KERMIT-MS⟩SEND,	*nazwy_plików*

* * *

Niniejszy artykuł zawiera jedynie ogólne omówienie protokołu **KERMIT**, jego implementacji dla IBM PC i zastosowania. Jest to w zasadzie wystarczające do samodzielnego posługiwania się programem KERMIT-MS i może stanowić podstawowy materiał do zaimplementowania tego protokołu na własnym komputerze. Pełniejsze omówienie Kermita można znaleźć w cytowanej literaturze [3].

Warto także podkreślić, że większość implementacji **KERMITA** należy do tzw. *public domain software*, tzn. jest rozpowszechniana bezpłatnie i udostępniania jedynie za cenę nośnika.

LITERATURA

[1] da Cruz F. et al.: MS-DOS KERMIT Documentation. Version 2.26, Columbia University, July 1984
[2] da Cruz F., Catchings B.: Kermit – A File-Transfer Protocol for Universities. Parts 1/2. Byte, p.255, No. 6, 1984; p. 143, No. 7, 1984
[3]da Cruz F.: Kermit – A File Transfer Protocol. Digital Press, Bedford (MA), 1987.

JANUSZ ZALEWSKI (oprac.)
Warszawa

Kermit
organizacja, implementacja i zastosowanie

Kermit jest protokołem komunikacyjnym opracowanym specjalnie do przesyłania plików. Do jego głównych cech należą:
- łączność półdupleksowa przez zwykłe połączenia terminalowe (np. V24),
- ukierunkowanie bajtowe (najmniejszą przesyłaną jednostką informacji jest znak ASCII),
- transmisja tylko w kodzie ASCII (inne kody muszą być poddane konwersji),
- długość pakietu w zasadzie ograniczona do 96 znaków,
- przesyłanie pakietów z potwierdzeniem.

W niniejszym artykule przedstawiono budowę i ważniejsze właściwości protokołu Kermit, omówiono jego implementację na mikrokomputery IBM PC i podano przykłady użycia do realizacji niektórych funkcji komunikacyjnych. Tekst artykułu oparto na referacie wygłoszonym podczas obrad X Szkoły Mikroprocesorowej w Łodzi, w listopadzie 1987 roku. Pełniejszy opis protokołu i jego implementacji można znaleźć w literaturze [1-3].

BUDOWA PAKIETU

Pakiet w protokole Kermit jest to ciąg bajtów (stanowiących znaki ASCII), umieszczonych na sześciu polach w następującej kolejności (rys.):
- początek pakietu (określony nazwą MARK), tj. znak ASCII-SOH (CTRL-A), który może być zmieniony programowo;
- liczba wszystkich znaków ASCII w reszcie pakietu (LEN, ang. *length*), włącznie z prefiksami i sumą kontrolną (p. poniżej); maksymalna długość pakietu wynosi 94 + 2 znaki, co wynika z ogólnej liczby 128 znaków możliwych do przedstawienia na 7 bitach, pomniejszonej o 32 znaki sterujące;
- numer kolejnego pakietu (SEQ, ang. *sequence*) modulo 64 (tj. od 0 do 63), służący do zabezpieczenia przed zgubieniem pakietu;
- typ pakietu (TYPE) oznaczany przez pojedynczy znak ASCII, według następującej zasady:

D – dane,
Y – potwierdzenie (ACK),
N – odrzucenie (NAK),
S – inicjowanie nadawania,
R – inicjowanie odbioru,
B – koniec transmisji (EOT),
F – nagłówek pliku,
Z – koniec pliku (EOF),
E – błąd,
G – polecenie komputera macierzystego, ang. *host* (pole danych zawiera nazwę polecenia do wykonania przez komputer macierzysty),
X – nagłówek tekstu przeznaczonego do wyświetlania na ekranie,
G – polecenie rodzajowe, oznaczające, że pełen kod polecenia jest zawarty w polu danych (np. **L** – wyrejestrowanie się, ang. *logout*, **F** – zakończenie pracy bez wyrejestrowania, **D** – zapytanie o zawartość skorowidza, **U** – zapytanie o użycie dysku, **E** – usunięcie pliku, **T** – wyświetlenie zawartości pliku itp.);

- dane (DATA) stanowiące treść pakietu; niedrukowalne znaki ASCII muszą być poprzedzone znakiem specjalnym (#) i przekształcone na znaki drukowalne przez dopełnienie siódmego bitu; znaki z istotnym ósmym bitem mogą być również poprzedzone innym znakiem specjalnym (&); znaki powtarzane wielokrotnie mogą być zastąpione jednym wystąpieniem tego znaku poprzedzonym znakiem specjalnym trzeciego rodzaju (wężykiem);
- suma kontrolna (CHECK) o długości 1-3 znaków odnosząca się do całego pakietu z pominięciem znaku początkowego i samego znaku sumy kontrolnej; stanowi drukowalny znak ASCII kodowany według tej samej reguły co znaki danych.

MIEJSCE KERMITA W MODELU ISO/OSI

Budowa pakietu stanowi podstawę do opracowania odpowiedniego algorytmu transmisji, tj. pełnego protokołu. Współczesne protokoły mają budowę warstwową, opartą na modelu odniesienia ISO/OSI (ang. *open system interconnection*). Podobnie budowa pakietu Kermita ma silny związek z architekturą systemu ISO/OSI.

Warstwa fizyczna w każdym systemie prowadzi transmisję na poziomie pojedynczych bitów i jest realizowana sprzętowo, np. przez sprzęt V24. Dwa pierwsze pola pakietu Kermita, MARK i LEN, i pole ostatnie, CHECK, są odpowiedzialne za detekcję i odebranie pakietu oraz za kontrolę i potwierdzenie odbioru. Są to funkcje warstwy łącza danych. Warstwa sesji używa numeru pakietu SEQ w celu powtórzenia transmisji brakującego pakietu lub usunięcia nadmiarowego, aby zawartość pliku była zachowana. W warstwie prezentacji następuje konwersja kodu (np. EBCDIC na ASCII itp.) oraz konwersja lub usuwanie niektórych znaków (np. CR lub LF). Typ pakietu i dane są odbierane przez warstwę aplikacyjną.

Dokładny opis algorytmów nadawania i odbioru jest dość złożony i przekracza ramy tego artykułu. Ich znajomość jest niezbędna jedynie dla wąskiej grupy osób implementujących Kermita. Dlatego najbardziej zainteresowanych warto odesłać do literatury źródłowej [3].

