11 1983

informatyka

ADA w testach

Języki czasu rzeczywistego

SART

Jeszcze mikroprocesory 16-bitowe

WYDAWNICTWO SIGMA

mikroKLAN

Mikroinformatyka nie ma wrogów w naszym kraju — oficjalnie. Z tej prostej przyczyny, że jej oficjalnie nie ma. Pewnie za kilka lat ktoś spostrzeże, jak bardzo nam jej brakuje i że tylko z tego powodu ciągle nie możemy wyjść z kryzysu. Gazety będą biadolić, Federacja Konsumentów domagać się, a Dziennik Telewizyjny zajmie się demaskowaniem. I wtedy powstanie... kolejny „program operacyjny", który po ciężkich bólach porodowych objawi się „PRLX-em-91" na licencji SINCLAIRA.

Rozwój mikroinformatyki w naszym kraju może jednak pobiec innym torem. Zaczęły już powstawać pierwsze kluby mikrokomputerowe, zrzeszające entuzjastów gier edukacyjnych. Na razie nieliczne, bo też ceny zagranicznego sprzętu wymagają niebagatelnego kapitału. Ale w sukurs przychodzą hobbyści. Powstają już tu i ówdzie mikrokomputery „domowego wyrobu". Są rozwiązania lepsze i gorsze, tańsze i droższe. Wadą wszystkich jest brak rozpowszechniania. Najważniejsze jednak, że są — a szkoda, by zainwestowany wysiłek i inwencja poszły na marne. Nie będziemy więc czekać, aż oślepi nas „zielone światło" dla mikroinformatyki.

Na łamach „mikroKLANU", od stycznia 1984 — comiesięcznej wkładki INFORMATYKI, będziemy opisywać użyteczne dla zapaleńców:

— ciekawe rozwiązania konstrukcyjne
— użyteczne procedury programowe
— zasady eksploatacji i dodatkowe możliwości mikrokomputerów
— niezbyt jasne właściwości popularnych układów
— sytuację „rynkową" — co, gdzie, za ile.

We wkładce publikować będziemy teksty z w i ę ź l e opisujące pomysł aplikacji lub procedury programowej (do 4 stron znormalizowanego maszynopisu: 30 wierszy po 60 znaków). Rysunki nie muszą być wykonywane na kalce; ważne natomiast, aby były całkowicie czytelne i ułatwiały zastosowanie pomysłu przez innych. Materiały powinny być nadsyłane w dwóch egzemplarzach, z załączeniem aktualnych adresów, a także numerów telefonów, umożliwiających szybki kontakt.

REDAKTOR

WYDAWNICTWO NACZELNA ORGANIZACJA TECHNICZNA
SIGMA
CZASOPISM I KSIĄŻEK TECHNICZNYCH
ul. Świętokrzyska 14a
00-950 Warszawa
skrytka pocztowa 1004

Redakcja: 00-041 Warszawa, ul. Jasna 14/16, pok. 243 i 244, tel. 27-71-40, dyżury redakcji 10.00—12.00

Zakł. Graf. „Tamka". Zam. 2287. Obj. 5,0 ark. druk. Nakład 3550 egz. M-106.

Cena egzemplarza zł 75.— INDEKS 36124 Prenumerata roczna zł 900.—

Informatyka

zastosowania w gospodarce, technice i nauce

Nr 11
MIESIĘCZNIK
1983
ROK XVIII
Listopad

ORGAN KOMITETU INFORMATYKI MINISTERSTWA NAUKI, SZKOLNICTWA WYŻSZEGO I TECHNIKI ORAZ KOMITETU NAUKOWO-TECHNICZNEGO NOT DS. INFORMATYKI

W NUMERZE:

SŁAWOMIR BŁASZCZAK
WACŁAW ISZKOWSKI
Instytut Informatyki
Politechnika Warszawska

Język ADA w testach

W kilku ubiegłorocznych numerach INFORMATYKI [5] przedstawiono nieformalny opis języka ADA oparty na podręczniku [3]. Obecnie przedstawiamy ćwiczenia umożliwiające samodzielne sprawdzenie zrozumienia poszczególnych konstrukcji języka. Forma ćwiczeń jest wzorowana na Self-Assesment Procedure, zamieszczonej w czasopiśmie Communications of the ACM [4]. Ich zadaniem jest też wskazanie na takie szczegóły konstrukcji języka, które nie występują w innych językach programowania lub mają inną semantykę. Oczywiście, podany zestaw ćwiczeń nie obejmuje wszystkich elementów języka.

Każde ćwiczenie składa się z dwóch części. Pierwsza zawiera formalnie poprawny fragment programu napisany w języku ADA. W części drugiej podano cztery lub pięć stwierdzeń odnoszących się do danego programu. Zadaniem rozwiązującego jest określenie, które spośród nich są nieprawdziwe. Odpowiedzi i krótkie wyjaśnienia umieszczono na końcu artykułu.

Ponieważ polska terminologia dla języka ADA nie została jeszcze ustalona, w ćwiczeniach przyjęto terminologię z poprzednich artykułów [2, 5], odwołując się — gdzie to było konieczne — do oryginalnego terminu angielskiego.

ĆWICZENIE 1

```
function SUMA (N: INTEGER) return INTEGER is
   S: INTEGER := 0;
begin
   for I in 1..N loop
      S := S+I;
   end loop;
   return S;
end SUMA;
```

A. Po wywołaniu funkcji SUMA (5), zmienna S przyjmie kolejno wartość: 0, 1, 3, 6, 10, 15.

B. Wartość funkcji po wywołaniu SUMA (k), gdzie k ma wartość dodatnią, jest równa sumie k liczb.

C. Jeżeli w treści funkcji zmienimy identyfikator I na Z, to działanie funkcji nie zmieni się.

D. Jeżeli w tekście programu zmienimy identyfikator N na K w obu miejscach występowania, to z punktu widzenia wywołania funkcji nic się nie zmieni.

E. Zadeklarowanie zmiennej I, analogicznie jak zmiennej S, nie zmieni poprawności przykładu.

ĆWICZENIE 2

```
function MIN (V: in VECTOR) return INTEGER is
   INDEX : INTEGER := V'FIRST;
begin
   for I in V'RANGE loop
      if V(I) < V(INDEX) then INDEX := I;
      end if;
   end loop;
   return INDEX;
end MIN;
```

A. Zmienna INDEX jest inicjowana wartością pierwszej składowej V.

B. Funkcja określa wartość minimalną w wektorze podanym jako argument (parametr aktualny) w jej wywołaniu.

C. Typ VECTOR musi być zdefiniowany w otoczeniu funkcji tylko jako:

```
type VECTOR is array (INTEGER range <> ) of
INTEGER;
```

D. Zmienna I przyjmie kolejno wartości od V'FIRST do V'LAST.

E. W deklaracji typu parametru V można usunąć zwrot in określający tryb pobrania dla przekazywania argumentu, co nie zmieni poprawności przykładu.

ĆWICZENIE 3

```
procedure SWAP (X, Y: in out INTEGER) is
   TEMP: constant INTEGER := X;
begin
   X := Y;
   Y := TEMP;
end SWAP;
```

A. Z deklaracji zmiennej TEMP można usunąć słowo constant.

B. Po wywołaniu SWAP (I, A (I)), gdzie:

```
I : INTEGER := 5; -- i
A : array (1..10) of INTEGER := (2, 4, 3, 7, 6, 8, 9, 2, 1, 0);
```

otrzymamy I = 6 i A (6) = 5.

C. Dla wymiany wartości typu FLOAT należy tylko zmienić:

(X, Y: in out FLOAT) i TEMP: constant FLOAT := X;

D. Dla deklaracji:

X: constant INTEGER: = 5; Y: INTEGER := 6;

i wywołania SWAP (X, Y) — zmienna Y przybierze wartość 5.

ĆWICZENIE 4

```
procedure BSORT (V: in out VECTOR) is
  K: INTEGER;
begin
for I in V'RANGE loop
   K := MIN (V(I..V'LAST)); -- funkcja MIN z przykładu 2
   SWAP (V(I), V(K)); -- procedura z przykładu 3
end loop;
end BSORT;
```

A. Po wykonaniu instrukcji BSORT(V), gdzie V jest typu VECTOR, wektor będzie uporządkowany niemalejąco.

B. Zmienną lokalną K można usunąć, wstawiając funkcję MIN jako indeks w wywołaniu procedury:

```
SWAP(V(I), V(MIN(I..V'LAST)));
```

C. Jeżeli składowe tablicy typu VECTOR są dowolnego typu (dla którego jest określona operacja „<"), to należy zmienić treści procedur BSORT, MIN i SWAP.

D. Dla kolejnych wykonań instrukcji:

```
K := MIN (V(I..V'LAST));
```

tablice będące argumentem funkcji MIN mają różny wymiar.

ĆWICZENIE 5

```
declare
   type DAY is (MON, TUE, WED, THU, FRI, SAT. SUN);
   type FREE is new DAY range SAT..SUN;
   D:  DAY : = SAT;
   F:  FREE;
   B: BOOLEAN;
begin
   F : = SUN;
   B : = F < FREE (D);
   D : = DAY (F);
end;
```

A. Po wykonaniu drugiej instrukcji bloku, zmienna B przyjmie wartość FALSE.
B. Relacja FREE'POS (FREE'FIRST) = DAY'POS (DAY' FIRST) jest prawdziwa.
C. Operator „<" w drugiej instrukcji jest predefiniowany dla argumentów typu DAY i dziedziczony przez typ FREE.
D. Klauzula DAY(F) zmienia argument typu FREE na wartość typu DAY.

ĆWICZENIE 6

```
procedure LIST is
   type TYPELEM is (HD, ELM);
   type ELEMENT;
   type LINK is access ELEMENT;
   type ELEMENT (TE: TYPELEM: = ELM; SIZE: INTE-
                                    GER range 1..20 : = 10)
     is record
       case TE is
       when HD = >
          NAME : STRING (1..SIZE);
       FIRST, LAST :  LINK : = null;
       when ELM = >
          DATA : array (1..SIZE) of INTEGER;
       PRED, SUCC : LINK : = null;
       end case;
     end record;
Q: LINK : = new ELEMENT (HD, 4, "KOL1");
E1, E2: LINK;
begin
   E1 : = new ELEMENT (ELM, 3, (1, 2, 3), Q);
   Q. FIRST : = E1; Q. LAST : = E1;
   E2 : = new ELEMENT (ELM, 2, (3, 4), E1, Q);
   Q. LAST : = E2; E1. SUCC : = E1;
end LIST;
```

A. Procedura LIST utworzy listę dwuelementową.
B. Wyróżnik SIZE określa, w zależności od wartości wyróżnika TE, długość nazwy listy lub wielkość tablicy DATA.
C. Instrukcja E1 : = new ELEMENT; jest poprawna.
D. Instrukcja E1 : = E2 zmieni zawartość pierwszego obiektu typu ELEMENT.
E. Jeśli pominiemy wartość początkową dla wyróżnika SIZE, to instrukcja z punktu C będzie poprawna.

ĆWICZENIE 7

```
package P is
   CONST : constant INTEGER : = 1000;
   TAB : array (1..CONST) of INTEGER;
   procedure D;
end P;
package body P is
V : INTEGER : = 0;
...
procedure Q is
   begin ... end Q;
procedure D is
   V1 : INTEGER : = 0;
   begin
      V : = V+1; V1 : = V1+1;
      ...
      for I in 1..CONST loop
         TAB (I) : = I;
      end lopp;
      Q;
   end D;
begin D; end P;
```

A. Poza pakietem P można wywołać procedurę Q przez zwrot P.Q.
B. Stała CONST jest widoczna w treści pakietu P bez definiowania jej w tej części.
C. Po każdym kolejnym wywołaniu procedury D wartość zmiennej V zwiększy się o 1.
D. Również zmienna V1 po każdym kolejnym wywołaniu procedury D zwiększy się o 1.
E. Wartości składowych tablicy TAB mogą być zmieniane poza pakietem P.

ĆWICZENIE 8

```
package COMPLEX_NUMBER is
   type COMPLEX is private;
   function EQ (X, Y: COMPLEX) return BOOLEAN;
   function „+" (X, Y: COMPLEX) return COMPLEX;
   function „—" (X, Y: COMPLEX) return COMPLEX;
   funktion „*" (X, Y: COMPLEX) return COMPLEX;
   function CMPLX (X, Y: FLOAT) return COMPLEX;
   function RE (X: COMPLEX) return FLOAT;
   function IX (X: COMPLEX) return FLOAT;
   private
      type COMPLEX is
      record
         REAL, IMAGE: FLOAT;
      end record;
end COMPLEX_NUMBER;
```

```
with COMPLEX_NUMBER; -- treść poza pakietem
   procedure USE_COMPLEX is
      use COMPLEX_NUMBER;
   C1, C2 : COMPLEX;
   R : FLOAT;
begin
   C1 : = CMPLX (1.0,2.0)
   C2 : = CMPLX (2.0,2.0);
   C1 : = C1+C2;
   if C1 = C2 then R : = RE (C1) end if;
end USE_COMPLEX;
```

A. W procedurze USE_COMPLEX można napisać R : = C1. REAL.
B. Operator "=" jest z założenia zdefiniowany dla typu COMPLEX.
C. Można zdefiniować funkcje operatora równości:
function „=" (X, Y: COMPLEX) return BOOLEAN renames EQ;
D. Powyższa funkcje można by zdefiniować, gdyby typ COMPLEX był limited private.
E. Operator "+" z instrukcji C1 : = C1+C2 został zdefiniowany w pakiecie COMPLEX_NUMBER.

ĆWICZENIE 9

```
B: declare
   task type RESOURCE is
      entry REQUEST;
      entry RELEASE;
      end  RESOURCE;
   task body RESOURCE is
      begin
         loop
            accept REQUEST;
            accept RELEASE;
      end loop;
      end  RESOURCE;
R1, R2 : RESOURCE;
AR : array (1..10) of RESOURCE;
type PTR is access RESOURCE;
PR : PTR : = new RESOURCE;
...
begin ... end B;
```

A. Zadania R1, R2 i AR(i) (i : 1..10) są uaktywnione w dowolnej kolejności.
B. Po zakończeniu wykonywania instrukcji bloku B wszystkie zadeklarowane w tym bloku zadania kończą swoje działanie.

C. Obiekty typu PTR można porównywać między sobą pod względem równości lub nierówności.
D. Instrukcja R1 : = R2; jest dozwolona.
E. Zgłoszenia do wejść AR(1).REQUEST i AR(2).REQUEST z różnych zadań mogą wykonywać się współbieżnie.

ĆWICZENIE 10

```
task Z1 is
   entry E1 (...);
   entry E2 (...);
end Z1;
task body Z1 is
   B1, B2 : BOOLEAN;
      I : DURATION;
   begin
   loop
      select
         when B1 = > accept E1 ... do I1; end E1;
      or
         when B2 = > delay I; I2;
      or
         delay 5.0; I3;
      or
         accept E2 (...) do I4; end E2;
         end select;
   end loop;
end Z1;
```

A. Jeśli B1 = B2 = TRUE, I = 2.0 i istnieje zgłoszenie do wejścia E2, to zostanie wykonana instrukcja I4.
B. Jeśli B2 = TRUE, B1 = FALSE oraz I = 0.0 i istnieje zgłoszenie do wejścia E2, to nie można określić, czy zostanie wykonana instrukcja I2 czy I4.
C. Jeśli B1 = FALSE, B2 = TRUE, I = 2.0 i nie ma zgłoszenia do żadnego z wejść przez co najmniej 5 sekund od chwili rozpoczęcia wykonania instrukcji select, to zostanie wykonana instrukcja I2.
D. W przedstawionej instrukcji select, w każdym przypadku zostanie wykonana jedna z instrukcji I1, I2, I3, I4.

ĆWICZENIE 11

```
task type SEM is
   entry P;
   entry V;
end SEM;
task body SEM is
   VAL: INTEGER : = 0;
begin
loop
   select
      when VAL > 0 = >
         accept P;
         VAL : = VAL-1;
      or
         accept V;
         VAL : = VAL+1;
      end select;
end loop;
end SEM;
S1, S2 : SEM;
```

A. Semafory S1, S2 mają wartość początkową równą 0.
B. Jeśli do wejść P i V zgłoszenia będą równoczesne i jeśli VAL > 0, to nie można określić, które ze zgłoszeń zostanie przyjęte jako pierwsze.
C. Zadanie, które zgłosiło się do wejścia P jest zawieszone na czas wykonania instrukcji VAL : = VAL-1.
D. Zadania typu SEM mogą zakończyć swoją pracę tylko na skutek wydania w innym zadaniu instrukcji **abort**.

ĆWICZENIE 12

```
procedure P is
   ERROR : exception;
   procedure R is
      -- część deklarująca R
   begin
      ...
      raise ERROR;
      ...
   end R;
   procedure Q is
      begin
         IQ1; R; IQ2;
         exception
            when ERROR = > IQ3;
            when others = > IQ4;
      end Q;
begin -- P
   IP1; Q; IP2;
exception
   when ERROR = > IP3;
   when others = > IP4;
end P;
```

A. Jeżeli protest (exception) ERROR zostanie zgłoszony podczas wykonywania procedury R wywołanej z procedury Q, to zostanie wykonany ciąg instrukcji IQ3.
B. Protest zgłoszony podczas procesu opracowania (elaboration) deklaracji procedury R (np .CONSTRAINT_ERROR) będzie obsłużony w procedurze Q przez instrukcje IQ4.
C. Jeśli w procedurze Q zadeklarujemy protest o nazwie ERROR i protest o takiej nazwie zostanie zgłoszony podczas wykonywania treści procedury R wywołanej przez Q, to zostanie wykonany ciąg instrukcji IQ4.
D. Każdy protest zgłoszony w treści procedury P zostanie w niej obsłużony (w niej samej lub w obiektach w niej zawartych).

ĆWICZENIE 13

```
procedure P is
   ERROR : exception;
task Z1 is
   entry E1;
end Z1;
task Z2 is
   end Z2;
task body Z1 is
   ...
begin
   ...
   accept E1 do IZ11 end E1;
   ...
exception
   when ERROR = > IZ12;
end Z1;
task body Z2 is
begin
   loop
   ...
   Z1.E1;
   ...
   end loop;
end Z2;
begin -- P
   ...
   exception
      when ERROR = > IP1;
      when others = > IP2;
end P;
```

A. Zgłoszenie protestu ERROR w ciągu instrukcji IZ11 spowoduje wykonanie ciągu instrukcji IZ12 oraz IP1.
B. Jeśli zadanie Z1 zostanie zakończone anormalnie (np. na skutek instrukcji **abort**) w czasie przebiegu spotkania (w ciągu instrukcji IZ11) i jeśli zadanie Z2 uczestniczyło w tym spotkaniu, to zostanie ono również zakończone.
C. Zgłoszenie protestu podczas procesu opracowywania deklaracji zadania Z2 spowoduje wykonanie ciągu instrukcj IP2.
D. Każdy protest zgłoszony w trakcie opracowywania lub wykonania procedury P oraz zawartych w niej zadań Z1 i Z2 zostanie obsłużony w niej lub w tych zadaniach.

ĆWICZENIE 14

```
generic
   type POINT is private;
   with function DISTANCE (X, Y: POINT) return
                                        FLOAT is <>;
function HERON (A, B, C: POINT) return FLOAT;
   -- treść procedury HERON
...
declare
```

ustawi wskazany znacznik zdarzenia w stan WŁĄCZONY. Jeżeli znacznik zdarzenia był już WŁĄCZONY, to żadne działanie nie nastąpi. Z powodu wcześniejszego wywołania w jakimś zadaniu procedury żądającej zmiany stanu, stan WŁĄCZONY może spowodować przejście jakiegoś zadania ze stanu OCZEKUJACE lub ZAWIESZONE do stanu WYKONYWANE. Dotyczy wszystkich zadań czekających na e.

Wykonanie wywołania podprogramu CLEAR o postaci:

CALL CLEAR (e, m)

spowoduje, że wskazany znacznik zdarzenia zostanie WYŁĄCZONY. Jeżeli znacznik zdarzenia był już WYŁĄCZONY, nie będzie żadnego skutku wywołania.

Wykonanie wywołania funkcji TESTEM(e, m) spowoduje zwrócenie wartości logicznej PRAWDA, jeżeli wskazany znacznik zdarzenia był WŁĄCZONY, zaś wartości logicznej FAŁSZ, jeżeli był WYŁĄCZONY. Jeżeli znacznik zdarzenia jest nieznany procesorowi, to zostanie zwrócona wartość FAŁSZ i parametr błędu wskaże stan błędny.

Wykonanie wywołania podprogramu MKEM o postaci:

CALL MKEM (e, m)

nie zmienia stanu wskazanego znacznika zdarzenia, ale powoduje zamaskowanie go. Skutek zamaskowania jest taki, że znacznik zdarzenia może dowolnie zmieniać swój stan bez żadnego wpływu na zadania, które ewentualnie są oczekujące lub zawieszone, czekając aż dany znacznik zdarzenia zostanie ustawiony.

Wykonanie wywołania podprogramu UNMKEM o postaci:

CALL UNMKEM (e, m)

umożliwi wykonanie czynności związanych ze wskazanym znacznikiem zdarzenia (odmaskowanie go). Jeżeli znacznik zdarzenia jest w stanie WŁĄCZONY, wszystkie działania związane z tym znacznikiem zostaną wykonane.

OPEROWANIE ZNACZNIKAMI ZASOBÓW

Wykonanie wywołania podprogramu LOCK o postaci:

CALL LOCK (r, m)

powoduje, że wskazany znacznik zasobu (określony przez wyrażenie całkowite r) zostanie zamknięty. Jeżeli dany znacznik zasobu był już zamknięty, system nadzorczy zawiesi wykonywanie zadania wywołującego.

Wykonanie wywołania podprogramu UNLOCK o postaci:
CALL UNLOCK (r, m)
powoduje podjęcie jednego z następujących działań:
- jeżeli znacznik zasobu jest otwarty, to nie będzie żadnego skutku
- jeżeli znacznik zasobu jest zamknięty i nie ma zadań zawieszonych wskutek wcześniejszej próby zamknięcia związanego z nim zasobu, to znacznik zasobu zostanie otwarty
- jeżeli znacznik zasobu jest zamknięty i istnieje jedno lub więcej zadań zawieszonych wskutek wcześniejszych prób zamknięcia związanego z nim zasobu, to jedno i tylko jedno zadanie przejdzie do stanu WYKONYWANE, a znacznik związany z zasobem pozostaje zamknięty (kryteria stosowane do wyboru zadania mającego przejść do stanu WYKONYWANE są zależne od procesora).

Wykonanie funkcji TLOCK (r, m) powoduje zbadanie stanu wskazanego znacznika zasobu. Funkcja przekaże wartość PRAWDA, jeżeli znacznik zasobu jest otwarty lub wartość FAŁSZ, jeżeli znacznik zasobu jest zamknięty, a następnie znacznik zasobu zostanie zamknięty. Celem wykorzystania funkcji TLOCK jest umożliwienie zadaniu zarezerwowania zasobu, jeżeli jest otwarty, albo kontynuowania wykonania, jeżeli jest zamknięty, co nie byłoby możliwe za pomocą wywołania CALL LOCK.

OPERACJE SEMAFOROWE

Wszystkie zmienne semaforowe mają postać wewnętrznych zmiennych systemu nadzorczego i jedyny sposób dostępu do nich jest możliwy przez argument r, który odnosi się do jednego, określonego semafora. Wartość tak zidentyfikowanej zmiennej danego semafora jest oznaczona przez s.

Wynikiem wywołania podprogramów SIGNAL i WAITS jest — odpowiednio — zwiększenie i zmniejszenie wartości semafora o wielkość j, dodatnią liczbę całkowitą, podawaną jako drugi argument wywołań. Dla WAITS zmniejszenie wartości zachodzi tylko wtedy, gdy wynik jest nie mniejszy od zera. W przeciwnym przypadku zadanie wywołujące jest zawieszane, zanim nastąpi zmniejszenie — i kontynuacja nie nastąpi do chwili, gdy $s \geqslant j$, tj. wynik zmniejszenia będzie nieujemny.

Poniżej opisano szczegółowo wywołania podprogramów dla realizacji mechanizmów synchronizacji przy użyciu semaforów.

Inicjowanie semafora

Wykonanie wywołania podprogramu PRESEM ma dwa cele:
— deklaruje zamiar użycia określonego semafora, umożliwiając systemowi wydanie ostrzeżenia diagnostycznego w czasie wykonania; jeżeli wywołano jakąś inną operację dotyczącą semafora, który nie jest zainicjowany
— ustanawia dla semafora wartość początkową.

Normalnie wywołuje się PRESEM tylko w fazie inicjowania wykonania programu czasu rzeczywistego. Postać wywołania jest następująca:

CALL PRESEM (r, s, m)

gdzie: r — wyrażenie całkowite określające semafor, s — wartość początkowa nadawana semaforowi.

Dopóki dla określonego semafora nie wykona się wywołania podprogramu PRESEM i jego wewnętrzna zmienna nie przybierze wartości s, wewnętrzna wartość jest niezdefiniowana i inne wywołanie systemowe odnoszące się do tego semafora spowoduje powrót z błędem. Parametr s jest wyrażeniem całkowitym. Jest to jedyny sposób, aby semafor przyjął wartość ujemną. Skutek początkowej wartości ujemnej jest taki, że wymaga się odpowiednio większego przyrostu wartości przy użyciu wywołania CAL SIGNAL, zanim może nastąpić zwolnienie semafora.

Parametr m sygnalizuje stan błędu jeżeli nie istnieje semafor o podanym oznaczeniu.

Oczekiwanie pod semaforem

Wykonanie wywołania podprogramu WAITS pociągnie możliwe zawieszenie wywołującego zadania, kontrolowane przez wskazany semafor.

Przy końcu wywołana wartość semafora s zostanie zmniejszona o wielkość j. Zmniejszenie wartości i następująca po nim kontynuacja zadania — wystąpią tylko wtedy, gdy operacja nada zmiennej s wartość nieujemną. W przeciwnym przypadku zadanie wywołujące jest zawieszane do chwili, gdy może nastąpić zmniejszenie wartości.

Wywołanie podprogramu ma postać:

CALL WAITS (r, j, m)

gdzie r i m — jak powyżej, j — wyrażenie całkowite, określające wielkość, o którą zmienna semafora ma być zmniejszona; wartość j musi być dodatnia (1 lub więcej), a $j = 1$ odpowiada prostemu semaforowi, używanemu najczęściej.

Zwolnienie semafora

Wykonanie wywołania podprogramu SIGNAL powiększa wartość semaforowej zmiennej całkowitej s, którą identyfikuje się jednym z argumentów wywołania. Podprogram uzyska wyłączny dostęp do tego semafora podczas operacji i wykona modyfikację jego wartości s: = s+j
gdzie j jest argumentem wywołania. Zmiana do wystarczająco dużej wartości dodatniej spowodowana tą operacją, może umożliwić zwolnienie przejścia do stanu WYKONYWANE dla zadania oczekującego na wystąpienie tego zdarzenia w stanie ZAWIESZONE.

Inne zadania, zawieszone wskutek wykonania CALL WAITS w odniesieniu do tego samego semafora r, zostaną poddane powtórnej ocenie warunków oczekiwania po zakończeniu wywołania SIGNAL. Zapewni to zawieszonym zadaniom możliwość uwolnienia i podjęcia działania, jak opisano powyżej. Kontynuacja ta podlega wszystkim normalnym ograniczeniom dotyczącym wyłączności operacji na jednym semaforze, tzn. w danej chwili będzie badane tylko jedno z zawieszonych zadań. Badanie to może spowodować ponowne zmniejszenie wartości semafora, wskutek zwolnienia zadania z operacji WAITS. Badanie trwa tak długo, dopóki istnieje możliwość, że wartość semafora jest większa lub równa wartości j jakiegoś zawieszonego zadania. Kolejność w jakiej sprawdza się zawieszone zadania, jest poza tym zależna od procesora.

Wywołanie ma postać:

CALL SIGNAL (r, j, m)

gdzie: j jest wyrażeniem całkowitym określającym przyrost zmiennej semaforowej, jeżeli jest możliwy do wykonania. Wartość j musi być dodatnia ($j \geqslant 1$), a $j = 1$ odpowiada prostemu semaforowi, używanemu najczęściej.

Odczyt wartości semafora

Pytanie o wartość semafora wykonuje się przez wywołanie funkcji całkowitej IRDSEM (r, m). Celem użycia tej funkcji nie jest operacja synchronizacji w systemie. Jej wywołanie może, na przykład, dostarczyć informacji o tym, jak daleki od wypełnienia jest bufor lub jakiś inny współdzielony zasób. Jeżeli wywołanie zostanie przyjęte (uznane), otrzyma wyłączny dostęp do semafora. Jeżeli inne wywołanie systemowe jest właśnie w trakcie dostępu do semafora w chwili wykonania wywołania IRDSEM, to wywołanie IRDSEM będzie podlegało tym samym mechanizmom opóźnionej odpowiedzi i ubiegania się o zasoby, jak w przypadku pozostałych operacji semaforowych. Przy powrocie z wywołania identyfikator funkcji będzie miał wartość równą wewnętrznej wartości semafora w chwili przyjęcia wywołania.

NORMALNE ZAKOŃCZENIE WYKONANIA

Wykonanie wywołania CALL EXIT zakończy wykonanie zadania i spowoduje powrót zadania do stanu UŚPIONE. Stan znaczników zdarzeń i semaforów nie ulega zmianie. Znaczniki zasobów uprzednio zamknięte przez to zadanie zostaną otwarte, zaś pliki — zwolnione.

Zwykle operacje FORTRANU — STOP i END — również umożliwiają zakończenie wykonania zadania. Wtedy jednak skutki dla dołączonych jednostek, takich jak pliki, są zgodne z normą FORTRANU, a skutki dla innych zasobów opisanych w niniejszej pracy są zależne od procesora.

Tłumaczył: KAZIMIERZ MALISZEWSKI

WOJCIECH WARSKI
Instytut Sterowania i Elektroniki Przemysłowej
Politechnika Warszawska

Alternatywa dla struktury blokowej w językach czasu rzeczywistego

Dzięki znanym zaletom języków wysokiego poziomu, są one coraz częściej stosowane do zadań tradycyjnie kodowanych w językach asemblera. Wskutek ciągłego poszukiwania nowych rozwiązań, następuje szybki wzrost liczby języków specjalizowanych tej klasy. Jest to szczególnie aktualne dla języków umożliwiających programowanie systemów czasu rzeczywistego.

Możliwość programowania systemów czasu rzeczywistego jest cechą wielu różnych języków, co wynika przede wszystkim z różnorodności ich zastosowań (od budowy systemów operacyjnych do sterowania procesami przemysłowymi). Istotnym elementem różnicującym te języki są też uwarunkowania sprzętowe. Jedna ich cecha jest wszakże wspólna — umożliwiają programowanie bieżącej współpracy komputera z urządzeniami zewnętrznymi. Oznacza to, że szczególnie istotne stają się wymagania krótkiego czasu odpowiedzi komputera na sygnały z otoczenia. Jeżeli założymy, że niecelowe są wówczas jednostkowe modyfikacje sprzętu (np. stosowanie układów pomocniczych lub szybszych procesorów), to konieczne jest wprowadzenie do języka takich konstrukcji semantycznych, które można używać wydajnie przy zachowaniu jego koniecznej komunikatywności.

O spełnieniu powyższych wymagań nie decyduje żaden wyizolowany element języka, lecz jego struktura, stanowiąca przedmiot dalszych rozważań, ma tu znaczenie wyjątkowo duże. Rozważania są prowadzone z punktu widzenia realizacji na mikrokomputerach co jest szczególnie ważne — na przykład — w zastosowaniach przemysłowych, telekomunikacyjnych i wojskowych. Za reprezentatywne dla tej techniki przyjęto możliwości klasycznych procesorów 8-bitowych (np. Z80). Jest to podyktowane przede wszystkim względami ekonomicznymi oraz — specyficznym dla Polski — warunkiem dostępności. Konsekwencją tego założenia jest konieczność uwzględnienia ubogiego zestawu rejestrów i sposobów adresowania. Ograniczenia te są charakterystyczne dla wielu nowszych procesorów, tak że powyższe założenie wcale nie wydaje się zbyt silne, gdy bierze się pod uwagę wymienione zastosowania.

KONSEKWENCJE ROZWIĄZANIA KLASYCZNEGO

Za podstawę projektu każdego nowego języka programowania uznaje się tradycyjnie strukturę blokową, w sensie hierarchicznie definiowanych procedur z możliwością dostępu do obiektów nielokalnych, definiowanych na poziomach nadrzędnych. Taką strukturę zachował również język ADA [3]. Zilustrowano ją w przykładzie 1.