IMPLEMENTACJA DLA PC-DOS

Implementacja Kermita dla systemu PC-DOS (MS-DOS), oznaczana dalej dla odróżnienia od protokołu dużymi literami KERMIT lub KERMIT-MS, składa się w wersji źródłowej 2 z następujących plików niezależnych sprzętowo:

MSKERM.ASM – program główny,
MSCMD.ASM – analizator składni poleceń,
MSSEND.ASM – obsługa polecenia **SEND**,
MSRECV.ASM – obsługa polecenia **RECEIVE**,
MSSERV.ASM – obsługa polecenia **SERVER**,
MSFILE.ASM – obsługa plików,
MSTERM.ASM – obsługa polecenia **CONNECT**,
MSCOMM.ASM – buforowanie portu i sterowanie przepływem danych,
MSSET.ASM – obsługa poleceń **SET**, **SHAW** i **STATUS**,
MSDEFS.H – struktury danych.

W każdej implementacji wymagane jest istnienie wszystkich powyższych modułów. Oprócz nich konieczne są moduły zależne od sprzętu, np. dla IBM PC:
MSXIBM.ASM – kod główny obsługi sprzętu,
MSYIBM.ASM – kod obsługi klawiatury i ekranu.
W czasie asemblowania każdego z modułów, na dysku domyślnym powinien znajdować się plik **MSDEFS.H** Łączenia modułów w wykonywalny program KERMIT dokonuje się typowym konsolidatorem **LINK**.

WYWOŁANIE KERMITA

KERMITA wywołuje się jak typowe polecenie zewnętrzne (tj. ładowalne) systemu operacyjnego PC-DOS, zarówno z pliku wsadowego, jak i bezpośrednio:

```
A>KERMIT
```

Program zgłasza się komunikatem, podając w pierwszym wierszu numer wersji, a w drugim informację o sposobie dostępu do samouczka:

```
IBM PC KERMIT-MS V2. xx
TYPE ? FOR HELP
KERMIT-MS>
```

Natychmiast po wywołaniu wykonują się polecenia zapisane na pliku **MSKERMIT.INI.** Z KERMITEM współpracuje się wydając polecenia opisane w następnych punktach. Powrót z KERMITA do systemu operacyjnego PC-DOS (jak i każdego innego) następuje po wydaniu polecenia EXIT:

```
KERMIT-MS>EXIT
A>
```

WSPÓŁPRACA Z DRUGIM KOMPUTEREM

Współpraca z komputerem zewnętrznym za pomocą programu KERMIT-MS może odbywać się na kilku poziomach złożoności:
- emulacja terminala (ang. *terminal emulation*),
- przesyłanie plików (ang. *file transfer*),
- zarządzenie plikami (ang. *file management*),
- ładowanie skrośne (ang. *down-, uploading*),
- usługodawca (ang. *server function*).

Poniżej omówiono najprostsze z wymienionych funkcji.

Połączenie mikrokomputera PC z komputerem zewnętrznym za pomocą KERMITA musi obejmować dostosowanie niektórych parametrów PC do warunków zewnętrznych. Polega to na zdefinowaniu tych parametrów poleceniem SET, np.:
- SET PORT *liczba*

gdzie *liczba* = 1 lub 2 (COM1 lub COM2);
- SET BAUD *szybkość*

gdzie *szybkość* = 300, 1200, 2400, 4800 itd. (bodów);
- SET PARITY *słowo_kluczowe*

gdzie *słowo_kluczowe* = NONE, ODD, EVEN, MARK lub SPACE (brak parzystości – NONE – umożliwia użycie ósmego bitu każdego znaku jako bitu danych, co pozwala na przesyłanie plików binarnych);
- SET WARNING *ON* lub *OFF*

co polega na wydaniu (lub nie) ostrzeżenia, że odbierany plik ma tak samą nazwę jak plik już istniejący (w ten sposób zapobiega się zniszczeniu zawartości pliku istniejącego).

POLECENIA

Najczęściej używa się poleceń związanych z emulacją terminala i przesyłaniem plików.

Emulacja terminala

Funkcja zwana emulacją terminala polega na naśladowaniu pracy terminala innego, na ogół większego, komputera – tzw. komputera macierzystego (ang. *host*). Zadanie to nie jest zbyt skomplikowane programowo, gdyż jego istotą jest głównie obsługa odpowiednich znaków sterujących, charakterystycznych dla danego komputera macierzystego lub jego systemu operacyjnego.

Funkcję tę wykonuje się wydając po wywołaniu KERMITA polecenie **CONNECT:**

```
KERMIT-MS>CONNECT
{Connecting to host. Type CTRL-] and C to return to PC}.
```

KERMIT odpowiada standardowym komunikatom zawierającym informację o sposobie powrotu do pracy z PC oraz – własnym tekstem podpowiedzi (ang. *prompt*) informującym o swojej gotowości.

Polecenia w trybie emulacji terminala wydaje się używając znaku **ESC**, tj. CTRL-], oraz jednoliterowego kodu operacji, np.:
? – samouczek, tj. wyświetlanie opisu poleceń jednoliterowych,
C – zamknięcie (ang. *close*) połączenia z komputerem zewnętrznym i powrót do pracy z PC,
Q – tymczasowe zaprzestanie (ang. *quit*) rejestrowania przebiegu sesji
R – dokończenie (ang. *resume*) rejestrowania przebiegu sesji.
Wydanie niedozwolonego polecenia z użyciem znaku CTRL-] powoduje wygenerowanie sygnału dźwiękowego.

PRZESYŁANIE PLIKÓW

Najważniejszą funkcją programu KERMIT jest przesyłanie plików. Służą do tego polecenia SEND i RECEIVE.

Polecenie SEND o następującej składni:

```
SEND nazwy_ plików
```

powoduje przesłanie plików o podanych nazwach (oddzielonych odstępami) do komputera zewnętrznego wyposażonego w KERMITA, pod warunkiem że wcześniej wydano mu już polecenie odbioru RECEIVE. Przebieg transmisji jest ilustrowany na ekranie przez podanie m. in. numeru przesyłanego pakietu, kolejnej próby przesłania tego pakietu itp., oraz zasygnalizowanie bezbłędnego zakończenia całej operacji.

Polecenie **RECEIVE** o następującej składni:

```
RECEIVE nazwy_ plików
```

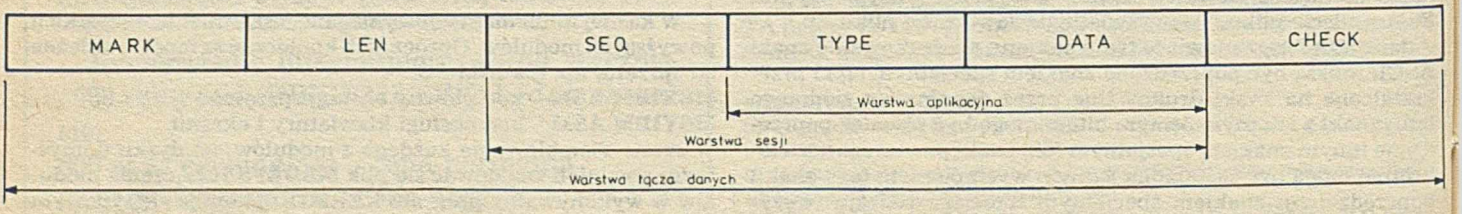
powoduje odebranie plików przesyłanych z innego systemu, pod warunkiem że wydano mu polecenie nadawania SEND. Nazwy odbieranych plików mogą odpowiadać nazwom urządzeń w systemie operacyjnym PC-DOS (np. **NUL, COM1, CON** itd.). Jeżeli odbierany plik nie dotrze w całości, to KERMIT-MS nie zarejestruje go w skorowidzu. Można jednak na to wpłynąć poleceniem:

```
SET INCOMPLETE KEEP
```

które powoduje zachowanie tej części pliku, która dotarła do adresata.