Przykład 1

```
procedure P1;
  var A
  procedure P2
    procedure P4
      .. P3 ..
    end
```

```
    ...
  end
  procedure P3
  var B
    ...
  end
begin
  ...
end
```

Procedura P4 ma możliwość wywołania każdej z pozostałych procedur, dostępna jej jest również zmienna **A**, ale nie zmienna **B**. Procedura P4 jest najniższa hierarchicznie, zaś przyporządkowany jej numer poziomu — największy.

Wady i zalety struktury blokowej przedstawiono w pracach [2] i [7]. W artykule [2] stwierdzono m.in., że znane zalety struktury blokowej nie kompensują, szczególnie w realizacji mikroprocesorowej, jej istotnych wad, tj.:

— nieczytelności (nawet średniej wielkości programów), spowodowanej fizycznym oddzieleniem nagłówka procedury od jej tekstu

— ograniczenia możliwości rozdzielnej kompilacji poszczególnych segmentów programu

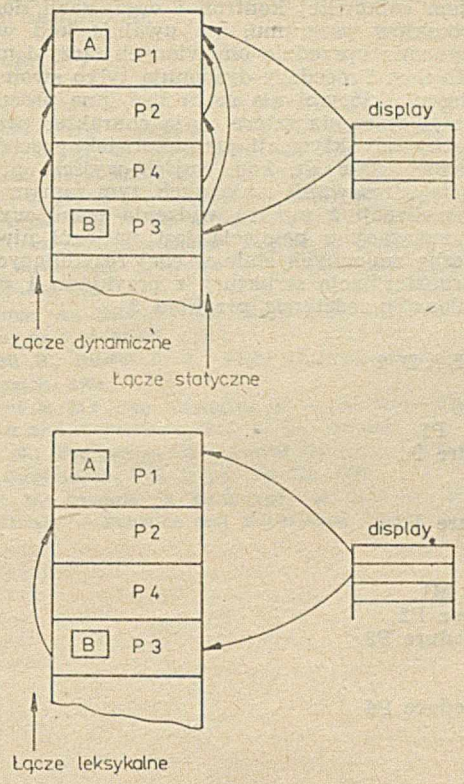
— nieefektywności implementacji procedur oraz dostępu do zmiennych nielokalnych.

Nie ulega wątpliwości, że ostatnia wada ma dla omawianych zastosowań największe znaczenie, bowiem może ograniczać w konkretnych przypadkach przydatność języka.

Koncepcja użycia struktury blokowej jest w pewnym sensie narzucona przez samą strukturę i dlatego nie ulega zmianom od czasu pierwszych kompilatorów ALGOLU [6]. Najistotniejszym problemem jest zachowanie dostępu do obiektów nielokalnych procedur; od sposobu jego rozwiązania zależy w dużej mierze efektywność systemu. Za najskuteczniejszą powszechnie uważa się tzw. metodę display. Z logicznego punktu widzenia display jest tablicą, której i-ty element jest wskaźnikiem do ostatnio utworzonego segmentu danych procedury na **i-tym** poziomie hierarchizacji. Segmenty danych (ang. activation record, data segment) są umieszczane dynamicznie według dyscypliny stosowej przy każdym wywołaniu procedury i giną z chwilą jej zakończenia. W ten sposób zajętość pamięci operacyjnej jest ograniczona tylko do tych zmiennych, które są aktualnie w użyciu. Jest to istotna zaleta struktury blokowej.

Dostęp do obiektów nielokalnych i konieczność przywracania, przy wyjściu z procedury, stanu sprzed jej wywołania — zmusza do łączenia segmentów danych. Służy temu tzw. łącze dynamiczne wraz z łączem statycznym [8] bądź łącze leksykalne [1]. Utrzymywanie tych łączy powoduje, że stały kod procedury musi być uzupełniony kodem wejścia i wyjścia z niej. Przykładowe realizacje mikrokomputerowe [8] wykazały, że ten dodatkowy kod ma znaczną długość, nawet przy wykorzystaniu stosu sprzętowego. Dodatkowym obciążeniem może być konieczność aktualizacji tablicy display, gdy statyczny poziom hierarchizacji procedury wywołanej jest niższy niż wywołującej. Konieczności tej uniknięto w metodzie Rohla [1], ale ma ona inne ograniczenia. Graficzną ilustrację łączy segmentów danych dla obu metod (po wywołaniu procedury P3 z procedury P4 w przykładzie 1) przedstawiono na rysunku.

Inny problem stanowi adresowanie zmiennych, które jest zależne od początku odpowiedniego segmentu danych. Oczywistym rozwiązaniem byłoby więc ładowanie rejestrów indeksowych adresami początkowymi aktualnie dostępnych segmentów, co umożliwiłoby dostęp do dowolnych zmiennych przez pojedyncze odwołanie się do pamięci. Niestety, mikroprocesory (podobnie jak wiele innych maszyn cyfrowych) mają z reguły nie więcej niż dwa takie rejestry (INTEL 8080 nie ma ani jednego). Stwarza to konieczność umieszczenia tablicy w pamięci, co najwyżej z zachowaniem jej wierzchołka w rejestrze. W ten sposób adresowanie zmiennych nielokalnych wymaga co najmniej dwóch odwołań do pamięci. Idea Kowaltowskiego [4], polegająca na wprowadzeniu adresowania bezwzględnego zamiast tablicy typu display, nie rozwiązuje problemu, bowiem przerzuca jedynie jego ciężar na kod wejściowo-wyjściowy procedury (konieczność przepisywania pewnych obszarów do lub ze stosu).



Łącza segmentów danych

Reasumując — struktura blokowa stwarza duże obciążenie czasowe, ponieważ wprowadza dodatkowy kod do programu i zmusza do mało efektywnego dostępu do niektórych jego obiektów. W wymienionych zastosowaniach istotna jest szybkość wykonywania programu, a nie — na przykład — minimalizacja wymaganego pola pamięci; tak więc struktura blokowa jest trudna do przyjęcia z przyczyn implementacyjnych. Zwrócę więc uwagę na inne rozwiązania, których głównym celem jest również ograniczenie widoczności obiektów programu [2, 9].

KONCEPCJA MODUŁU

Ogólniejsze mechanizmy kontroli nad propagacją identyfikatorów istnieją w takich językach, jak: CONCURRENT PASCAL, EUCLID, ALPHARD, CLU, MODULA, ADA czy LOGLAN. Najprostszym spośród tych mechanizmów jest niewątpliwie koncepcja modułu o charakterze statycznym, opracowana przez Wirtha jako uzupełnienie struktury blokowej w języku MODULA.

W ujęciu Wirtha moduł jest zbiorem stałych, zmiennych, procedur i deklaracji, zawartych między jego nagłówkiem a znacznikiem końca. Wszystkie te obiekty są lokalne dla modułu z wyjątkiem tych, których identyfikatory znajdują się na tzw. liście eksportowej. Podobnie kontrolowany jest również przepływ informacji w drugą stronę — wewnątrz modułu widoczne są tylko te obiekty spoza niego, których identyfikatory widnieją na tzw. liście importowej. Moduł stanowi więc dla propagacji identyfikatorów tamę, która otacza żądane fragmenty programu. Listy — importowa i eksportowa — są jedynymi przepustami umożliwiającymi przepływ informacji do i z modułu.

Jednoczesna obecność w języku struktury blokowej i hierarchicznie definiowanych modułów (jak w MODULI i ADZIE) wydaje się być nadmiarem. Obie konstrukcje mają ten sam, choć różnie realizowany, zasadniczy cel: ograniczenie widoczności obiektów. Przeplatanie się w programie hierarchii modułów i procedur ma też fatalny wpływ na jego czytelność.

ALTERNATYWA

Propozycją niniejszego artykułu jest struktura języka mającego hierarchizowane, statyczne moduły i liniowo definiowane procedury. Rozwiązanie takie obarcza kompilator zadaniem całkowitej kontroli i organizacji dostępu do różnych obiektów programu, ale uwalnia kod wynikowy od utrzymywania uprzednio omówionych łączy i mechanizmu typu display. Procedury dysponują tylko swoimi obiektami lokalnymi (którymi nie mogą być inne procedury) oraz obiektami otoczenia (które mają charakter permanentny, niezależny od aktywacji jakichkolwiek procedur). Ponieważ zbędny staje się kod wejścia-wyjścia procedur i upraszcza się adresowanie zmiennych, tym samym program ulega optymalizacji z punktu widzenia czasu wykonywania. Zysk związany z pamięcią jest na ogół niwelowany większą liczbą zmiennych stale w niej rezydujących. Przykładową transformacją struktury z przykładu 1 na strukturę modułową przedstawia przykład 2.

PRZYKŁAD 2

```
module MØ
   expose  P1
   procedure P1
      ...
   end
   procedure P3
      ...
   end
   module M1
      expose P2
      procedure P2
      ...
      end
      procedure P4
      ..P3..
   end
   end M1
end  MØ
```

Powyższe rozwiązanie niesie w sobie szereg zmian do oryginalnej koncepcji modułu Wirtha:

- Zmienne deklarowane w module, lecz nielokalne dla jego procedur (tzw. zmienne własne modułu), istnieją przez cały czas biegu programu, stanowiąc wygodny bufor, łączący wykonywane w rozdzielnych chwilach procedury.

- Sterowanie przepływem informacji między modułami jest jawne i odbywa się za pomocą dyrektyw z list: importowej **use**, eksportowej **expose** i restrykcji **private**. Użycie każdej z nich jest opcjonalne.

- Lista importowa use dowolnego modułu specyfikuje nazwy modułów, których obiekty własne (udostępniane ich listą eksportową **expose**) stają się widoczne w tym module. Zmienne, do których dostęp uzyskano tą metodą, można jedynie oczytywać, bowiem jedyne legalne operacje na nich reprezentują procedury ich modułów macierzystych, bądź im podległych. Zasada ta stanowi powrót do rozwiązania MODULI, wobec zniesienia jej w bliźniaczych językach SB-MOD i PORTAL.

- Zorganizowane hierarchicznie moduły przejmują funkcje struktury blokowej. Zatem w module widoczne są obiekty deklarowane w nim samym, obiekty stanowiące jego otoczenie [1]), obiekty eksponowane przez moduły, ich nazwy znajdują się na liście importowej **use**) oraz eksponowane przez moduły bezpośrednio lokalne.

- Lista restrykcji **private** specyfikuje identyfikatory tych obiektów własnych modułu, które są ściśle lokalne, tzn. nie mogą być wykorzystane nawet przez moduły hierarchicznie podległe. Jest to lista redundacyjna, z definicji nieobecna w modułach najniższych hierarchicznie. Jej użycie umożliwia uniknięcia kolizji nazw w przypadku wymiany pojedynczego modułu w programie.

- Jednoczesna widoczność, w dowolnym punkcie programu, obiektów o takim samym identyfikatorze jest zawsze błędem, sygnalizowanym przez kompilator. Nie funkcjonuje więc zasada przesłania obiektów hierarchicznie dalszych przez bliższe.

Omówione zasady ilustruje przykład kartoteki pojazdów (przykład 3).

PRZYKŁAD 3

```
module carmanagement;
   expose manage;
   private n;
   type car = record ident : char;
                        ...
                     end
   var n : integer: = 0; ——— licznik stanu kartoteki
   procedure manage (...);
         ... ——— operuje na procedurach
                  modułu  carfile
   end
   module carfile;
      expose add, delete, update, file;
      var file : array/1..1000/of car;
      procedure add(data:car);
         ...
      end
      procedure delete(data:car);
         ...
      end
      procedure update(data:car);
         ...
      end
      procedure findident(id : char, out n : integer);
         ... ——— procedura lokalna modułu
      end
   end carfile
end carmanagement
module supervision;
   use carfile;
      ... ——— inny moduł korzystający z kartoteki
end supervision
```

Moduły SUPERVISION i CARMANAGEMENT korzystają z procedur modułu CARFILE stosując odpowiednio listę **use** i zasadę modułu bezpośrednio lokalnego. Możliwość modyfikacji tablicy FILE uzyskują wyłącznie przez użycie jednej z procedur eksportowanych ADD, DELETE lub UPDATE, natomiast prawo jej bezpośredniego odczytywania uzyskują dzięki umieszczeniu jej nazwy na liście **expose**. Zauważmy, że ewentualne użycie w module CARFILE identyfikatora „n" nie kolidowałoby z taką samą nazwą licznika z modułu nadrzędnego, ponieważ ten ostatni jest zastrzeżony jako **private**.

Tak określona struktura jest kompromisem między prostotą języka a potrzebami współczesnych zastosowań, którym najlepiej odpowiada np. język ADA; choć istnieją pewne z nim analogie (por. [3]). Dyrektywne sterowanie propagacją identyfikatorów zastępuje rozdział jednostek programu na część deklaracyjną i ciało. Przedstawione rozwiązanie jest wynikiem nieco innego spojrzenia na strukturę języka — odzwierciedla ono lepiej inżynierskie wymagania programistów aplikacyjnych. Zbędne stają się niektóre możliwości ADY czy LOGLANU [5], wskutek czego powstaje język (pojęciowo i implementacyjnie) o klasę prostszy.

* * *

Omówiona propozycja nie jest pozbawiona pewnych wad. Po pierwsze — wykorzystanie pamięci operacyjnej jest mniej efektywne w porównaniu ze strukturą blokową, bowiem duża część zbioru zmiennych ma charakter permanentny. Po drugie — język wymaga kompilatora wieloprzebiegowego, o znacznie rozbudowanych mechanizmach operujących tablicami symboli.

Przedstawione podejście ma jednak znacznie więcej zalet.

- Zastąpienie struktury blokowej modułami powoduje, że zbędny jest mechanizm typu display. Każdemu wywołaniu procedury towarzyszy jedynie prosta modyfikacja wskaź-

[1]) Otoczenie modułu stanowią wszystkie obiekty własne i deklaracje definiowane w tych modułach hierarchicznie wyższych, które się kolejno — począwszy od tego modułu — obejmują.

nika do alokowanego segmentu danych. Dostęp do zmiennych własnych modułu można uzyskać stosując adresowanie bezwzględne.

- Moduły nie pozostawiają żadnego śladu w kodzie wynikowym programu.
- Modularyzacja programu pozwala ściśle powiązać jego określone partie z funkcją, jaką pełnią dla fizycznego otoczenia.
- Moduły umożliwiają zabezpieczenie przed niewłaściwym wykorzystaniem wszystkich swoich obiektów, ukrywając przy tym te, które są uznane za szczegóły implementacyjne algorytmu, a eksponując obiekty o bardziej globalnym charakterze.
- Hierarchia modułów odzwierciedla kolejne fazy przetwarzania informacji i umożliwia systematyczne programowanie, analogicznie jak w przypadku struktury blokowej.
- Wychodząc z założenia, że programowanie systemów czasu rzeczywistego jest silnie powiązane z programowaniem współbieżnym, prezentowaną strukturę można łatwo wzbogacić o konstrukcje wielozadaniowości, np. w stylu ADY.
- Opracowanie systemu rozdzielnej kompilacji jest dla struktury modułowej znacznie prostsze niż dla blokowej.

Powyższą koncepcję można zastosować nie tylko do języków czasu rzeczywistego. Nie jest ona obarczona żadną z przedstawionych wad struktury blokowej, natomiast przejmuje wszystkie jej zalety. Z uważnej analizy tej propozycji wynika, że jej istotą jest przerzucenie kosztów ograniczenia widoczności obiektów — z kodu wynikowego programu do kompilatora języka. Cena tej operacji wydaje się być niewielka wobec zysków, które mogą być istotne dla wielu zastosowań.

LITERATURA

[1] Bishop J. M., Barron D. W.: Procedure Calling and Structured Architecture. Computer Journal. Vol. 23, pp. 115—122. 1980
[2] Hanson D. R.: Is Block Structure Necessary? Software — Practice and Experience. Vol. 11, pp. 853—866. 1981.
[3] Ichbiah J. D. et al.: Preliminary Ada Reference Manual. SIGPLAN Notices. Vol. 14, no. 6, part A. 1979
[4] Kowaltowski T.: Parameter Passing Mechanisms and Run Time Data Structures. Software — Practice and Experience. Vol. 11, pp. 757—765, 1981
[5] Kreczmar A., Salwicki A.: Język programowania Loglan. Informatyka, nr 7, 8—9/1982. 1/1983
[6] Randell B., Russel L. J.: Algol 60 Implementation. Academic Press. London, 1964
[7] Tennent R. D.: Two Examples of Block Structuring. Software — Practice and Experience. Vol. 12, pp. 385—392. 1982
[8] Wirth N.: The Design of a Pascal Compiler. Software — Practice and Experience. Vol. 1, pp. 309—333, 1971
[9] Wirth N.: Modula: a Language for Modular Multiprogramming. Software — Practice and Experience. Vol. 7, pp. 3—35, 1977.