W czasie przesyłania plików można używać znaków sterujących m. in.:
CTRL-X – zaniechanie przesyłania aktualnego pliku i przejście do przesyłania następnego,
CTRL-Z – całkowite zaniechanie przesyłania.
W nazwach plików można używać niejednoznacznych nazw, tak jak w systemie PC-DOS, tzn. gwiazdki w celu zastąpienia wielu znaków, lecz znaku równości „=" zamiast znaku zapytania do zastępowania pojedyńczych liter (ponieważ znak zapytania służy KERMITOWI do wywoływania samouczka).

dokończenie na s. 26



Format pakietu w protokole Kermit

Terminologia systemów operacyjnych (1)

Temat obecnego odcinka rozważań terminologicznych jest dość jednoznaczny, lecz nieco mylący. Kiedy 2-3 lata temu zgromadziłem kilka polskich wydawnictw nt. systemu operacyjnego RSX-11 [1-4], przeznaczonego dla komputerów PDP-11, i zamierzałem poświęcić nieco uwagi polskiej terminologii stosowanej przez poszczególnych autorów, przeszkodziły temu pilniejsze sprawy, które należało podjąć na łamach **INFORMATYKI**. Próbując teraz wrócić do tego tematu spostrzegłem, co narzuca się niedoparcie, że problematyka systemu RSX–11 znacznie się zdezaktualizowała – oczywiście za sprawą rozpowszechnienia mikrokomputerów osobistych, pracujących w Polsce w większości pod nadzorem systemu operacyjnego PC-DOS.

Z tego faktu płynie oczywista nauka. Wszelkie byty informatyczne, niezależnie od tego, czy idzie o sprzęt czy oprogramowanie, by nie ograniczać się tylko do systemów operacyjnych, mają żywot skończony i w większości krótkotrwały. Nie można więc zbytnio zawierzać temu, co aktualnie jest najbardziej popularne, bo nawet jeśli jest czymś więcej niż tylko produktem zwodniczej mody, to i tak przeminie, pozostawiając co najwyżej kilka terminów o trwałej wartości. Uwaga ta dotyczy szczególnie sfery języka, bo choć same wytwory techniki szybko znikają z pola widzenia, to wprowadzone wraz z nimi nazwy pozostają w obiegu dużo dłużej. Ten fakt skłania mnie do tego, aby na tle innych książek o systemach operacyjnych rozważyć niektóre pojęcia systemu RSX-11 i porównując terminologię różnych autorów krytycznie ją zweryfikować. Do rozważań włączę więc jeszcze dwie ostatnio wydane książki, dotyczące systemów operacyjnych [5, 6]. Gwoli ścisłości należy dodać, że wyłączę z tej tematyki terminologię systemu plików, szczegółowo omówioną w numerach 6 i 7 bieżącego rocznika **INFORMATYKI**.

Jednym z najbardziej podstawowych, a jednocześnie najbardziej kontrowersyjnych terminów jest polski odpowiednik angielskiego słowa *command,* oznaczającego **rozporządzenie wydane do systemu operacyjnego przez użytkownika.** Większość autorów tłumaczy to słowo jako **komenda**, lecz stosowane są także inne odpowiedniki, jak **polecenie, rozkaz, zlecenie i dyrektywa,** niestety często używane wraz z wyrazem komenda w identycznym znaczeniu w tym samym tekście. Nie ma wątpliwości, że taki bałagan terminologiczny jest niepożądany i warto zbadać, czy nie dałoby się uporządkować nazewnictwa.

Przede wszystkim, należy wyeliminować wyraz **rozkaz**. W terminologii polskiej rozróżnienie jest jasne – rozkazy dotyczą wyłącznie poziomu sprzętowego, niezależnie od tego, jaki jest oryginał angielski – najczęściej *instruction* lub *command*, rzadziej – *order.*

Z kolei, co rejestrują już w pewnym stopniu nowsze słowniki języka polskiego, wyraz **zlecenie** nabrał we współczesnej polszczyźnie znaczenia wyraźnie urzędniczego. Zleceniem jest nazywana zazwyczaj pisemna umowa na wykonanie określonej pracy. Jej uczestnikami są zleceniodawca i zleceniobiorca. Otwarcie zlecenia, zamknięcie zlecenia, zlecenie wypłaty, zlecenie naprawy, zlecenie na podjęcie pieniędzy, to typowo urzędnicze sformułowania. Spróbujmy coś przedsięwziąć, np. dokonać większego zakupu, bez wystawienia, przyjęcia lub podpisania zlecenia – jest to niemożliwe. Krótko mówiąc, zlecenie jest to pismo zlecające. Radziłbym zdecydowanie, aby pozostawić ten wyraz urzędnikom i ich pięknej biurokratycznej nowomowie, a nie wprowadzać go do wypowiedzi i tekstów informatycznych.

Inny wyraz – **dyrektywa** – jest w polskiej terminologii informatycznej rozumiany tradycyjnie jako instrukcja dla translatora (kompilatora, asemblera itp.) – odpowiednik angielskiego *directive*. Można to znaczenie rozszerzyć i przyjąć, co zresztą jest spotykane w praktyce, że dyrektywą będzie się nazywać także inne instrukcje wydawane z programu do programu, np. przeznaczone dla jądra systemu operacyjnego (do wykonania przez jądro), tzw. wywołania systemowe (ang. *system call, system directive*).

Warto też dodać, że użyty powyżej termin **instrukcja** należy rozumieć jako instrukcję języka programowania, instrukcję w programie – odpowiednik angielskiego *statement.*

Tak więc ostatecznie, jako odpowiednik angielskiego terminu *command* zalecałbym używanie wyrazu **polecenie**. Ta propozycja wynika nie tylko z uzasadnionego wyeliminowania innych wyrazów kandydujących do tej roli. Jest to najbardziej naturalne z dydaktycznego punktu widzenia. Każde ćwiczenie czy zadanie, które dajemy do rozwiązania uczniom w szkole, ma polecenie wyrażające wolę lub intencję nauczyciela lub autora podręcznika. Taką budowę zadań i ćwiczeń wpaja się uczniom w szkołach od najmłodszych klas do ostatnich lat studiów. Nie ma więc powodu, aby w informatyce nie podtrzymać tej dobrej zasady i uniknąć wprowadzenia języka urzędniczego, co zdarzyło się właśnie autorom książki przeznaczonej dla szkół [6].

Nie wydaje mi się natomiast, aby udało się wyeliminować używanie w języku polskim wyrazu **komenda**. Nie sądzę też, by było to niezbędne. Jest to przecież najbardziej naturalne tłumaczenie angielskiego *command*, na dodatek, słowo funkcjonujące już w języku polskim – choć w innych znaczeniach.

Reasumując, wydaje mi się, że najbardziej właściwym terminem polskim na oznaczenie angielskiego wyrazu *command* (w znaczeniu – rozporządzenie wydane do systemu operacyjnego przez użytkownika) jest polski wyraz **polecenie**. Nie protestowałbym jednak zbyt gwałtownie przeciwko używaniu w tym znaczeniu wyrazu komenda, natomiast wszelkie inne wyrazy bym odrzucił.

Sądzę, że na zakończenie tej części rozważań warto podać ogólne zasady dotyczące używania poszczególnych terminów polskich, zależnie od ich rzeczywistego znaczenia, a bez względu na brzmienie oryginalnych nazw angielskich:

1) **rozkazy** (ang. *instruction, command,* rzadziej *order*) dotyczą wyłącznie poziomu sprzętu,
2) w językach programowania nie występują rozkazy, lecz **instrukcje** (ang. *statement* – rzadziej *instruction*),
3) do programów kieruje się **polecenia** (ang.*command*)
4) polecenia wydawane z programu nazywa się **dyrektywami** (ang. *directive, system call,* rzadziej – *command*).

JANUSZ ZALEWSKI

LITERATURA

[1] J. Wiśniewska: Wybrane informacje o systemie operacyjnym RSX-11M. Instytut Informatyki, Uniwersytet Warszawski, 1983

[2] M. Maniecki: System programowania RSX-11. Instytut Informatyki, Politechnika Warszawska, 1984

[3] A. Paprocki, P. Wolański: Podstawowe możliwości systemu RSX-11M. Instytut Maszyn Matematycznych, Warszawa, 1984

[4] J. Deminet: System operacyjny RSX-11. WNT, Warszawa, 1986

[5] W. Iszkowski, M. Kalinowska-Iszkowska, M. Maniecki: Projektowanie systemów operacyjnych w ujęciu syntetycznym. PWN, Warszawa, 1987

[6] W. Cellary, J. Rykowski: System operacyjny CP/J dla mikrokomputera Elwro 800 Junior.

Cztery kompilatory Ady na mikrokomputery PC (2)

W obecnej części omówiono dwa pozostałe kompilatory: AdaVantage i Janus Ada,

AdaVantage

AdaVantage, wersja 1.00, jest dużym podzbiorem Ady o stosunkowo niskiej cenie 129,95 dolara (nowsza atestowana wersja 2.0 kosztuje prawie 800 dolarów). Kompilator udostępnia zaawansowane konstrukcje języka, jak: wielozadaniowość, jednostki rodzajowe i wyjątki. W omawianej wersji kompilatora jednostki rodzajowe są zaimplementowane w sposób podobny do znanych z języka C rozwinięć makrodefinicji. Takie rozwiązanie jest ekonomiczne czasowo, ale nie przestrzennie. Zarządzanie zadaniami nie wykorzystuje kwantowania czasu. Programowanie współbieżne oparto na jednoprocesorowym schemacie szeregowania według priorytetu, z przełączaniem zadań w punktach spotkań i instrukcjach **delay**. Dodatkowo, wymagania odnośnie pamięci obniżają liczbę jednocześnie wykonywanych zadań. Firma informuje, że ta niedogodność zostanie usunięta w przyszłych wersjach kompilatora.

Inne niezaimplementowane mechanizmy języka obejmują: dołączanie kodu maszynowego do postaci źródłowej programu, klauzule reprezentacji i adresów, jak również wejścia przerwań w zadaniach. Dodatkowo, funkcja **Form** użyta w sformatowanych operacjach we-wy zawsze udostępnia pusty napis.

W AdaVantage zaimplementowano typ **long integer** jako rozszerzenie języka. Dla nowicjuszy są przeznaczone dodatkowe pakiety **iio** i **fio** do wykonywania całkowitoliczbowych i zmiennoprzecinkowych operacji we-wy. Te pakiety powstają przez konkretyzację całkowitoliczbowych i zmiennoprzecinkowych procedur rodzajowych pakietu **text_io**. Pakiet ada_io jest prostym sprzężeniem z **text_io** wykonującym terminalowe operacje we-wy.

Na czterech dyskietkach o pojemności 360 KB znajduje się kompilator, konsolidator i zarządca biblioteki – nie ma natomiast edytora. Firma Meridian Software Systems oferuje również dwa pakiety zawierające dodatkowe biblioteki procedur DOS-a i programy usługowe (DOS Environment Package i Utility Package).

Kompilator AdaVantage wykorzystuje system zarządzający biblioteką do przyporządkowania nazw plików jednostkom kompilacji, weryfikowania wzajemych związków między skomplikowanymi jednostkami i ustaleniami kolejności kompilacji, zachowania zgodności między symbolami przypisanymi przez programistę i przypisanymi przez kompilator, a także – do wyboru odpowiedniego kodu wynikowego do konsolidacji.

System AdaVantage zawiera programy umożliwiające tworzenie i incjowanie bibliotek oraz usuwanie i zmianę spisu odwołań w bibliotece. Działanie tych programów jest sterowane odpowiednimi opcjami programowymi. Używając programów usługowych należy za pomocą zarządcy biblioteki dołączyć odwołanie do nowego pakietu. Dzięki temu, podczas kompilacji i konsolidacji zapewniony jest dostęp do bibliotek rozrzuconych po różnych skorowidzach systemu operacyjnego. Kompilator ma również opcje umożliwiające dostęp do określonego pliku danych bibliotecznych, generowanie programu asemblerowego z opcjonalnymi anotacjami i wykonywanie kompilacji w sposób jawny (ang. verbose).

Konsolidator wysokiego poziomu **BAMP** (ang. build Ada main program) ma kilka opcji, np. możliwość wyboru innego pliku danych bibliotecznych niż standardowy **ADA.LIB**, określenie wielkości programu głównego, wielkości stosu dla zadań współbieżnych, niestandardowej nazwy dla programu wykonywalnego i ujawnianie poszczególnych kroków w tworzeniu programu.