PAWEŁ STASIEWICZ
JAN STEPANIEC

Warszawskie Przedsiębiorstwo Informatyki
Przemysłu Budowlanego ETOB

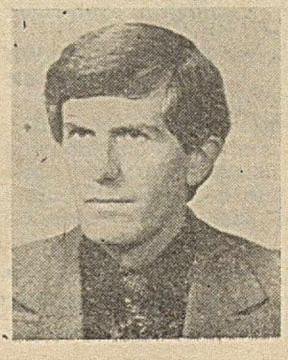
Automatyzacja zdalnej obsługi użytkownika w systemie GEORGE-3

Znaczna część profesjonalnych ośrodków obliczeniowych, zwłaszcza tych ogólnodostępnych, przeżywa od kilku już lat trudności natury ekonomicznej. Załamanie gospodarki i zmiany w sposobie jej sterowania spowodowały zawieszenie eksploatacji wielu systemów o charakterze ponadobiektowym oraz wycofanie zamówień na nowe opracowania. Jednocześnie, przy znacznym wzroście kosztów działalności ośrodków (płace, koszty materialne), możliwy był tylko stosunkowo niewielki wzrost cen usług obliczeniowych. Dała o sobie znać względna taniość pracy administracyjno-biurowej w relacji do cen sprzętu komputerowego, na którym prowadzone są obliczenia. Tak więc ośrodki informatyki znalazły się w sytuacji, w której — według życzeń autorów reformy gospodarczej — powinna była znaleźć się cała gospodarka.

Aby nie zbankrutować, ośrodki te muszą w sposób radykalny obniżać koszty własne działalności oraz zwiększać przy posiadanym potencjale sprzętowym wolumen usług — podnosząc przy tym zarówno ich jakość, jak i atrakcyjność form dostępu użytkownika do komputerów.

W przypadku ośrodków eksploatujących komputery ODRA 1305 droga do intensyfikacji wykorzystania sprzętu oraz podniesienia jakości i atrakcyjności usług prowadzi poprzez takie zrekonfigurowanie sprzętu, aby była możliwa praca pod nadzorem systemu operacyjnego GEORGE-3.

Mgr inż. PAWEŁ STASIEWICZ ukończył w 1960 r. studia na Wydziale Łączności Politechniki Wrocławskiej. Informatyką zajmuje się od 1962 r. Jest autorem szeregu systemów komputerowego wspomagania projektowania miejskich sieci sanitarnych oraz systemów sterowania obliczeniami dla klasy programów naukowo-technicznych w systemie operacyjnym GEORGE-3



Mgr inż. JAN STEPANIEC ukończył w 1962 r. Wydział Konstrukcji Przyrządów Moskiewskiej Wyższej Szkoły Technicznej im. Baumana, specjalność urządzenia i maszyny liczące. W latach 1967—1968 pracował w firmie ICL 1900. Od 1968 zajmuje się systemami zarządzania dla budownictwa i przemysłu materiałów budowlanych. Obecnie kieruje pracami nad rozwojem nowych technologii przetwarzania i zastosowań w WPIPB ETOB.

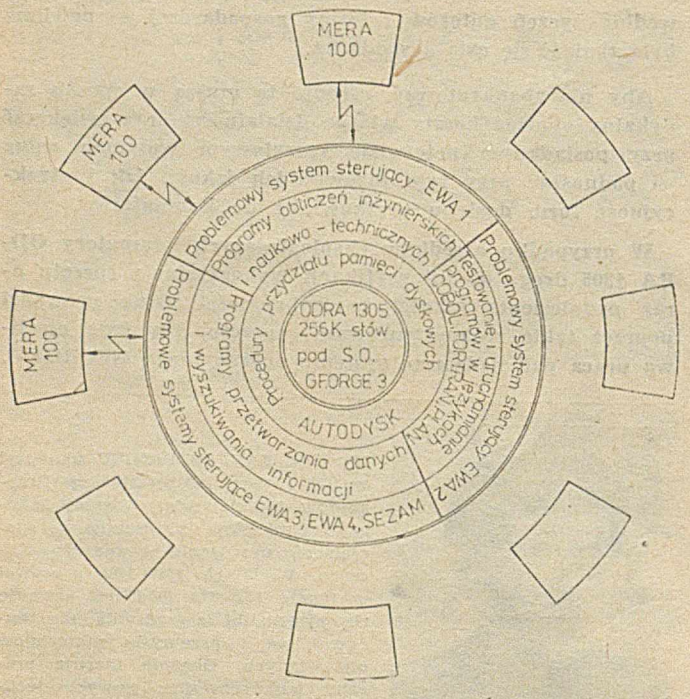
Pozwala to znacznie zwiększyć wydajność w porównaniu do eksploatacji z użyciem egzekutora. Podstawowe walory systemu GEORGE-3, takie jak: wysoki stopień wykorzystania sprzętu (w tym zarówno procesora, który może obsługiwać równocześnie co najmniej kilkanaście programów, jak i znakowych urządzeń zewnętrznych nie współpracujących bezpośrednio z programami), bardzo wygodny i wysoce zautomatyzowany system zdalnej obsługi użytkowników (praca interaktywna w podsystemie MOP i zdalna wsadowa), automatyczne archiwowanie zbiorów wszelkich formatów w pamięci masowej (PZS) sprawny edytor tekstów, automatyczne rozliczanie użytkowników, itp. — są praktycznie znane stosunkowo niewielkiej liczbie ośrodków, które wdrożyły ten system eksploatacji.

Szereg ośrodków przygotowuje się obecnie do jego wdrożenia, zwłaszcza że ostatnio zaistniały realne możliwości zakupu większych pamięci operacyjnych (z firm polonijnych), co jest jednym z warunków efektywnego wykorzystania GEORGE'A-3.

Poza koniecznością posiadania odpowiednio dużej pamięci operacyjnej oraz sprzętu do zdalnej obsługi użytkowników (multiplekser, skaner lub procesor komunikacyjny oraz modemy i terminale), występują jeszcze dwie inne bariery, których pokonanie jest niezbędne do korzystania z systemu GEORGE-3.

Pierwsza — to zapewnienie wszystkim użytkownikom systemu, zwłaszcza użytkownikom zdalnym, bieżącego (bez dłuższego oczekiwania) dostępu do pamięci dyskowych. W zakresie zbiorów niewielkich (do ok. 1 mln znaków) wystarczają standardowe możliwości systemu (zbiory PZS). Problem automatycznego, tj. bez wymiany pakietów dyskowych, przydziału zbiorów większych, w tym zbiorów o topografii optymalnej ze względu na czas sortowań, zgodnie z zapotrzebowaniem wszystkich użytkowników na zbiory dyskowe — został rozwiązany przez zastosowanie grupy procedur o nazwie AUTODYSK, rozszerzających standardowe możliwości systemu operacyjnego.

Druga bariera — to konieczność poznania oraz praktycznego opanowania umiejętności stosowania komend i innych rozwiązań systemu operacyjnego. Podstawowy zasób wiedzy dla programistów zastosować ma w podręczniku producenta komputerów objętość ok. 700 stron. Dla praktycznego opanowania tej wiedzy potrzebny jest okres wielu miesięcy, a nawet kilka lat. Problem ten można znacznie złagodzić dostarczając użytkownikom makrokomend (zwanych dalej problemowymi systemami sterującymi) obsługujących uruchamianie, testowanie i realizację całych klas programów zastosowań. Zwalnia to użytkowników od konieczności obsługi programów za pomocą indywidualnie przygotowywanych zestawów komend systemu operacyjnego, bądź sprawę tę oddala w czasie.



Schemat urządzeń programowych automatyzujących obsługę użytkownika w sieci teleprzetwarzania WPIPB ETOB

Schemat kompleksu rozwiązań programowych automatyzujących obsługę użytkownika w sieci teleprzetwarzania z WPIPB ETOB pokazano na rysunku.

ROZSZERZENIE MOŻLIWOŚCI SYSTEMU GEORGE-3

Zadania przydziału użytkownikom miejsc na pakietach dyskowych oraz montażu i demontażu pakietów w trakcie eksploatacji komputera stwarzają prawie zawsze trudności kierownikom instalacji i operatorom komputera. Problemy są tym poważniejsze, im gorsza jest relacja liczby posiadanych jednostek pamięci dyskowej do liczby użytkowników danego komputera. Zazwyczaj stosowane są dwa sposoby: powiększanie instalacji o dalsze jednostki pamięci dyskowej lub — przy ograniczonych możliwościach w tym zakresie — wyrafinowane techniki harmonogramowania jednoczesnej pracy użytkowników korzystających z pamięci dyskowych.

W prezentowanym rozwiązaniu wykorzystano inną koncepcję, pozwalającą uniknąć wad wymienionych metod (rozbudowy sprzętu lub pracochłonnego harmonogramowania). Metoda ta ma szczególne walory przy dysponowaniu pakietami dyskowymi o dużej pojemności w przypadku, gdy znaczna liczba użytkowników przypada na niewielką liczbę jednostek dyskowych, lub zapotrzebowanie użytkowników na pamięci jest w trakcie dnia roboczego na tyle zmienne, że uniemożliwia efektywne harmonogramowanie montażu przydzielonych im pakietów.

Metoda rozwiązania

Całość lub część jednostek dyskowych obejmuje się zasadą niewymienności zamontowanych w nich pakietów. Łącznie tworzą ono pulę tzw. pakietów wspólnego użytkowania. Zakładanie na nich zbiorów (egzozbiory mianowane w nomenklaturze systemu GEORGE-3) możliwe jest jedynie za pomocą procedury (makrokomendy) UDAS systemu operacyjnego. Rezerwacji dokonują użytkownicy w ramach swoich uruchamianych zdań pod nadzorem systemu operacyjnego, z tym że miejsca umieszczenia poszczególnych zbiorów nie są im znane. Makrokomenda UDAS działa w oparciu o zawarte w odpowiednim zbiorze sterującym numery pakietów wspólnego użytkowania, obciążając poszczególne pakiety jednostki dyskowej w sposób losowy lub inny — ustalony przez kierownika instalacji.

Rezerwacja zbiorów wygasa najpóźniej przy zakończeniu zadania, które je rezerwowało. Jest to więc realizacja pamięci masowej o dostępie swobodnym typu niearchiwalnego. Z użytkowego punktu widzenia daje to taki efekt, jakby instalacja komputerowa miała ogromną, znacznie większą od rzeczywistej, liczbę jednostek dyskowych. Makrokomenda UDAS obsługuje wszelkie niezbędne czynności organizacyjne związane z pracą na egzozbiorach: rezerwację, kasowanie, kopiowanie, składowanie i odzyskiwanie zawartości zbiorów z taśm, wydruki zawartości itp.

Pulę odrębną (całkowicie lub częściowo) od pakietów wspólnego użytkowania tworzą pakiety przeznaczone do obsługi sortowań. Dla danego użytkownika rezerwacji zbiorów roboczych dla sortowania dokonuje procedura (makrokomenda) RESORT. Lokalizacja egzozbiorów wybierana jest z uwzględnieniem minimalizacji czasu sortowania. Rezerwacja zbiorów dla sortowań ważna jest tylko do chwili zakończenia działania programu sortującego lub wyjątkowo — do zakończenia zadania.

Spis pakietów wspólnego użytkowania oraz pakietów do obsługi sortowań pozostaje w dyspozycji kierownika instalacji komputerowej. Stan ich zainstalowania natomiast może się zmieniać w trakcie pracy systemu, w konkurencji z pakietami dla zastosowań czasu rzeczywistego, nie objętymi metodyką wspólnego użytkowania.

Procedurami wspólnego użytkowania może być objęta całość lub jedynie część powierzchni pakietów. Procedury ewentualnego demontażu pakietów, badania stopnia ich aktualnego zapełnienia itp. obsługiwane są przez makroko-

mendę pomocniczą UDASI. Procedury zapewniają pełną bezkolizyjność fizycznych nazwy egzozbiorów poszczególnych zadań. Nazwy te tworzone są bowiem automatycznie — na podstawie nazw zadań i kodów użytkowników (stosowanych dla ich identyfikacji i rozliczeń). Procedury mają niezbędne zabezpieczenia restartowe na wypadek awarii systemu operacyjnego.

Przy składowaniach wykorzystywana jest technologia tzw. długiej taśmy, tj. wielozbiorów formatu *MT, niedostępnych w standardowej wersji systemu operacyjnego.

Omawiane procedury zapisane zostały w języku komend systemu operacyjnego GEORGE-3 (tworzą makrokomendy). Makrokomendy podlegają dostosowaniu do specyfiki instalacji komputerowej, ze względu na liczbę i rodzaj pamięci dyskowych oraz strukturę eksploatowanych zadań, a także przyjęty system kodów użytkowników.

Efekt użytkowy (ekonomiczny)

Zastosowanie omówionej procedury pozwala, nawet przy bardzo ograniczonym wyposażeniu instalacji komputerowej w pamięci dyskowe, rozwinąć pełną moc instalacji na rzecz użytkowników — zarówno lokalnych (przetwarzanie wsadowe), jak i zdalnych (terminale MOP oraz zdalne przetwarzanie wsadowe). Barierę możliwości stanowi tutaj łączna pojemność pamięci dyskowych, która powinna zaspokajać sumę dynamicznie kształtujących się chwilowych potrzeb użytkowników, a nie — liczba posiadanych jednostek dyskowych, która przy tradycyjnych metodach pracy musi być znaczna.

Dysponując — na przykład — pakietami wspólnego użytkowania o pojemności 90 mln znaków (półtora pakietu o pojemności 60 mln znaków) i przyjmując, że przeciętne zadanie potrzebuje ok. 5 mln znaków na egzozbiory, można bezkolizyjnie obsługiwać bez harmonogramowania 18 zadań jednocześnie, tj. symulować sytuację, w której dla użytkowników mamy nie półtora pakietu (90 mln znaków), a 18 pakietów po 8 mln znaków każdy (po 5 mln znaków efektywnie wykorzystywanych).

Minimalny efekt ekonomiczny w przypadku przeciętnej instalacji komputerowej — to możliwość rezygnacji z zakupu pamięci dyskowych i sterownika o koszcie rzędu milionów złotych.

Praca bez wymiany pakietów przynosi także następujące dodatkowe korzyści:

- zmniejszenie obciążenia operatorów komputera w wyniku eliminacji czynności montażu i demontażu pakietów
- zmniejszenie tempa zużycia technicznego mechanizmów dyskowych.

Uzyskany wzrost przepustowości instalacji komputerowej daje rezygnacja z korzystania ze zbiorów formatu *DA w PZS na rzecz pracy z egzozbiorami, zakładanymi i kasowanymi dynamicznie w trakcie realizacji zadań, a także — rezygnacja z wykorzystywania szpul taśmy magnetycznej (ezgotaśm) na rzecz wielozbiorów formatu *MT w PZS (makrokomenda DLUGATASMA).

Do „wspólnego użytkowania", a także sortowania, mogą być wykorzystanie grupy jednostek (pakietów) dyskowych o różnych pojemnościach. Alokacja egzozbiorów roboczych dla sortowań następuje w grupie pakietów o pojemnościach dostosowanych do żądanej wielkości egzozbiorów.

Na instalacjach przeciężonych — z punktu widzenia bieżącego (chwilowego) zapotrzebowania miejsca na dyskach — może być wprowadzony system kolejek (ewentualnie odrębny dla każdej grupy pojemnościowej pakietów). Użytkownicy MOP informowania są wówczas o położeniu swoich zadań w kolejkach. Operatorzy komputera mają natomiast możliwość operatywnego dokonywania w nich zmian w kolejności zadań.

Poza swoimi funkcjami związanymi z organizacją pracy na pakietach wspólnego użytkowania, makrokomenda UDAS pełni także funkcję makrokomendy uniwersalnej obsługującej całość programów organizacyjnych dla pamięci dyskowych (#XPJC, #XPJD, #XPJW itd.).

EWA-1 — OBSŁUGA OBLICZEŃ INŻYNIERSKICH I NAUKOWO-TECHNICZNYCH

Podstawowym zadaniem problemowego systemu EWA-1 jest automatyzacja czynności występujących przy uruchamianiu, testowaniu i realizacji programów inżynierskich i naukowo-technicznych w sieci teleprzetwarzania zdalnego i lokalnego w podsystemie MOP systemu operacyjnego GEORGE-3.

System EWA-1 umożliwia obsługę bez żadnych zmian i przeróbek otwartego zbioru programów użytkowych wykonanych na komputerach ODRA 1300 pod nadzorem egzekutora oraz wiązanie tych programów w systemy, przy wymianie informacji pomiędzy programami na poziomie zbiorów taśmowych, dyskowych lub urządzeń podstawowych — symulowanych w PZS. Podczas ich działania z danych pobieranych z programów binarnych w postaci parametrów linii sterujących, zdarzeń programowych itp. — budowana jest automatyczna makrokomenda ich obsługi, zgodna z wymogami GEORGE-3.

Parametryczne sterowanie systemem umożliwia wprowadzenie danych i wyprowadzanie wyników na różnych nośnikach maszynowych; można np. bezpośrednio przyłączyć do programu urzadzenia minikomputera MERA-100 w postaci minikasety, drukarki mozaikowej lub monitora ekranowego.

Program binarny w systemie może być wywołany (z taśmy magnetycznej, pamięci dyskowej, PZS) komendą ładowania: LOAD, FIND lub skróconą nazwą, jeśli został wprowadzony na stałe do biblioteki programów użytkowych systemu EWA-1.

EWA-2 — URUCHAMIANIE I TESTOWANIE PROGRAMÓW UŻYTKOWYCH

Zadaniem EWY-2 jest przede wszystkim wspomaganie procesów uruchamiania i testowania w systemie GEORGE-3 na komputerach ODRA 1305 programów użytkowych napisanych w języku: PLAN FORTRAN oraz COBOL. EWA-2 umożliwia automatyzację następujących czynności związanych z translacją i realizacją programów użytkowych:

- wybór jednego z trzech możliwych translatorów (PLAN, COBOL, FORTRAN) bez podawania w parametrach dodatkowych informacji
- tworzenie parametrów do makrokomendy obsługi wybranego translatora
- wykonanie przetłumaczonego i skonsolidowanego programu binarnego, bez potrzeby pisania makrokomendy jego obsługi, która jest tworzona dynamicznie w czasie wykonywania programu użytkowego.

System EWA-2 do swego działania pobiera dane z linii sterujących programów źródłowych lub binarnych. Praca w systemie jest bardzo prosta, gdyż jedynym parametrem obligatoryjnym jest nazwa zbioru z programem źródłowym. Program źródłowy napisany w powyższych językach może być umiejscowiony w PZS lub na kasecie magnetycznej minikomputera MERA-100. Program można też wprowadzać dynamicznie (w czasie działania translatora) z klawiatury terminala. Jak wykazała praktyka, system ten jest szczególnie przydatny dla użytkowników terminali. Pozwala on na wykorzystanie bogatego i bardzo efektywnego oprogramowania standardowego zawartego w systemie GEORGE-3 przy minimalnej znajomości języka komend tego systemu.

EWA-3 — OBLICZENIA PROGRAMÓW RESTARTOWALNYCH

Zadaniem EWY-3 jest usprawnienie wykorzystania zasobów dużego komputera oraz urządzeń zainstalowanych w sieci teleprzetwarzania. Usprawnienie obciążeń instalacji można uzyskać poprzez wznowienia restartowalnych programów w zadaniach przerwanych; przerwanych z różnych przyczyn, np:

- załamania i upadku systemu operacyjnego GEORGE-3
- technicznej awarii komputera (z wyjątkiem zniszczenia pakietów dyskowych zawierających PZS)

• zakłóceń w pracy łącz teleprzetwarzania

• działań operatorskich związanych z doborem optymalnej, w danej chwili, mieszanki zadań (posyłanie i sprowadzenie zadań z zasobnika)

• przerwań i zaniechań wykonania zadania przez użytkowników terminali (np. podglądu cząstkowych wyników obliczeń itp.).

System ten jest szczególnie przydatny dla programów, które obciążają zasoby instalacji komputerowej ponad przeciętne parametry ustalone dla danego ośrodka.

Możliwość restartu (automatyczny restart z załamań systemu GEORGE-3) w dowolnym czasie i z dowolnych przyczyn pozwala na optymalne obciążenie danej instalacji, a tym samym zapobiega powstawaniu wąskich przekrojów w złożonym procesie przetwarzania danych. Ma to zasadniczy wpływ na komfort pracy obsługi instalacji i odczuwalne efekty ekonomiczne, gdyż w sposób zasadniczy eliminuje straty powodowane wznawianiem zadań od początku.

W systemie EWA-3 został praktycznie rozwiązany problem eksploatacji restartowalnych programów o długim okresie obliczeń (np. trwających kilka lub kilkanaście godzin), gdyż straty związane z restartami nie mają tu żadnego znaczenia. Przykładowo — przy eksploatacji programu obliczeń zanieczyszczeń atmosfery gazami (PROGRAM ATMOS) straty te wynoszą ułamek procenta czasu obliczeń.

ANDRZEJ W. ABRAMOWICZ
Instytut Maszyn Matematycznych
Warszawa

System automatycznego redagowania tekstów literackich

W Instytucie Maszyn Matematycznych w Warszawie podjęto realizację systemu automatycznego redagowania tekstów — na potrzeby przemysłu poligraficznego. Podstawą prac była umowa zawarta z Ośrodkiem Badawczo-Rozwojowym Przemysłu Poligraficznego. Przy projektowaniu systemu zakładano całkowitą automatyzację składania i łamania tekstów oraz fotoskładu — podstawowych etapów produkcji poligraficznej.

W ramach umowy zrealizowano projekt koncepcyjny i szczegółowy projekt techniczny. Niestety, do tej pory system nie został wykorzystany. Prace przerwano (z przyczyn niezależnych od Instytutu) na etapie realizacji modułów programowych systemu. Warto dodać, że niektóre programy zostały uruchomione, a wśród nich jądro systemu z modułem justowania [1]). Po uruchomieniu system mógłby być stosowany zarówno w dużych drukarniach (przede wszystkim przy produkcji książek), jak i w tzw. małej poligrafii.

Niezbędnym warunkiem jest oczywiście zastosowanie odpowiedniej konfiguracji sprzętowej, przy czym zasadniczą rolę odgrywa tu sama naświetlarka. To właśnie problem największy, w Polsce naświetlarki nie są produkowane. Nieliczne, pojedyncze systemy fotoskładu pracujące w polskich drukarniach są produktami zachodnioeuropejskimi, kupowanymi ad hoc, bez — jak się wydaje — gruntownej analizy dostępnych na tamtejszym rynku systemów do automatycznego przetwarzania danych tekstowych, a także — bez wizji kompleksowej automatyzacji polskiego przemysłu poligraficznego.

Proponowany System Automatycznego Redagowania Tekstów (SART) zaprojektowany jest w oparciu o krajowy sprzęt komputerowy. Jest on przy tym tak zaprojektowany, aby mógł pracować z różnymi naświetlarkami (wymienny moduł programowy postprocesora). Co więcej — system ten wymaga jedynie „nagiej" naświetlarki (wiele firm dostarcza naświetlarki wyposażone w procesor i pełniące pewne funkcje redagujące — oczywiście odpowiednio droższe). Tak więc, aby system mógł być stosowany, należałoby użyć już działającej w Polsce zagranicznej naświetlarki, zakupić nową, lub wyprodukować własną — krajową. Pierwsze rozwiązanie nie ma sensu o tyle, że stosowane pojedyncze urządzenia stanowią element składowy pracujących systemów. Drugie trudne jest do zrealizowania ze zrozumiałych względów, tym bardziej, że jeśli system miałby być wdrażany na szeroką skalę (a takie były założenia), nie wystarczyłby zakup pojedynczego egzemplarza. Wobec znanych trudności i barier, z którymi boryka się przemysł poligraficzny w Polsce, oraz trudności importowych, konkluzja jest oczywista: skonstruowanie rodzimej naświetlarki jest koniecznością. Wydaje się przy czym celowe, aby była to naświetlarka wyposażona jedynie w podstawowe funkcje — wszelkie funkcje redagujące pełniłyby programy systemu. Byłoby to rozwiązanie stosunkowo tanie i uniwersalne, dające jednocześnie dużo swobody przy realizacji oprogramowania systemu.

Celowość wprowadzania komputerowej techniki w produkcji poligraficznej nie budzi — z ekonomicznego punktu widzenia — żadnych wątpliwości. Należy tylko wyrazić nadzieję, że nasze drukarnie możliwie szybko zainteresują się wprowadzeniem na szeroką skalę powszechnie już stosowanej na świecie techniki, nie czekając, aż będące na ich wyposażeniu archaiczne urządzenia ulegną całkowitemu zużyciu.

OGÓLNA CHARAKTERYSTYKA SYSTEMU

Projektując System Automatycznego Redagowania Tekstów zakładano, że będzie on tworzony etapami, a jego rozwój będzie następował przez opracowywanie coraz to nowych modułów programowych, rozszerzających funkcje użytkowe oraz zakres zastosowań. Zakładano też możliwość rozbudowy konfiguracji sprzętowej.

[1]) Ogólne problemy automatycznego redagowania tekstów zostały przedstawione przez Andrzeja W. Abramowicza w INFORMATYCE nr 7—8/83 — przyp. red.

Podstawowym etapem prac było zaprojektowanie i realizacja wersji pilotowej systemu, przeznaczonej do składania tekstów literackich, a więc nie zawierających tak skomplikowanych form składu, jak wzory (np. matematyczne), wykresy, złożone tabele, czy skład wielołamowy. Wersja pilotowa zapewnia jednak w pełni automatyczną realizację najbardziej specyficznych, a jednocześnie najbardziej pracochłonnych procesów, jakimi są składanie i łamanie tekstów oraz sterowanie naświetlarką, za pomocą której będą uzyskiwane formy kopiowe tekstu przeznaczonego do druku.

System pilotowy jest systemem jednostanowiskowym i zawiera formater (tj. program redagujący) działający w trybie wsadowym. Był on realizowany eksperymentalnie na minikomputerze MERA-60 w standardowej konfiguracji (system operacyjny RT-60). Docelowo system byłby implementowany na minikomputerze SM-4 pod nadzorem systemu operacyjnego DOS/RW.

Założono, że programy przetwarzania tekstu, złożone z tekstu źródłowego i rozkazów języka programowania automatycznego składu, będą tworzone i poprawiane poza systemem SART na wyspecjalizowanym stanowisku przygotowywania danych, wyposażonym w monitor z klawiaturą o repertuarze znaków dostosowanym do potrzeb poligraficznych. Znaki z tego repertuaru stanowią jednocześnie kod wejściowy systemu SART.

Program przetwarzania tekstu, czyli program wejściowy przekształcany jest przez kolejne moduły programowe systemu SART — tj. przez moduły translacji, justowania wierszy i formowania stronic — do postaci uogólnionego programu składu tekstu, który zawiera niezbędne dane dla tzw. uogólnionej naświetlarki. Są to następujące dane:

— kody symboli graficznych, które mają być naświetlane, oraz kolejność ich naświetlania
— rodzaj pisma (krój i odmiana)
— wielkość pisma (stopień)
— współrzędne x, y punktu, w którym ma być naświetlany kolejny symbol graficzny.

Uogólniony program składu przetwarzany jest do postaci programu roboczego, tj. do postaci wymagnej do sterowania konkretną naświetlarką. Sterowanie naświetlarką odbywa się w trybie „off-line" — za pośrednictwem dysku elastycznego lub taśmy perforowanej, na których zapisany jest roboczy program składu tekstu.

JĘZYK PROGRAMOWANIA AUTOMATYCZNEGO SKŁADU TEKSTU LITERACKIEGO

Język Programowania Automatycznego Składu Tekstu Literackiego (PASTEL) służy do opisu algorytmów składania i łamania tekstów. Instrukcje tego języka, a są nimi na ogół ciągi znaków zawarte między znakami początku i końca instrukcji, operują na dowolnych napisach złożonych ze znaków alfabetu wejściowego.

W zbiorze instrukcji można wyróżnić pewien podzbiór instrukcji podstawowych, tj. takich, które muszą być użyte w każdym programie składu .Do podzbioru tego należą m.in. instrukcje definiujące format wiersza i strony oraz rodzaj i wielkość pisma. Mają one określone parametry standardowe, które obowiązują wtedy, gdy użytkownik nie użyje instrukcji lub gdy pominie występujący w niej parametr. Parametry standardowe pełnią więc rolę parametrów domyślnych.

Dużym ułatwieniem dla użytkownika jest możliwość tworzenia makroinstrukcji, które służą do podobnych celów jak w innych językach programowania. Ponadto za pomocą tego mechanizmu można w skróconej, czytelnej formie opisywać powtarzające się fragmenty tekstu źródłowego.

Język PASTEL nie jest językiem zbyt elastycznym. Realizacja systemu SART była częściowo finansowana przez Ośrodek Badawczo-Rozwojowy Przemysłu Poligraficznego, którego życzeniem było, aby język programowania w maksymalnym stopniu odzwierciedlał tradycyjne metody stosowane dotychczas przy ręcznej adiustacji technicznej tekstów. Główną zaletą języka jest wspomniany mechanizm makroinstrukcji.

Alfabet języka PASTEL jest zbiorem znaków, których ciągi tworzą dane tekstowe i instrukcje tego języka. Alfabet ten obowiązuje na wejściu systemu przetwarzania tekstów i jest zwany alfabetem wejściowym. Zawiera on:

— duże i małe litery alfabetu łacińskiego
— duże i małe litery polskie oraz niektóre litery międzynarodowe ze znakami diakrytycznymi [2]).
— cyfry arabskie
— znaki pisarskie i specjalne.

Alfabetem wyjściowym jest zbiór znaków stosowany w końcowej fazie przetwarzania, tzn. w fazie naświetlania. Zależy on od możliwości stosowanych urządzeń fotoskładu.

Instrukcje języka mają postać:

< XYZ p1, p2,...,pn >

gdzie: < — znak początku instrukcji, XYZ — trzyliterowy kod operacji, p1, p2,...pn — parametry oddzielone przecinkami, > — znak końca instrukcji.

Definicje makroinstrukcji mają postać

< NAZWA: „tekst" >

gdzie: NAZWA — nazwa makroinstrukcji, „tekst" — dowolny napis zawierający instrukcje i (lub) dane tekstowe. Możliwe jest „zagnieżdżanie" makroinstrukcji.

Wywołanie makroinstrukcji powoduje, że argument „tekst" zostanie zrealizowany w miejscu wywołania. W tym miejscu zostaną wykonane wyspecyfikowane instrukcje.

Typowym przykładem zastosowania makroinstrukcji jest użycie ich do składania tytułów (np. rozdziałów) czy przypisów, które złożone są z reguły pismem o innych parametrach niż w tekście podstawowym. Definicje makroinstrukcji będą wówczas zawierały wszystkie rozkazy określające postać typograficzną określonego fragmentu tekstu. Wystarczy więc wywołać w miejscu gdzie fragment powinien wystąpić, odpowiednią makroinstrukcję, a następnie podać tylko samą treść. Makroinstrukcje są również wygodnym narzędziem w przypadku powtarzających się partii tekstu. Używając ich, unika się wielokrotnego umieszczania w programie składu tego samego tekstu.

Najbardziej charakterystycznymi instrukcjami języka PASTEL są:

- podstawowy parametr pisma — określa numer kroju i odmiany pisma
- stopień pisma — określa rozmiar pisma
- format wiersza
- rozmiar wcięcia akapitowego
- sposób justowania — określa sposób rozmieszczenia tekstu w wierszu; tekst może być justowany do lewego lub prawego marginesu, „do środka" (centrowanie), na pełny format; drugi parametr tej instrukcji określa sposób zakończenia wiersza: bez dzielenia wyrazów na sylaby i z dzieleniem (jeśli wyraz nie da się podzielić na sylaby, to dzielenie nie następuje) oraz z dzieleniem arbitralnym, gdy wyraz nie dzieli się na sylaby
- stopień wiersza (odległość między sąsiednimi wierszami)
- odstępy międzywyrazowe (minimalny, optymalny i maksymalny)
- format kolumny
- przesuw poziomy
- przesuw pionowy
- wypełnienie wiersza określoną kombinacją znaków — instrukcja używana np. przy formowaniu spisów treści o postaci: „tytuł"... „numer strony"; w tym przypadku mamy do czynienia z wypełnieniem kropkami (użytkownik nie może explicite podać liczby kropek)
- spacjowanie („rozstrzelenie" znaków w tekście)
- podkreślenie
- wcięcie (lewo- i prawostronne)
- linia — parametry określają długość i grubość linii
- znak z rozszerzonego repertuaru — definiuje znak z repertuaru urządzenia wyjściowego, który nie należy do zbioru znaków wejściowych
- numer strony — parametry określają sposób rozmieszczenia

[2]) Znaki diakrytyczne — znaki graficzne dodawane do liter dla oznaczenia brzmień odmiennych od brzmień odpowiadających danym literom (np. ą, ę, ź, ś) — przyp. red.

oraz rodzaj i stopień pisma numerów, początkową wartość licznika stron itp.
- pagina żywa (sposób składania)
- przypis (sposób składania)
- odsyłacz przypisu — jeden z parametrów określa wzorzec znaków, z których składa się odsyłacz (dopuszcza się stosowanie różnych wzorców)
- nowy wiersz (określenie końca poprzedniego wiersza), a także — nowy akapit, nowa strona i nowy rozdział
- włączenie naświetlania
- wyłączenie naświetlania
- zatrzymanie przesuwu — umożliwia naświetlenie dowolnej liczby znaków na jednym polu
- komentarz — jego tekst jest ignorowany przy rozliczaniu wiersza
- akcent ruchomy
- zakaz kończenia wiersza na podanym odstępie międzywyrazowym — powoduje łączenie wyrazów, które nie mogą być rozdzielone przy przenoszeniu do następnego wiersza
- punkt podziału wyrazu, określony przez użytkownika.