Utility Package udostępnia funkcje przestępne, a także zapewnia dostęp do argumentów poleceń systemu operacyjnego i biblioteki obsługi napisów. DOS Environment Package zawiera biblioteki sterowania kursorem i atrybutami obrazu, wyświetlania znaków graficznych, dostępu do rejestracji czasu systemowego, wykonywania działań systemowych na plikach, sterowania wykonaniem programów potomnych oraz przydzielania i zwalniania pamięci.

Janus/ Ada

Janus/ Ada jest jednym z pierwszych podzbiorów Ady zaimplementowanych na mikrokomputery. Aktualnie jest sprzedawany w trzech konfiguracjach: mała **C-Pak**, średnia **D-Pak** i zaawansowana **S-Pak**. Niniejsze omówienie dotyczy **D-Pak**. Janus/ Ada nie ma środowiska wspomagającego tworzenie oprogramowania; programista działa z poziomu systemu operacyjnego. Konfiguracja D-Pak obejmuje kompilator, konsolidator, asembler i disasembler. Programy źródłowe są kompilowane na kod wrodzony. Istnieje możliwość tworzenia plików typu COM dla programów o objętości mniejszej od 64 KB (Mode 10) lub plików EXE w wypadku większych programów (Mode 11). Posiadanie koprocesorów 8087 i 80287 jest opcjonalne, ale zalecane przy dużych obliczeniach.

W podręczniku użytkownika omówiono składnię Ady i porównano ją z implementacją Janus/ Ada. Wytłumaczono, wraz z podaniem przyczyn, wszystkie odstępstwa od normy Ady.

Wywołując kompilator z poziomu operacyjnego można podać dowolną kombinację 13 opcji. Wiele z tych opcji można również określić za pomocą dyrektyw umieszczonych w programie źródłowym. Opcje kompilatora obejmują podawanie skróconych komunikatów błędów, zaniechanie wytwarzania kodu dla programu uruchomieniowego, generowanie instrukcji mikroprocesora 8087 zamiast wywołań bibliotecznych, tworzenie plików wydruku bądź uaktywnienie optymalizacji.

Janus/ Ada wyświetla dużo informacji o przebiegu kompilacji. Komunikaty błędów są jasne, a w podręczniku zawarto dodatkowe wyjaśnienia. Konsolidator Janus/ Ady o nazwie **JLINK** również posiada wiele opcji kierujących procesem konsolidacji. Jedna z nich polega na wyborze biblioteki zmiennoprzecinkowej (działania na liczbach zmiennoprzecinkowych krótkich lub normalnych, przy udziale lub bez koprocesora 8087). Inne opcje obejmują ustawianie początkowego adresu programu, ustalanie danych i stałych w obrębie programu i segmentów.

W podręczniku opisano właściwości Ady częściowo lub w ogóle nie implementowane w kompilatorze. Janus/ Ada nie dopuszcza wielozadaniowości, a wyjątki i jednostki rodzajowe są zaimplementowane częściowo. Implementacja jednostek rodzajowych jest ograniczona do najprostszych wypadków. Częściowe udostępnienie wielu właściwości dotyczy różnych obszarów Janus/ Ady. Na przykład, nie są dostępne ustalone typy zmiennoprzecinkowe, a typy wskaźnikowe mogą wskazywać tylko na typy proste. Nie zaimplementowano też inicjalizowania pól rekordów, a selektory rekordów wariantowych mogą być używane tylko jako selektory pól wariantowych. Nie implementowano przemianowania pakietów i podprogramów, a sekwencyjne i bezpośrednie operacje we-wy są ograniczone do typów prostych. Ponadto, w pakiecie **text_io** nie zawarto operacji we-wy na typach wyliczeniowych (dodatkowe informacje – „Informatyka" nr 4, 5 i 6, 1987)

Te ograniczenia powstrzymują przed użyciem Janus/Ady jako głównego języka programowania. W obecnej postaci Janus/ Ada w przybliżeniu prezentuje poziom Pascala lub Moduli-2.

Testy porównawcze

Testy omawianych kompilatorów przeprowadzono na mikrokomputerze IBM PC AT (z zegarem o częstotliwości 6 MHz) pracującym pod kontrolą PC-DOS wersji

3.1, wyposażonym w koprocesor 8087, 512 KB pamięci RAM, dysk stały 20 MB, jeden napęd dyskietek o wysokiej gęstości i jeden napęd dyskietek 360 KB. Kompilator firmy Alsys testowano z zainstalowaną, dodatkową kartą pamięci. Testy obejmowały standardowy zestaw amerykańskiego miesięcznika „Byte". Test Sito obrazuje czas potrzebny na obliczenie jednej iteracji generatora liczb pierwszych według algorytmu Eratostenesa. W teście „Obliczenia" wykonuje się 10 000 mnożeń i dzieleń na liczbach zmiennoprzecinkowych. Testy „Zapis_Dysku" i „Odczyt_Dysku" polegają na zapisaniu i odczytaniu z dyskietki o wysokiej gęstości pliku sekwencyjnego o długości 64 KB. Wykonano również testy obejmujące rozpowszechnione operacje, takie jak sortowanie i odwracanie macierzy. W teście „Sortowanie_Całkowitoliczbowe" mierzy się szybkość utworzenia uporządkowanej tablicy 1000 liczb całkowitych i posortowania jej w odwrotnym porządku za pomocą metody Shella-Metznera. Test „Alokacja_Dynamiczna" mierzy czas tworzenia drzewa binarnego liczb całkowitych, którego węzły są alokowane podczas wykonania, jak również czas potrzebny na przejście wszystkich węzłów w kolejności ich wstawiania. Test „Odwracanie_Macierzy" polega na utworzeniu macierzy o wymiarach 20 × 20 i przypisaniu liczby 2 wszystkim elementom diagonalnym, a liczby 1 dowolnym pozostałym. Test „Rekurencja" wykonano wykorzystując algorytm szybkiego sortowania.