MODUŁY PROGRAMOWE SYSTEMU

System SART składa się z następujących modułów programowych:
- translatora
- justowania
- stronicowania
- postprocesora.

Zadaniem dwóch pierwszych modułów jest przetworzenie programu wejściowego do postaci pośredniej, która będzie przetwarzana kolejno przez moduły stronicowania i postprocesora, dając w efekcie postać wymaganą przy wykorzystaniu konkretnej naświetlarki.

Translator jest dwuprzebiegowy. W pierwszym przebiegu sprawdza on poprawność formalną programu, wykonując przy tym takie czynności, jak przekodowywanie rozkazów czy przeliczanie wartości parametrów numerycznych na jednostki wewnętrzne systemu. W drugim przebiegu, przy współpracy z modułem justowania, redaguje wersję pośrednią, która zawiera wyjustowany tekst i jest programem wejściowym dla modułu stronicowania (dzielącego tekst na strony).

W wersji pilotowej systemu SART największą wagę przywiązywano do modułów translatora (projektowanego i realizowanego z myślą o ewentualnych rozszerzeniach funkcji użytkowych systemu) oraz justowania (podstawowym kryterium oceny systemu pilotowego była poprawność i estetyka redagowanych wierszy). Język PASTEL zawiera bardzo bogate możliwości typograficznego kształtowania wiersza przez użytkownika, niespotykane nawet w najbardziej złożonych systemach zagranicznych, co jednak powoduje, że moduł justowania jest bardzo rozbudowany.

Poszukując optymalnego — ze względu na wielkość odstępu międzywyrazowego — punktu zakończenia wiersza, moduł justowania musi uwzględniać cały szereg kryteriów. Nie ogranicza się on jedynie do prostego wyboru najkorzystniejszej spacji lub najlepszego z możliwych punktów podziału na sylaby, jak to rozwiązano w innych systemach. Konieczność uwzględnienia wielu kryteriów, których kolejność stosowania zmienia się w zależności od wielkości formatu wiersza i występujących w nim obiektów, wynika z jednej strony z możliwości pełne ingerencji użytkownika w sposób justowania, z drugiej zaś — z reguł zawartych w branżowych normach poligraficznych.

W obrębie tego samego wiersza mogą obowiązywać różne opcje, dotyczące dzielenia wyrazów na sylaby, oraz zakaz rozdzielania, przy przenoszeniu pewnych grup wyrazów do następnego wiersza. Ponadto użytkownik może wyznaczać punkty podziału wyrazów. Mogą pojawić się linie, znaki specjalne i stałe odstępy oraz takie specyficzne instrukcje, jak wypełnienie wiersza bądź spacjowanie. Wszystko to wpływa na wybór optymalnego punktu zakończenia wiersza. Przy składaniu małych formatów są ponadto stosowane specyficzne reguły redagowania.

Przy realizacji systemu przyjęto zasadę, że zatrzymuje on pracę wyłącznie w przypadku wystąpienia takich błędów, przy których dalsze przetwarzanie jest niemożliwe. Zasada ta jest konsekwencją specyfiki typograficznego kształtowania tekstu i związanej z nią semantyki języka programowania. Może pojawić się bowiem wiele sytuacji, o których trudno rozstrzygnąć, czy zostały zamierzone przez użytkownika, czy nie. Po pierwszym przebiegu moduł translatora informuje użytkownika o wykrytych błędach, dając w ten sposób możliwość ich poprawienia. Faza redagowania przebiega już bez ingerencji użytkownika.

Przykładem błędu, przy wystąpieniu którego system przerywa redagowanie, jest sytuacja, gdy format wiersza jest tak mały, że nie mieści się w nim przygotowany do złożenia pierwszy w tym wierszu pojedynczy symbol graficzny. Tego typu błędy semantyczne mogą być wykryte dopiero w fazie formatowania tekstu.

* * *

Główne zalety systemu SART (podobnie jak innych tego typu komputerowych systemów przetwarzania tekstów) to:
- automatyzacja najbardziej pracochłonnych procesów wydawniczych
- wysoka jakość poligraficzna wydawnictw
- usprawnienie archiwizowania i ponownego wykorzystania tekstów (np. przy wznowieniach)

a ponadto pełnej ingerencji użytkownika w sposób redagowania tekstu.

Jak już wspomniano, język programowania automatycznego składu PASTEL uwzględnia specyfikę sposobu adiustacji dotychczas stosowanego w poligrafii, nie powinien więc być trudny do opanowania. Istotną cechą systemu jest fakt, że umożliwia on stosowanie różnych naświetlarek, przy czym instalacja naświetlarki wymaga dołączenia odpowiedniego postprocesora.

Przegląd języków wysokiego poziomu opracowany został na podstawie referatu „An Overview of High-Level Languages" wygłoszonego przez Jean E. Sammet na konferencji SEAS AM'82 we wrześniu 1982. J. E. Sammet jest pracownikiem Działu Systemów Rederalnych w firmie IBM (Bethseda, Maryland), a w kołach naukowych jest znana z wielu publikacji, zwłaszcza cyklicznie publikowanych od 1968 — „Spisów języków" („Language Rosters"). Referat jest próbą opisu całokształtu współczesnej wiedzy w tej dziedzinie oprogramowania, a także zarysu przewidywanego w niej rozwoju.

Opracowanie polskie jest w stosunku do oryginału znacznie skrócone, niemniej i tak byliśmy zmuszeni do podzielenia całości na kilka części, które będziemy publikować w kolejnych numerach. (Red.)

Przegląd języków wysokiego poziomu (1)

Mimo że termin „języki wysokiego poziomu" jest stosowany od dawna, to właściwie ciągle jeszcze nie znaleziono ścisłej jego definicji. Na potrzeby niniejszego przeglądu można użyć definicji, mającej na celu ustalenie podziału pomiędzy językami, które intuicyjnie uważamy za języki wysokiego poziomu (np. FORTRAN, COBOL) i językami, o których, także intuicyjnie, wiemy, że nimi nie są (np. asemblery). Natomiast rozróżnienie pomiędzy językami wysokiego poziomu a bardzo rozwiniętymi pakietami zastosowań niestety nie jest już tak jasne.

Język wysokiego poziomu ma następujące cztery właściwości:

- nie wymaga od użytkownika znajomości kodu maszynowego
- jest niezależny od konkretnej maszyny, a w związku z tym napisany w nim program może być wykonany na różnych maszynach (doświadczenie wykazało, że pełnej niezależności nie można osiągnąć)
- w procesie translacji jeden element kodu źródłowego jest zazwyczaj zamieniany na kilka instrukcji maszynowych (kryterium to nie odnosi się do każdego elementu kodu źródłowego, lecz jedynie do jego wykonywalnych elementów; w szczególności nie dotyczy deklaracji danych)
- w języku jest stosowana notacja podobna do naturalnego zapisu, przyjętego powszechnie w problematyce, na potrzeby której język ten stworzono.

Stosowanie niektórych z wyżej podanych kryteriów jest dosyć trudne w przypadku pewnych grup języków. Na przykład, istnieją języki określane czasem jako „języki pośrednie" (ang. „midway") lub „języki zorientowane maszynowo" (ang. MOL, Machine — Oriented Languages), które zawierają konstrukcje typowe dla języków wysokiego poziomu, ale jednocześnie umożliwiają autentyczny, bezpośredni dostęp do kodu maszynowego. Najwcześniejszym i najważniejszym prototypem większości języków należących do tej grupy był język PL/360 (1968), którego twórcą był Wirth. Trudno jest także odróżnić języki asemblerowe, wyposażone w rozbudowane zbiory makrorozkazów, od bardzo prostych języków wysokiego poziomu. Istnieją też języki programowania charakteryzujące się tak prostymi rozkazami i zapisem, że programy napisane w tych językach wyglądają jak parametry dla pakietów zastosowań. Z drugiej strony istnieją bardzo rozwinięte pakiety zastosowań, które dopuszczają różne typy parametrów oraz instrukcji sterujących, co również utrudnia ustalenie granic pomiędzy nimi a językami wysokiego poziomu.

Najważniejszym elementem charakteryzującym język wysokiego poziomu jest obszar zastosowań, dla którego język został zaprojektowany. Obszar ten może być szerszy lub węższy, a stwierdzenie tego zależy często od stopnia znajomości dziedziny zastosowania. Na przykład, gdy pada stwierdzenie, że FORTRAN jest przede wszystkim użyteczny dla zastosowań naukowych, to obszar zastosowań FORTRANU z pewnością jest uważany za szeroki. Stwierdzenie to nie uwzględnia jednak np. różnicy pomiędzy obliczeniami numerycznymi a operowaniem formułami, czego w języku tym nie można efektywnie zrealizować.

Okazuje się, że im mniej wiadomo na temat konkretnej dziedziny, tym bardziej wydaje się ona jednolita, a przypadkowy obserwator często uważa, że nie ma potrzeby wprowadzania dla niej różnych języków programowania. Na przykład — osoba, która nie jest dobrze wprowadzona w problemy techniki może mieć trudności w stwierdzeniu, że potrzebne są różne języki dla tak różnych zastosowań, jak projektowanie logiczne oraz budownictwo. Nawet jeśli ktoś w tym przypadku uznaje, że są to dwie różne dziedziny zastosowań, to często zdarza się, że trudno mu z kolei uwierzyć w dalszy podział budownictwa, uwzględniający rozmaite aspekty tej dziedziny techniki. W praktyce istnieje wiele odrębnych języków dla różnych działów i aspektów budownictwa. Oczywiście nie wszystkie te języki są szeroko stosowane.

Niektóre z języków, których nie można jednoznacznie zakwalifikować do języków wysokiego poziomu, zalicza się czasem do specjalnych kategorii, jak np. języków rozkazów czy języków redagowania tekstów.

Należy uświadomić sobie, że chociaż dany język został zaprojektowany z myślą o konkretnym obszarze zastosowań, to zazwyczaj można go stosować również w innych obszarach.

NAJWAŻNIEJSZE WYDARZENIA W ROZWOJU JĘZYKÓW PROGRAMOWANIA

Poniżej zostanie przedstawiona krótka historia tego co wydarzyło się w dziedzinie języków wysokiego poziomu na przestrzeni ostatnich dwudziestu lat. Ażeby z tej perspektywy można było pełniej ocenić etap, w którym znajdujemy się obecnie, omówione zostaną również wydarzenia wcześniejsze.

Lata 1945—1953

Najwcześniejszym projektem języka programowania, który spełnia podaną wyżej definicję języka wysokiego poziomu był „PLANKALKÜL" opracowany przez K. Zuse w Niemczech (1945). Niestety projekt ten nie został zrealizowany. Następnym krokiem w tej dziedzinie był SHORT

CODE, zrealizowany przez J. Mauchly i współpracowników w firmie REMINGTON RAND UNIVAC (1949—1950). Kolejny był nie nazwany i nie zrealizowany język, którego autorem był Rutishauser w Szwajcarii (1952). W 1953 r. J. Backus wraz ze współpracownikami opracował SPEEDCODING dla komputera IBM 701. W tym samym czasie w firmie REMINGTON RAND powstały języki A-2 i A-3, oparte na trójadresowym pseudokodzie dla operacji matematycznych, a w firmie lotniczej BOEING opracowano BACAIC, również dla komputera IBM 701.

Wszystkie te języki, a także inne, które w tym okresie opracowano, próbowały dać użytkownikom komputerów, którymi byli wtedy z reguły naukowcy, notację bardziej podobną do stosowanej w matematyce niż kod wewnętrzny maszyny. Niektóre z tych języków pozwalały użytkownikom zapisywać wyrażenia matematyczne w stosunkowo wygodnej postaci, ale większość z nich takich możliwości jeszcze nie miała. Żaden z nich nie wywarł jednak większego wpływu na języki programowania i nie wniósł trwałych elementów do ich rozwoju. Miały one również niewielki wpływ na sposób myślenia ludzi. W maju 1953 r. J. H. Laning Jr i N. Zierler w uniwersytecie MIT uruchomili na komputerze WHIRLWIND system programowania, który jako pierwszy w USA pozwalał użytkownikowi zapisywać wyrażenia w postaci bardzo już zbliżonej do zapisu stosowanego w matematyce.

Lata 1954—1960

W 1954 r. rozpoczęto w firmie IBM pod kierunkiem J. Backusa prace nad językiem FORTRAN, które zakończono pod koniec 1956. Mniej więcej w tym samym czasie gdy opublikowano wstępny raport na temat FORTRANU, w firmie REMINGTON RAND (UNIVAC) grupa pod kierunkiem G. Hopper rozpoczęła prace nad językiem AT-3, który potem stał się znany jako MATH-MATIC. J. Backus twierdzi, że wysłał do firmy REMINGTON egzemplarz wstępnego projektu FORTRANU z listopada 1954, ale obecnie trudno jest ocenić jak duży wpływ miał ten raport na opracowanie języka MATH-MATIC. Faktem jest jednak, że dokumentacja wstępnego projektu FORTRANU wyprzedziła wszelkie dokumenty dotyczące projektu języka MATH-MATIC.

MATH-MATIC koncepcyjnie był podobny do FORTRANU, jednak różnił się od niego syntaktyką. Trudno jest stwierdzić, który z tych dwóch języków zrealizowano jako pierwszy. Jednak ze wszystkich prac prowadzonych równolegle w połowie lat pięćdziesiątych jedynie FORTRAN wytrzymał próbę czasu. Aż do 1957 roku oba wspomniane języki były pierwszymi ważniejszymi przykładami praktycznego zastosowania języków wysokiego poziomu o cechach zbliżonych do języków, które znamy i stosujemy obecnie.

Jak już podkreślono, główny nacisk był położony początkowo na rozwój języków do zastosowań naukowych. Dopiero w 1958 roku w firmie REMINGTON RAND opracowano podobny do angielskiego język FLOW-MATIC, przeznaczony do przetwarzania danych. Projekt języka i jego implementację na komputerze UNIVAC I wykonano pod kierunkiem Grace Hopper.

Jeszcze jeden język z tego okresu zasługuje na wzmiankę, a mianowicie język APT. Prace nad nim rozpoczęto w 1956 r. w MIT pod kierunkiem Douglasa T. Rossa. Był to pierwszy język opracowany dla bardzo specyficznej dziedziny zastosowań, a mianowicie numerycznego sterowania obrabiarek. O użyteczności tego języka świadczy fakt, że w zmodyfikowanej postaci był on w 1980 r. jeszcze szeroko stosowany.

Lata 1958 i 1959 obfitowały w wydarzenia, które miały duży wpływ na rozwój języków programowania, zarówno w ośrodkach uniwersyteckich, jak i poza nimi. Wymienić tu należy następujące wydarzenia:

- Język IAL (International Algebraic Language), który stał się sławny jako ALGOL 58. IAL wywarł głęboki wpływ na programowanie, ponieważ był poprzednikiem ALGOLU 60, jednego z najważniejszych do chwili obecnej języków programowania, a także był podstawą rozwoju trzech innych języków (NELIAC, MAD i CLIP). Wszystkie te języki, poza CLIPEM, który stał się wzorem dla języka JOVIAL, były szeroko stosowane.
- Język do przetwarzania list (początek 1958 r.)
- Rozpoczęcie prac nad językiem do przetwarzania list LISP, który miał być przeznaczony na potrzeby badań nad sztuczną inteligencją
- Pierwsza implementacja języka do przetwarzania napisów COMIT
- Utworzenie w maju 1958 r. komitetu CODASYL (Conference on Data Systems Languages), który przyczynił się do rozwoju języka COBOL przez zdefiniowanie jego opisu
- Opracowanie i udostępnienie opisów języków AIMACO, COMMERCIAL, TRANSLATOR oraz FACT, przeznaczonych do przetwarzania danych
- Opublikowanie opisu języka JOVIAL.