Wyniki testów porównawczych (gwiazdka oznacza czas dla A-kodu); czasy podane dla „Alokacji_ Dynamicznej" oznaczają tworzenie i przeszukiwanie

Kompilator	Plik wykonywalny (KB)	Kompilacja i konsolidacja (min:sek)	Wykonanie (min:sek)
Sito (plik źródłowy 881 B)			
Alsys	42,4	1:39	00:07
Artek	39,1	1:40	00:06
AdaVantage	25,1	1:25	00:27
Janus/Ada	35,9	0:50	00:09
Obliczenia (plik źródłowy 459 B)			
Alsys	43,1	2:14	00:03
Artek	39,8	1:42	00:03
AdaVantage	30,2	1:00	00:03
Janus/Ada	33,7	0:43	00:05
Zapis_Dysku (plik źródłowy 422 B)			
Alsys	42,2	1:33	00:13
Artek	39,3	1:36	00:48
AdaVantage	23,9	1:25	00:45
Janus/Ada	36,9	0:45	00:48
Odczyt_Dysku (plik źródłowy 315 B)			
Alsys	42,2	1:34	00:15
Artek	38,2	1:33	00:43
AdaVantage	23,3	1:23	00:23
Janus/Ada	35,7	0:43	00:13
Sortowanie (plik źródłowy 2168 B)			
Alsys	43,5	2:13	00:11
Artek	–	–	3:46(x)
AdaVantage	28,8	1:34	1:00
Janus/Ada	37,0	0,47	00:19
Alokacja_Dynamiczna (plik źródłowy 2560 B)			
Alsys	43,2	2:02	00:11/00:09
Artek	–	–	4:16(x)/2:14(x)
AdaVantage	26,6	1:32	00:39/00:28
Janus/Ada	37,2	0:47	00:33/00:23
Odwracanie Macierzy (plik źródłowy 1414 B)			
Alsys	42,2	2:24	00:03
Artek	–	–	00:07(x)
AdaVantage	31,5	1:39	00:06
Janus/Ada	34,9	0,46	00:04
Rekurencja (plik źródłowy 1792 B)			
Alsys	43,3	2:10	0:04
Artek	–	–	0:52(x)
AdaVantage	28,5	1:34	0:16
Janus/Ada	36,7	0:47	0:05

Wyniki testów przedstawione w tabeli pokazują, że AdaVantage wytwarza najmniejsze pliki wykonywalne. Wyniki kompilacji i konsolidacji faworyzują Janus/ Adę. Prawdopodobnie, Janus/ Ada może szybko wytwarzać wykonywalne pliki dlatego, że stanowi podzbiór Ady, co pozwala ominąć pewną liczbę mechanizmów kontroli. Przyszłe wersje Janus/ Ady odpowiedzą na pytanie, czy dodanie nowych właściwości nie spowolni kompilacji i konsolidacji. AdaVantage, implementacja bliska pełnej Ady, zajmuje drugie miejsce pod względem czasu kompilacji i konsolidacji. Alsys Ada w większości wypadków wytwarza pliki najszybciej wykonywane. Pod tym względem, kompilator Janus/ Ada zajmuje drugie miejsce.

Program tłumaczący firmy Artek nie obsługuje prawidłowo programów testowych zawierających procedury lokalne. Są to testy: „Odwracalnie_Macierzy", „Sortowanie_Całkowitoliczbowe", „Rekurencja" i „Alokacja_Dynamiczna". W tych wypadkach program tłumaczący nie mógł wytworzyć pliku wykonywalnego lub wytwarzał kod powodujący załamanie systemu. Z drugiej strony, pliki pośrednie A–kodu dla wszystkich testów były wykonywane dobrze, ale bardzo wolno.

. . .

Wybór jednego z kompilatorów zależy od kilku czynników: stopnia znajomości Ady, zapotrzebowania na oprogramowanie w Adzie, kosztów, łatwości użycia, wymagań sprzętowych.

Dla potencjalnych, profesjonalnych programistów Ady, którzy są w stanie przeznaczyć mikrokomputer PC/AT na stanowisko wytwarzania oprogramowania w Adzie, można polecić drogi pakiet firmy Alsys. AdaVantage, o bardziej przystępnej cenie i umiarkowanych wymaganiach sprzętowych, ale ograniczonym zakresie implementacji, jest odpowiedni dla programistów o średnich umiejętnościach.

Kompilator firmy Artek i RR Software są użyteczne dla nowicjuszy. Artek Ada ma przyjazne sprzężenie z użytkownikiem, a Janus/ Ada – dokumentację odpowiednią dla początkujących. Jednakże, jeśli ceny tych pakietów pozostaną na poziomie znacznie przewyższającym cenę AdaVantage, to na pewno decyzja ich wyboru nie będzie uzasadniona ekonomicznie.

Oprac.: **M. KUC**
na podstawie „Byte"

Mokrzycki W.: Przegląd technik grafiki komputerowej (1)
INFORMATYKA 1988, nr 9, s. 1
Pierwsza część przeglądu technik grafiki komputerowej, zawierająca omówienie postaci danych obrazowych, sposobów przekształcania obrazów i ich wprowadzania do komputera oraz urządzeń i techniki obrazowania.

Mokrzycki W.: Survey of computer graphics technology (1)
INFORMATYKA 1988, No 9, p. 1
First part of the computer graphics technology survey, which includes discussion of the image data form, the way of image conversion and its input to computer, as well as of imaging devices and technics.

Mokrzycki W.: Eine Übersicht von Techniken der Computergrafik (1)
INFORMATYKA 1988, Nr 9, S. 1
Erster Teil einer Übersicht von Computergrafiktechniken, der eine Besprechung von Bilddatengestalt, Umwandlungsverfahren von Bildern und deren Eingabe in einem Computer, sowie von Einrichtungen und Techniken der Bilddarstellung umfasst.

Kruszewska D.: System operacyjny IPIX
INFORMATYKA 1988, nr 9, s. 5
Charakterystyka systemu operacyjnego IPIX, opracowanego dla komputerów typu IBM PC w Instytucie Podstaw Informatyki PAN w Warszawie.

Kruszewska D.: IPIX operating system
INFORMATYKA 1988, No 9, p. 5
Characteristics of the IPIX operating system, which was elaborated for IMB PC and compatible computers in the Informatics Foundations Institute of Polish Academy of Sciences in Warsaw.

Kruszewska D.: IPIX-Betriebssystem
INFORMATYKA 1988, Nr 9, S. 5
Eine charakteristik von IPIX, einem neuen Betriebssystem für Computer von IBM PC-Typ, das im Institut für Informatikgrundlagen des Polnischen Akademie der Wissenschaften in Warschau erarbeitet wurde.

Turski W. M.: Kanoniczny krok procesu programowania (2)
INFORMATYKA 1988, nr 9, s. 9
Druga część charakterystyki metody kroków kanonicznych, zawierająca konkretny przykład jej użycia oraz zrealizowane zastosowania zagraniczne.

Turski W. M.: Canonical step of programing process (2)
INFORMATYKA 1988, No 9, p. 9
Second part of canonical steps method characteristics, which includes a real example of its using and realized applications abroad.

Turski W. M.: Kanonischer Schritt eines Programmierungsprozesses (2)
INFORMATYKA 1988, Nr 9, S. 9
Zweiter Teil einer Charakteristik von Methode der kanonischen Schritte, der einen konkreten Beispiel ihrer Anwendung und Information über schon realisierten Anwendungen im Ausland umfasst.

Foersterling F.: Wymagania stawiane bazom danych w biurach (2)
INFORMATYKA 1988, nr 9, s. 12
Druga część omówienia problemu baz danych w biurach, zawierająca charakterystykę architektury bazy danych w biurowym systemie informacyjnym.

Foersterling F.: Requirements towards office data bases (2)
INFORMATYKA 1988, No 9, p. 12
Second part of discussion of problems connected with data bases in offices, which includes characteristics of dase base architecture in office information system.

Foersterling F.: Anforderungen an Datenbanken in Büros (2)
INFORMATYKA 1988, Nr 9, S. 12
Zweiter Teil einer Besprechung von Problemen, die mit Datenbanken in Büros verbunden sind. Es wurde eine Charakteristik von Architektur der Datenbank in einem Büroinformationssystem angegeben.

Rybus R.: Nowoczesne programowanie w języku C
INFORMATYKA 1988, nr 9, S. 15
Charakterystyka zasad inżynierii oprogramowania oraz możliwości efektywnego stosowania tych zasad przy programowaniu w języku C.

Rybus R.: Modern programing in the C-language
INFORMATYKA 1988, No 9, p. 15
Characteristics of programing engineering rules and possibilities of effective application of these rules by programing in the C-language.

Rybus R.: Moderne Programmierung in der C--Sprache
INFORMATYKA 1988, Nr 9, S. 15
Eine Charakteristik von Grundlagen des Programmierungsingenieurwesens und von Möglichkeiten der effektiven Anwendung dieser Grundlagen bei Programmierung in der C-Sprache.

Podraza R.: Sterowanie przepływowe (1). Opis modelu
INFORMATYKA 1988, nr 9, s. 19
Pierwsza część charakterystyki zasad funkcjonowania systemów sterowania przepływowego, zawierająca omówienie modelu takiego sterowania.

Podraza R.: Dataflow control (1). The model description
INFORMATYKA 1988, No 9, p. 19
First part of characteristics of data flow control system performing rules, which includes discussion of the control model.

Podraza R.: Durchflusssteuerung (1). Eine Beschreibung des Modells
INFORMATYKA 1988, Nr 9, S. 19
Erster Teil einer Charakteristik von Funktionierungsgrundsätzen der Durchflusssteuerungsystemen, der eine Besprechung des Modells solcher Steuerung umfasst.

Franczyk J.: Komputerowe wspomaganie projektowania układów cyfrowych w języku Modlan
INFORMATYKA 1988, nr 9, s. 23
Charakterystyka języka do projektowania układów cyfrowych Modlan, opracowanego w Instytucie Elektroniki Politechniki Śląskiej.

Franczyk J.: Computer assisted design of digital circuits in the Modlan language
INFORMATYKA 1988, No 9, p. 23
Characteritcs of the Modlan language for designing of digital circuits, which was elaborated in the Electronics Institute of Silesia Technical University.

Franczyk J.: Computerunterstützte Projektierung von digitalen Schaltungen in der Modlan--Sprache
INFORMATYKA 1988, Nr 9, S. 23
Eine Charakteristik der Modlan-Sprache für Projektierung der digitalen Schaltungen, die im instituf für Elektronik der Schlesischen Technischen Universität erarbeitet wurde.

Zalewski J.: Kermit – organizacja, implementacja i zastosowanie
INFORMATYKA 1988, nr 9, s. 27
Charakterystyka i przykłady użycia protokołu komunikacyjnego do przesyłania plików Kermit oraz jego implementacji na mikrokomputery IBM PC

Zalewski J.: Kermit – organization, implementation and application
INFORMATYKA 1988, No 9, p. 27
Characteristics and application examples of the Kermit, communication protocol for files transmitting and its implementation on IBM PC microcomputers.

Zalewski J.: Kermit – Organisation, Implementierung und Anwendung
INFORMATYKA 1988, Nr 9, S. 27
Eine Charakteristik und Anwendungsbeispiele des Kermits, eines Kommunikationsprotokolls für Dateiübermittlung, sowie seiner Implementierung auf IMB PC Mikrocomputern.

Kto jest kim w IFIP

Aage Melbye

Aage Melbye, wiceprzewodniczący IFIP, urodził się w 1933 r. w Kopenhadze. Tytuł magistra ekonomii i badań operacyjnych uzyskał na Uniwersytecie Kopenhaskim w 1957 r. Karierę zawodową rozpoczął w Regnecentralen (Duński Instytut Sprzętu Komputerowego) pod kierunkiem Nielsa Becha, pierwszego przedstawiciela Danii w Zgromadzeniu Ogólnym IFIP. A. Melbye był związany z pionierskimi zastosowaniami komputerów w sektorze publicznym i prywatnym. Rok akademicki 1964–1965 spędził na Uniwersytecie Carnegie-Mellon jako wizytujący pracownik naukowy w dziedzinie systemów informacyjnych.

W 1971 r. przeszedł do Sparekassernes Datacenter – centrum danych duńskich banków oszczędnościowych, zatrudniającym 500 osób, i został jego dyrektorem generalnym w 1973 r. Zrealizował pierwszy w duńskiej bankowości duży system interakcyjny. Był też odpowiedzialny za stworzenie i implementację typowo rozproszonego systemu o 6000 stanowisk roboczych dla całej sieci duńskich banków oszczędnościowych.

A. Melbye został w 1984 r. niezależnym konsultantem specjalizującym się w menedżerskich i organizacyjnych aspektach informatyki. Jest autorem wielu artykułów o zastosowaniach komputerów. Ponadto, pracował jako nauczyciel i egzaminator na kilku uniwersytetach duńskich.

A. Melbye rozpoczął oficjalną współpracę z IFIP w 1978 r. jako przedstawiciel duńskiej organizacji członkowskiej DANFIP (Duńska Federacja Przetwarzania Informacji), choć przedtem uczestniczył już w pierwszych czterech kongresach IFIP. Obecnie działa w IFIP jako przewodniczący Komitetu Polecającego, Komitetu Finansowego i zastępca przewodniczącego Komitetu Statutów i Przepisów. W 1980 r. został wybrany członkiem zarządu, a podczas Zgromadzenia Ogólnego, które odbyło się we wrześniu 1986 r. w Dublinie, wybrano go wiceprzewodniczącym IFIP.

Pan Melbye z żoną Ellen i czwórką dzieci mieszka na północ od Kopenhagi w miasteczku Holte. Znajduje przyjemność w słuchaniu muzyki kameralnej, a wśród bliskich przyjaciół w grze na wiolonczeli.