Spośród języków, pochodzących z lat 1958—1959 lub wcześniejszych, do 1980 roku przetrwały i były dość szeroko używane: ALGOL 60, APT, COBOL, FORTRAN, JOVIAL i LISP.

Duży wpływ na rozwój większości wymienionych osiągnięć miały względy ekonomiczne, a mianowicie wysokie koszty programowania. Każde narzędzie, które mogło zmniejszyć te koszty było przyjmowane bardzo życzliwie. Jednakże decydujący wpływ na akceptację każdego języka miała zwykle analiza, czy koszt wynikający z czasu realizacji tych języków na komputerze nie przewyższa oszczędności czasu ludzkiego, jakie powstają z usprawnienia procesu programowania.

Lata 1961—1970

Lata 1961—1970 to okres dojrzewania dziedziny języków programowania. W tym okresie walka o stosowanie języków wysokiego poziomu została praktycznie wygrana w tym sensie, że stosowanie kodu wewnętrznego stało się już raczej wyjątkiem niż regułą. Chociaż idea opracowywania oprogramowania podstawowego przy użyciu języków wysokiego poziomu została zaakceptowana, to jednak w dalszym ciągu w kodzie wewnętrznym powstawało więcej programów systemowych niż programów zastosowaniowych. Niektóre zalety języków wysokiego poziomu odnoszą się również do dużych makrosystemów oraz „języków pośrednich" (ang. „half-way"), takich jak PL/360, które jednak nie były niestety niezależne od maszyny.

W omawianej dekadzie szczególnie duże znaczenie zdobyły trzy nowe języki, a mianowicie ALGOL 60, COBOL i PL/I, z których tylko dwa ostatnie powszechnie stosowano w USA. W okresie tym zdefiniowano ALGOL 68, a około 1970 r. rozpoczęto jego realizację.

Wprowadzenie w połowie lat sześćdziesiątych koncepcji podziału czasu oraz pracy konwersacyjnej spowodowało rozwój języków przeznaczonych do pracy w trybie bezpośrednim: języka JOSS, a następnie — języka BASIC. Oba te języki były bardzo szeroko stosowane; każdy z nich ma wiele rozszerzeń i wielu naśladowców. W latach sześćdziesiątych, w pewnych specyficznych kręgach użytkowników, stał się również popularny język APL/360.

Impulsem rozwoju języków do operowania formułami były języki FORMAC oraz FORMULA ALGOL, chociaż tylko pierwszy z nich był szeroko stosowany. Pojawienie się języka SNOBOL spopularyzowało przetwarzanie napisów.

Języki symulacyjne GPSS i SIMSCRIPT umożliwiły prowadzenie na szerszą skalę prac związanych z symulacją, a także stały się zachętą do poszukiwania innych języków symualcyjnych. W dalszym ciągu prowadzone były prace nad językami dostosowanymi do specjalnych obszarów zastosowań (np. budownictwa, testowania sprzętu).

Być może z punktu widzenia praktyki jednym z najważniejszych osiągnięć (chociaż lekceważonym przez teoretyków), był w tym okresie postęp w opracowywaniu norm FORTRANU i COBOLU oraz rozpoczęcie standaryzacji języka PL/I.

Lata 1971—1980

Okres 1971—1980 przyniósł zaledwie kilka poważniejszych osiągnięć w dziedzinie języków wysokiego poziomu.

Wśród nich znalazły się:

- realizacja ALGOLU 68 i jego początkowo niewielkie wykorzystanie
- realizacja i bardzo szerokie stosowanie języka PASCAL
- ogromne natężenie w Departamencie Obrony USA prac związanych z rozwojem języka, który ostatecznie nazwano ADA, a przeznaczonego dla wbudowanych systemów komputerowych
- koncepcja ogólności danych (ang. data abstractions)
- koncepcja programowania funkcjonalnego Backus'a (1978)
- języki eksperymentalne, takie jak CLU, EUCLID, SETL.

Na razie za wcześnie jest jeszcze na stwierdzenie, które z tych koncepcji lub języków będą miały wpływ na dziedzinę programowania.

Ponieważ języki wysokiego poziomu są bardzo szeroko stosowane, często pojawia się pytanie dotyczące ich liczby i struktury rodzajowej. Istnieje na ten temat wiele mitów. Mity te potęgują fakt istnienia zarówno wielu organizacji, jak i publikacji poświęconych wyłącznie językom programowania.

LICZBA I WYKORZYSTANIE JĘZYKÓW

Mierzenie wielkości charakteryzujących języki wysokiego poziomu jest bardzo trudne. Ustalenie aktualnej liczby istniejących języków wiąże się z problemem określenia między nimi różnic, tzn. stwierdzenia, które języki różnią się między sobą w sposób zasadniczy, a które są tylko niewielkimi modyfikacjami innych języków. Niestety, jest to zagadnienie do tej pory jeszcze bardzo słabo zdefiniowane. Aby rozstrzygnąć problem nie można bowiem brać pod uwagę jedynie nazw języków. Rozpatrzmy na przykład przypadek języków SIMSCRIPT I, I.5, II, II.5, II.5+, które pojawiają się w różnych publikacjach. Po szczegółowym porównaniu tych języków autorka opracowania stwierdziła, że wśród pięciu wymienionych tylko dwa języki są istotnie różne (SIMSCRIPT I.5 i II.5), natomiast pozostałe są niewielkimi modyfikacjami jednego z tych dwóch języków. Jak więc widać, rozwiązanie tego problemu jest niezbędne do rozważań nad liczbą języków, zarówno istniejących w przeszłości, jak i obecnie. Autorka od wielu lat publikowała „Spis języków programowania" („Roster of Programming Languages"), zawierający listę i podstawowe informacje o wszystkich językach używanych w USA.

W spisach tych języki zostały pogrupowane według głównych rodzajów zastosowań, dla których zostały zaprojektowane, lub według sposobu obecnego wykorzystywania. Spisy uwzględniają następujące obszary zastosowań: obliczenia numeryczno-naukowe, przetwarzanie danych, przetwarzanie napisów i list, operowanie formułami oraz „różne dziedziny zastosowań" (ang. „multipurpose"). Języki należące do tej ostatniej grupy są często nazywane uniwersalnymi (ang. general purpose), co jednak nie jest właściwym określeniem (gdyby takie języki naprawdę istniały, to nie byłyby potrzebne pozostałe rodzaje).

Na podstawie wspomnianych spisów okazało się, że główne języki programowania, takie jak BASIC, COBOL, FORTRAN, są stosowane znacznie szerzej niż początkowo zakładano i z tego powodu należy je właściwie zaliczać do ostatniej grupy.

Poza wymienionymi, istnieje duża grupa języków, które autorka określa jako „języki do specjalnych zastosowań" (ang. languages for specialized application areas, specialized application langugages lub special—purpose languages). Przykładami języków należących do tej grupy są: APT (sterowanie numeryczne), COGO (geometria analityczna) oraz SIMSCRIPT (symulacja dyskretna). Niektórzy dziwią się, że języki do operowania formułami nie są zaliczane do grupy języków „specjalnych"; bierze się to stąd, że operowanie formułami jest działem nienumerycznej matematyki, a matematyka jest wykorzystywana w tak wielu dziedzinach zastosowań, że trudno ją traktować jako specjalny obszar zastosowań.

Tabela 1. Rozwój języków wysokiego poziomu wg dziedzin zastosowań

Dziedziny zastosowań	Liczba języków w poszczególnych latach				
	1971	1972	1973	1975	1977
Numeryczno-naukowe	brak danych		32	28	20
Przetwarzanie danych			3	4	4
Przetwarzanie napisów i list			14	12	11
Operowanie formułami			12	11	10
Różne			24	24	34
Specjalne			86	88	90
Łącznie	165	184	171	167	169

Autorka wiele lat temu zaobserwowała interesujące zjawisko — przy każdym kolejnym przygotowaniu „Spisu języków" liczba języków do specjalnych zastosowań była mniej więcej równa połowie wszystkich języków. Poza tym stwierdziła, że liczba języków w poszczególnych dziedzinach zastosowań pozostaje od wielu lat na tym samym mniej więcej poziomie. Widać to w tabeli, opracowanej na podstawie „Spisu języków". Nie są to zbyt pewne dane i nie należy wyciągać z nich zbyt daleko idących wniosków, ale na usprawiedliwienie należy podkreślić, że są to jedynie dostępne dane tego typu. Dane w tabeli mają pokazać tendencje w dziedzinie języków wysokiego poziomu.

Analiza poszczególnych liczb bez dodatkowych komentarzy może jednak doprowadzić do błędnych wniosków — na przykład zmniejszenie liczby języków do naukowych zastosowań numerycznych zostało częściowo spowodowane zmianą klasyfikacji niektórych z tych języków (np. APL), a mianowicie część z nich została zaliczona do grupy języków specjalnych zastosowań. Ponadto, niewielka liczba języków do przetwarzania danych wynika z dominacji COBOLU w tej kategorii języków.

Bardzo często stawiane jest pytanie, który z języków programowania jest najszerzej stosowany. Autorka uważa, że jest to źle sformułowane pytanie i dlatego trudno na nie prawidłowo odpowiedzieć. Jest wiele parametrów, które mogą być brane pod uwagę celem określenia stopnia wykorzystania języków wysokiego poziomu. Parametrami takimi mogą być: liczba instalacji, liczba użytkowników, a także — programów, wierszy programów źródłowych lub programów wynikowych, kompilatorów, maszyn z kompilatorami, czy opisanych algorytmów. Można także brać pod uwagę ilość czasu maszyny wykorzystanego na: kompilację, opracowanie, uruchamianie, wykonywanie programów. Gdy się więc mówi o „szeroko stosowanych językach" należy najpierw ustalić co się przez to rozumie.

Opracowanie:
TERESA WOJCIEKIAN
HALINA CIECHOMSKA

W poprzednim numerze przedstawiliśmy pierwszego trzydziestolatka sieci ETOB — przedsiębiorstwo warszawskie. Poniżej autoprezentacja rówieśnika, ETOBU w Krakowie. W następnych numerach — korzystając z okazji jubileuszu — przedstawimy jeszcze przedsiębiorstwo katowickie, a także kilka prac twórczych pracowników sieci. (Red.)

30 lat krakowskiego ETOBU

Krakowskie Przedsiębiorstwo Informatyki Przemysłu Budowlanego ETOB jest jednym z najstarszych ośrodków obliczeniowych w Polsce. Powstało w 1953 roku równolegle z oddziałem warszawskim, jako Oddział Biura Rozliczeń Budownictwa Przemysłowego. Przedmiotem działania miało być „prowadzenie ewidencji gospodarki materiałowej, rozliczeń robocizny i kosztów własnych za pomocą maszyn statystyczno-rachunkowych, opracowywanie projektów organizowania rozliczeń oraz instruktaż w zakresie dokumentacji w odniesieniu do jednostek podległych Ministrowi Budownictwa Przemysłowego". Pierwsze wyposażenie stanowiły dwa zestawy maszyn liczącо-analitycznych.

W styczniu 1969 r. Oddział Krakowski BRB został przekształcony na Centrum Elektronicznej Techniki Obliczeniowej Przemysłu Budowlanego ETOB Oddział w Krakowie. Następne lata (1970—1973) to okres szybkiego rozwoju przedsiębiorstwa. Utworzono zakłady obliczeniowe w Rzeszowie i Kielcach. Zainstalowano i uruchomiono pierwsze komputery: dwie maszyny MIŃSK-32, ODRĘ-1304 i ODRĘ-1103, co pozwoliło na znaczny wzrost jakościowy i ilościowy świadczonych usług obliczeniowych. W lipcu 1972 r. Oddział w Krakowie przekształcono w Krakowskie Przedsiębiorstwo Informatyki Przemysłu Budowlanego ETOB.

W latach 1974—1977 nastąpiły dalsze zmiany. Kupiono i zainstalowano dwa komputery ODRA-1305. Zakład obliczeniowy w Rzeszowie został prze-

Oprogramowanie narzędziowe

Mogłoby się wydawać, iż posiadanie zróżnicowanego sprzętu komputerowego świadczy o prężności ośrodka obliczeniowego, ponieważ kryje się za tym wszechstronna kadra, większa moc przerobowa, szeroka gama ofert dla użytkownika itp. Wszystko to prawda. Ale jest jeszcze druga strona medalu: codzienne kłopoty.

Zróżnicowanie sprzętowe stwarza duże trudności przede wszystkim służbom eksploatacyjnym. Nakłada to z kolei większe wymagania w stosunku do oprogramowania narzędziowego, a zwłaszcza w zakresie konwersji zbiorów danych między jakościowo różnymi systemami komputerowymi i minikomputerowymi. Równocześnie dostarczane przez producentów oprogramowanie podstawowe, szczególnie w pierwszych fazach rozwoju produkcji, nie zaspokaja niekiedy nawet podstawowych potrzeb użytkowników. Między innymi z tego też powodu, wśród użytkowników systemu ODRA-1300 rozpowszechniły się np. najrozmaitsze programy „scratch' 'ujące" czy wręcz — modyfikowane w sposób bardziej lub mniej udany egzekutory. Mają one usuwać niedogodności programu XOMY, oferowanego przez producenta.

Krakowskie Przedsiębiorstwo Informatyki Przemysłu Budowlanego ETOB wyposażone jest obecnie w bardzo zróżnicowany park maszynowy:

- system MIŃSK-32, zainstalowany w roku 1972 (przewidziany do likwidacji w 1984)
- system ODRA-1305 (nr fabr. 16), zainstalowany w roku 1974
- system R-32 (nr fabr. 80), zainstalowany w roku 1979.

Ponadto w ostatnich trzech latach zainstalowane zostały następujące systemy minikomputerowe:

- MERA 9150 (dwa zestawy z 28 stanowiskami łącznie)
- PSPD-90 (Programowana Stacja Przygotowania Danych)
- MERA 100.

Celem niniejszych rozważań jest zaprezentowanie własnych prac w zakresie oprogramowania narzędziowego przy tak zróżnicowanym parku komputerowym. Systematyzując materiał, dokonano podziału na oprogramowanie dotyczące: systemu ODRA-1300, systemu R-32 oraz wspomnianych systemów minikomputerowych. Pominięta zostanie natomiast prezentacja opracowań dla kończącego pracę komputera MIŃSK-32.

*

Z chwilą zainstalowania w roku 1974 komputera ODRA-1305 przedsiębiorstwo było wyposażone w dwa komputery MIŃSK-32 oraz w cztery zestawy maszyn analitycznych systemu ARITMA. W związku z tym powstał problem przeniesienia zbiorów danych na nowy komputer. W tym celu opracowano następujące dwa programy:

- program ZR90 o podobnych funkcjach jak standardowy program XRMH, lecz odczytujący karty 90-kolumnowe systemu ARITMA
- program ZMTM, dokonujący konwersji zbiorów danych zapisanych na taśmach magnetycznych w systemie MIŃSK-32 na format systemu ODRA-1300.

W trakcie rozwoju i zdobywania doświadczeń eksploatacyjnych pojawiło się wiele nowych zagadnień, wymagających programowych rozwiązań. I tak dla przykładu:

- W systemie ODRA-1300 w porównaniu chociażby z systemem MIŃSK-

kształcony w samodzielne przedsiębiorstwo.

W latach 1978—1979 kupiono i zainstalowano maszynę R-32 oraz dwie jednostki MERA 9150.

W 1981 roku zakład obliczeniowy w Kielcach został przejęty przez Centrum ETOB w Warszawie.

Przedsiębiorstwo świadczy usługi informatyczne przede wszystkim dla przedsiębiorstw resortu budownictwa, przedsiębiorstw budowlanych innych resortów oraz biur projektowych:

- projektowanie, programowanie i wdrażanie systemów na potrzeby zarządzania, sterowania procesami inwestycyjnymi i obliczeń inżynierskich
- eksploatację systemów ewidencyjnych, inżynierskich i sterujących procesami inwestycyjnymi
- doradztwo organizacyjne w zakresie zarządzania — projektowanie harmonogramów sieciowych robót budowlanych oraz wykonywanie przedmiarów robót dla kosztorysowania
- serwis techniczny urządzeń do przygotowywania danych oraz minikomputerów zainstalowanych u użytkowników.

Przedsiębiorstwo zatrudnia 220 pracowników, w tym 36 w pionie projektowania.