Oprac. MK na podst. IFIP Newsletter

Konferencje

Sieci komputerowe '89

Konferencja odbędzie się w dniach 27–30 czerwca 1989 roku w Politechnice Wrocławskiej. Będzie to trzecia już ze zwoływanych co dwa lata konferencji naukowych nt. projektowania, zastosowań i eksploatacji sieci komputerowych. Przewiduje się czynny udział specjalistów zagranicznych. Referaty programowe i referaty uczestników naświetlą uzyskane w ostatnich dwóch – trzech latach wyniki oraz stan krajowych i zagranicznych badań naukowych w dziedzinie sieci komputerowych, a także rozwoju różnych zastosowań sieci. Sesje tematyczne obejmować będą: metodologię projektowania, architekturę i standardy, współpracę sieci różnych typów, rozproszone bazy danych, systemy przekazywania wiadomości, podsystemy transmisji danych, eksploatację, zarządzanie i administrowanie sieciami zastosowania (transport, administracja, bankowość, ubezpieczenia, systemy informacyjne).

Szczegółowe informacje: dr Jacek Gruber, SIECI KOMPUTEROWE '89, Komitet Organizacyjny, Centrum Obliczeniowe, Politechnika Wrocławska, Wybrzeże Wyspiańskiego 27, 50–370 Wrocław, Tel. 20-33-40, teleks 742254 pwr pl

Kongres IFIP'89

W dniach od 28 sierpnia do 1 września 1989 r. odbędzie się w San Francisco (USA) kolejny cykliczny **XI Światowy Kongres Międzynarodowej Federacji Przetwarzania Informacji (IFIP)**. Hasłem tej niewątpliwie największej i najważniejszej międzynarodowej konferencji informatycznej będzie: „Lepsze narzędzia dla profesjonalistów" („Better Tools for Professionals"), stanowiące syntetyczny wyraz najbardziej odczuwalnych potrzeb środowiska w obecnej fazie rozwoju informatyki.

Obradom Kongresu tradycyjnie towarzyszyć będzie w dniach 29–31 sierpnia wielka wystawa, prezentująca aktualną ofertę wszystkich branż światowego przemysłu informatycznego. Natomiast w dniach 26 i 27 sierpnia, a więc przed oficjalnym otwarciem, zorganizowanych będzie sześć jednodniowych seminariów na najbardziej aktualne tematy współczesnej informatyki. Wzorem wielu poprzednich Kongresów, dla uczestników IFIP'89 przewidziano w programie organizowanie wycieczek stwarzających możliwość poznania interesujących osiągnięć i rozwiązań użytkowych w rejonie San Francisco.

Lokalizacja XI Kongresu IFIP oczywiście radykalnie ogranicza krąg ewentualnych uczestników z Polski. Dla wszystkich niewątpliwie interesujący będzie jednak ogłoszony już ramowy program, który jako produkt przemyśleń wielonarodowego Komitetu Programowego, wyznacza chyba najtrafniej najważniejsze obecne obszary i główne kierunki rozwojowe współczesnej informatyki. Nasi Czytelnicy bez trudu mogą stwierdzić, że większość z nich jest zbieżna z obecnie realizowaną tematyką INFORMATYKI.

Oficjalny program IFIP'89 zawiera następujące sformułowania głównych obszarów tematycznych Kongresu:

- Podstawowe narzędzia (Fundamental Tools),
- Języki i systemy operacyjne (Languages and Operating Systems),
- Komunikacja i systemy rozłożone (Communication and Distributed Systems),
- Systemy baz wiedzy (Knowlegde-Based Systems),
- Inżynieria oprogramowania (Software Engineering),
- Technologia i zastosowanie superkomputerów (Supercomputing),
- Narzędzia wspomaganego projektowania układów VLSI (VLSI-CAD Tools),
- Automatyzacja biur (Office Automation),
- Automatyzacja fabryk (Factory Automation),
- Kształcenie (Education),
- Komputer i społeczeństwo (Computer and Society).

Obrady w każdym z wymienionych jedenastu obszarów tematycznych będą zorganizowane w formie wyodrębnionych konferencji, obejmujących wygłoszenie zamówionych referatów programowych oraz przyjętych przez Komitet Programowy referatów nadesłanych, a także dyskusji panelowych. Spodziewana jest współpraca i udział przedstawicieli ważniejszych krajowych i międzynarodowych programów rozwoju informatyki.

O dalsze informacje i szczegółowy program Kongresu należy zwracać się pod adres:
11th World Computer Congress, P.O. Box 18-P, Denver, Colorado 80218, USA.

Warunki prenumeraty na lata 1988–1989

Prenumeratorzy zbiorowi – jednostki gospodarki uspołecznionej, instytucje i organizacje społeczne zamawiają prenumeratę dokonując wpłat wyłącznie na blankiecie „wpłata-zamówienie" (jest to „polecenie przelewu" rozszerzone dla potrzeb Wydawnictwa o część dotyczącą zamówienia).

Blankiety te będą dostarczane dotychczasowym prenumeratom przez Zakład Kolportażu. Nowi prenumeratorzy otrzymają je po zgłoszeniu zapotrzebowania (pisemne lub telefoniczne) w Zakładzie Kolportażu.

Prenumeratorzy indywidualni – osoby fizyczne zamawiają prenumeratę dokonując wpłaty w UPT lub NBP na blankiecie NBP. Na odwrocie wszystkich odcinków blankietu należy wpisać tytuł czasopisma, okres prenumeraty, liczbę zamawianych egzemplarzy oraz wartość wpłaty. Wpłacać należy na konto: NBP III Oddział Warszawa 1036-7490-139-11.

Prenumerata ulgowa – przysługuje wyłącznie osobom fizycznym – członkom SNT, studentom i uczniom szkół zawodowych. Warunkiem prenumeraty ulgowej jest poświadczenie blankietu wpłaty (przed jej dokonaniem) na wszystkich odcinkach pieczęcią Koła SNT, wyższej uczelni lub szkoły. Sposób zamawiania prenumeraty ulgowej jest taki sam jak prenumeraty indywidualnej. W prenumeracie ulgowej można zamówić tylko po jednym egzemplarzu każdego czasopisma.

Uwaga! Miesięcznik „Aura" może być zamawiany w prenumeracie ulgowej również przez uczniów szkół ogólnokształcących.

Prenumerata ze zleceniem wysyłki za granicę – zamawia się tak jak prenumeratę indywidualną. Dodatkowo należy podać na blankiecie wpłaty nazwisko i dokładny adres odbiorcy.

Cena prenumeraty ze zleceniem wysyłki za granicę jest dwukrotnie wyższa.

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 w 1988 roku

(dotyczy numerów 7–12)

CENA		Prenumerata normalna		Prenumerata ulgowa (bez zmiany)	
normalna	ulgowa (bez zmiany)	kwartalna	półroczna	kwartalna	półroczna
250 zł	50 zł	750 zł	1500 zł	150 zł	300 zł

Cena w 1989 roku

miesięczna		kwartalna		półroczna		roczna	
normalna	ulgowa	normalna	ulgowa	normalna	ulgowa	normalna	ulgowa
300 zł	60 zł	900 zł	180 zł	1800 zł	360 zł	3600 zł	720 zł