Krakowskie Przedsiębiorstwo Informatyczne
Przemysłu Budowlanego ETOB
ul. Wadowicka 10
30-415 Kraków
Telefony:
centrala: 66-80-22
dyrektor: 66-72-31
z-ca dyrektora ds. produkcji: 66-28-09
główny technolog: 66-60-36
główny elektronik: 66-69-40
koordynator ds. produkcji: 66-60-36
dział rozpowszechniania systemów: 66-09-86
Teleks: 032-22-75

Przedsiębiorstwo eksploatuje obecnie systemy na rzecz 74 użytkowników.

Systemy te powstały głównie w przedsiębiorstwie i dawnej organizacji CENTRUM ETOB. Dominują systemy ewidencyjne związane z zarządzaniem. Równocześnie zwiększa się zapotrzebowanie ze strony przedsiębiorstw wykonawstwa inwestycyjnego na usługi obliczeniowe w zakresie kosztorysowania robót budowlanych.

W wyniku reformy gospodarczej daje się zauważyć zwiększone zainteresowanie użytkowników urządzeniami minikomputerowymi. Przedsiębiorstwo nie zostało tym zaskoczone, jeszcze w roku 1980 podjęte zostały prace przygotowawcze, poprzedzające kupno Programowanej Stacji Przygotowania Danych PSPD-90. Wybór nie był przypadkowy; wpływ na decyzję — oprócz parametrów użytkowych — miała bliska współpraca z producentem, MERA KFAP w Krakowie. Równocześnie rozpoczęto instalację tych urządzeń u naszych klientów.

Stacja PSPD-90 pracuje ponadto jako urządzenie zewnętrzne systemu ODRA-1305. Stacja jest podłączona do sterownika CDT-325 i symulowana jako czytnik (perforator taśmy papierowej), umożliwiając obustronną transmisję.

Prezentowane poniżej artykuły pracowników przedsiębiorstwa poruszają niektóre problemy związane ze specyfiką naszej działalności, dużego ośrodka obliczeniowego ze zróżnicowanym parkiem maszyn i różnorodnymi wymogami użytkowników.

Sierpień 1983 r.

HENRYK RAJCHEL
Dyrektor KPIPB ETOB

-32 pojemność taśmy magnetycznej wykorzystywana jest w sposób mało efektywny; opracowano program SDMT dokonujący agregacji wielu zbiorów w dowolnej strukturze (zbiory proste, złożone) i umożliwiający ich odtwarzanie w postaci pierwotnej.

- Opracowano program SRMH, spełniający funkcje firmowego programu XRMH, lecz o zwiększonych możliwościach, usuwając przy tym niektóre jego niedogodności. Program SRMH umożliwia opcjonalne wykorzystanie dwóch czytników kart oraz eliminuje niepoprawne zakończenie pracy programu, spowodowane m.in. przez błędy pamięci taśmowej.

- Pewną lukę w oprogramowaniu firmowym dla pamięci dyskowych wypełniają opracowane programy DYSK i MAPA. Program DYSK służy do optymalizacji pamięci dyskowej służącej do zapisu biblioteki programów. Średniej wielkości bibliotekę programów można skrócić po przebiegu tego programu o 15—30%. Program MAPA natomiast drukuje mapę zbioru złożonego, podobną do mapy pakietu drukowanej przez program XPJD. Ponadto umożliwia on relokację zbiorów oraz odtwarzanie zbiorów z pakietów uszkodzonych.

- Opracowano programy PLAN i FRED dla programistów systemowych i korzystających z języka PLAN. Program PLAN przeznaczony jest do drukowania (w postaci instrukcji języka PLAN) programów binarnych oraz testów technicznych zapisanych na taśmie magnetycznej lub dysku. Program FRED umożliwia śledzenie wykonywania programów binarnych z szeregiem dodatkowych możliwości, np. drukowanie podczas śledzenia wybranych obszarów programu czy ograniczenie śledzenia do określonych instrukcji.

W początkowym okresie eksploatacji komputera ODRA-1305 w celu usprawnienia jego wykorzystania, a jednocześnie z powodu niemożności zainstalowania systemu operacyjnego GEORGE-3 (a nawet GEORGE-2), kupiono w ZETO Gdynia pakiet AFRODYTA. Korzystając z koncepcji tego pakietu opracowano w przedsiębiorstwie minisystem operacyjny ADAM, realizujący prawie wszystkie funkcje systemu GEORGE-2, lecz pracujący w trybie konwersacyjnym i mający dużo większe możliwości niż wspomniany pakiet AFRODYTA.

*

Nieco inaczej przedstawiała się sytuacja w przedsiębiorstwie w momencie wdrażania do eksploatacji systemu R-32. W tym czasie eksploatowano, jak wiadomo, równocześnie dwa systemy MIŃSK-32 oraz system ODRA-1305. Warunki były jednak o tyle korzystne, że wiele ośrodków w kraju miało już bogate doświadczenia i dorobek w zakresie użytkowania komputerów Jednolitego Systemu.

Jakie podjęto zadania? Przede wszystkim kupiono w ZETO Gdynia symulator ODRA-1305 na R-32. Umożliwiło to złagodzenie okresu przejścia z ODRY na system RIAD. Sprzyjającą okolicznością był też fakt, że w sieci przedsiębiorstw ETOB zwrócono baczniejszą uwagę na problem koordynacji i wspólnego finansowania ważniejszych prac w zakresie oprogramowania narzędziowego. Wiodące w tej problematyce, katowickie przedsiębiorstwo ETOB, opracowało wiele bardzo interesujących programów. Szczególnie przydatne okazały się:

- IEBCNVOR i IEBCNVRO — programy konwersji zbiorów danych (również programów źródłowych) między komputerami ODRA i RIAD, charakteryzujące się dużą uniwersalnością i większymi możliwościami niż programy o zbliżonym charakterze, oferowane przez ELWRO czy ZETO Poznań
- IEBCNVMR — program konwersji zbiorów danych z MIŃSK-32 na R-32
- IEBPRINT — uniwersalny program tworzenia wydawnictw.

W praktyce eksploatacyjnej komputera ODRA-1305 wykorzystywany jest szeroko program GAMP (X68A) z pakietu SOD. Brak tego typu oprogramowania w systemie R-32/OS stał się dla nas inspiracją do opracowania programu o tej samej nazwie, realizującego prawie wszystkie funkcje programu GAMP. Z podobnych względów, korzystając z doświadczeń eksploatacji na komputerze ODRA, opracowano w systemie RIAD program

ISZERRTM, realizujący funkcje odtwarzania zbiorów z uszkodzonych taśm magnetycznych.

Współpraca naszego przedsiębiorstwa z innymi ośrodkami wykorzystującymi sprzęt minikomputerowy skłoniła nas m.in. do opracowania programów dwustronnej konwersji między komputerem R-32 a minikomputerem VARIAN-73.

*

Podobnie jak komputery, również zainstalowane w przedsiębiorstwie urządzenia minikomputerowe są bardzo niejednorodne. Możliwości użytkowania systemu MERA 9150 zostały poszerzone poprzez kupno oprogramowania w Centrum Informatyki Gospodarki Morskiej. Pozwala to na wykorzystywanie tych urządzeń nie tylko do wprowadzania danych, lecz również do tworzenia nieskomplikowanych autonomicznych systemów przetwarzania danych oraz m.in. do zapisu i edycji programów źródłowych w różnych językach dla komputerów ODRA-1300 czy R-32.

Z inicjatywy służb technologicznych podjęto realizację interesującego wniosku racjonalizatorskiego, umożliwiającego przyłączenie drukarki systemu ODRA-1300 w miejsce znacznie wolniejszej drukarki mozaikowej DZM-180, co w połączeniu z przygotowywanym oprogramowaniem pozwoli m.in. na rozładowywanie taśm magnetycznych z zapisem wydruków z komputerów ODRA i RIAD.

Zainstalowane minikomputery PSPD--90 i MERA 100 pracują nie tylko autonomicznie, ale również są podłączone do systemu ODRA-1305 jako urządzenia wejścia-wyjścia, symulowane jako czytnik i perforator taśmy papierowej. Szczególną uwagę poświęcono systemowi PSPD-90 przede wszystkim z tego powodu, iż jest to minikomputer bardzo rozpowszechniony na terenie objętym działalnością naszego przedsiębiorstwa. Opracowano tu wiele narzędzi programistycznych, usprawniających eksploatację i tworzenie programów użytkowych. Są to m.in.:

- MANIPULANT — program umożliwiający kopiowanie i edycję dowolnych fragmentów pamięci i zapisów na dyskietkach
- GENERATOR — program generujący z programu binarnego samodzielną dyskietkę systemową
- program sortujący na dyskietkach
- biblioteka podprogramów realizujących wiele funkcji pomocniczych, np. konwersja różnych postaci danych, operacje na łańcuchach, operacje na dyskietkach.

*

Przedstawione przedsięwzięcia w zakresie oprogramowania narzędziowego nie wyczerpują oczywiście tematyki i zagadnień, którymi zajmowało się przedsiębiorstwo. Zasygnalizowane zostały jedynie główne problemy. W tej mierze można by sformułować następujące wnioski:

- niejednorodność zainstalowanego parku maszyn i urządzeń powoduje konieczność utrzymywania zbyt dużych służb konserwatorskich i technologicznych
- rozpoczęciu produkcji jakościowo nowego sprzętu musi towarzyszyć równoczesne tworzenie środków techniczno-programowych łagodzących proces zmiany sprzętu
- duże znaczenie ma właściwe przygotowanie przedsiębiorstwa przed zainstalowaniem komputera; w okresie tym należy wnikliwie ocenić dostępne oprogramowanie (jak wskazuje praktyka, producent nie dysponuje, niestety, wyczerpującymi informacjami na ten temat)
- wiele prac jest niepotrzebnie dublowanych w poszczególnych ośrodkach, wynika to między innymi z niedostatecznego przepływu informacji.

ZBIGNIEW NOWAK
JANUSZ RUDAWSKI

Przyłączenie drukarki systemu ODRA-1300 do MERY 9150

Wysokie ceny oraz niewielki wybór dostępnego na rynku sprzętu komputerowego sprawiają, że opłacalne jest wyszukiwanie niekonwencjonalnych zastosowań posiadanego sprzętu. Spośród parku maszynowego krakowskiego ETOBU największe możliwości nietypowego wykorzystania ma system MERA 9150, przeznaczony do wprowadzania danych z wielu klawiatur na taśmę magnetyczną. W zastosowaniu tym dwa posiadane systemy tego typu pracują w naszym przedsiębiorstwie tylko na jedną, a sporadycznie na dwie zmiany.

Sprzęt tego systemu stanowią: uniwersalny minikomputer NOVA 2000, stały dysk magnetyczny, pamięć taśmowa PT 105, 32 konsole operatorskie oraz drukarka DZM-180. Sprzęt ten może być z powodzeniem wykorzystany podczas drugiej i trzeciej zmiany do wykonywania wielu drobnych zadań odciążających duże komputery, np. do wydrukowania zawartości taśm magnetycznych, sformatowanych na ODRZE lub RIADZIE. Umożliwiające taką operację oprogramowanie zostało już w znacznej mierze opracowane. Najistotniejszą przeszkodą jest w tym przypadku brak w systemie MERA 9150 szybkiej drukarki.

Przedsiębiorstwo posiada nie wykorzystywaną obecnie drukarkę wierszową DW-312 systemu ODRA. Drukarka ta, zbyt wolna do współpracy z dużym komputerem, jest jednak kilkakrotnie szybsza od DZM-180. Aby umożliwić zastosowanie jej w systemie MERA 9150, autor niniejszego artykułu opracował urządzenie o roboczej nazwie MEROD, które pośredniczy w transmisji danych między minikomputerem i drukarką. Od strony drukarki symuluje ono kanał maszyny cyfrowej ODRA serii 1300, natomiast od strony minikomputera — zastosowano w nim wersję drukarki DZM-180.

Kwalifikatory przesyłane do drukarki są przygotowywane w zależności od znaku sterującego poprzedzającego wysłanie rozkazu PISZ. Urządzenie interpretuje trzy znaki sterujące: CR, LF i FF. Pozostałe znaki sterujące są ignorowane i drukowane jako spacje, nie powodując przesłania do drukarki nowego rozkazu PISZ. W pierwszym cyklu po włączeniu zasilania jest przesyłany kwalifikator odpowiadający znakowi sterującemu CR. Jeżeli do drukarki zostanie przesłane więcej niż 120 znaków nie zawierających CR, LF ani FF (dla DW-312, która ma wiersz 120-znakowy), to znaki 121 i następne będą nabijane na pierwszych pozycjach tego samego wiersza.

Względnie duża złożoność układu jest spowodowana dużą różnicą w poziomie organizacji logicznej standardowego interfejsu ODRY i niestandardowego kabla drukarki MERY 9150. Układ został zbudowany z powszechnie dostępnych obwodów scalonych TTL serii UCY 7400. W obecnej wersji zawiera on 36 obwodów scalonych oraz dwa typowe pakiety nadajników i odbiorników interfejsu systemu ODRA. Układ jest przewidziany do wbudowania w obudowę minikomputera i wykorzystuje jego zasilacze. Wymaga napięć zasilających +5 V, +15 V, —5 V, wszystkie o poborze prądu 0,5—0,7 A.

Podczas projektowania oraz wykonywania prototypu zwracano baczną uwagę, aby niepotrzebnie nie ograniczać urządzenia do współpracy tylko z DW-312 lub tylko z określoną wersją minikomputera. W efekcie opracowana konstrukcja umożliwia podłączenie dowolnej drukarki systemu ODRA-1300 (również szybkich drukarek, np. DW-325) do systemu MERA 9150, a po dobudowaniu nieskomplikowanego adaptera (dalsze cztery obwody scalone) — do dowolnego systemu zawierającego DZM-180.

PAWEŁ GRABSKI

cznych mikroprogramy testowania i monitorowania, natomiast w warunkach normalnej eksploatacji w pamięci tej przechowywany jest system rozkazów kanałowych USU, rejestry sterowania i inne informacje systemowe.

Jako konsekwencję wirtualnej organizacji pamięci wprowadzono także pośrednie adresowanie danych w kanale, które pozwala na przekształcenie adresów wirtualnych (logicznych — występujących podczas przesyłania danych między pamięcią a urządzeniami zewnętrznymi) na adresy rzeczywiste.

Ciekawą zapewne informacją dla użytkowników komputerów MINSK-32 jest fakt oferowania przez producenta licznych środków programowych i technicznych zapewniających pełną wymienność z tym komputerem.

Cechą charakterystyczną omawianego modelu są stosunkowo małe wymagania dotyczące warunków pracy. Producent gwarantuje prawidłową pracę komputera w temperaturze 5°—40°C oraz przy wilgotności względnej od 40 do 95%. Gorsze niestety są parametry szybkości pracy zarówno procesora jak i poszczególnych kanałów w porównaniu z odpowiednimi parametrami polskiego komputera EC 1032 (RIAD I).

Nowy komputer wyposażony został w bogate oprogramowanie techniczne oraz nowoczesne oprogramowanie systemowe.

W skład oprogramowania technicznego wchodzą:

— pakiet testów diagnozy mikroprogramowej (KMD), służący do wykrywania i lokalizacji błędów w poszczególnych blokach logicznych procesora

— TEST-MONITOR — autonomiczny, zunifikowany system testów sprawdzających prawidłowość pracy całej instalacji EC 1035-01

— zespół nieautonomicznych testów urządzeń zewnętrznych, pozwalających sprawdzić poprawność pracy urządzeń peryferyjnych bez przerywania pracy całego systemu komputerowego.

W skład oprogramowania systemowego wchodzą:

— system operacyjny OS/VS

— system programowania i kompilacji

— pakiety oprogramowania aplikacyjnego — całkowicie zgodne z odpowiednim oprogramowaniem komputerów serii 370.

Oprac. RYSZARD GRZESIAK
na podstawie
„Rechentechnik Datenverarbeitung"
nr 3/83

Rozwój mikroelektroniki na Węgrzech

Obecnie na Węgrzech produkuje się 30 mln tranzystorów i 15 mln układów scalonych rocznie, jednak — choć są one znanym w świecie eksporterem w dziedzinie elektroniki — stale pogłębia się zależność od importu. W celu zmiany tego uzależnienia, rząd zdecydował w grudniu 1981 r. o nadaniu najwyższego priorytetu rozwojowi przemysłu elektronicznego, przeznaczając na te działania 175 mln dol. Większość tej sumy dotyczy prac badawczo-rozwojowych oraz kupna urządzeń i technologii z dziedziny mikroelektroniki.

Pierwszym krokiem w realizacji programu było połączenie Instytutu Badawczego Przemysłu Telekomunikacyjnego z fabryką półprzewodników TUNGSRAM. Nowe Przedsiębiorstwo Mikroelektroniczne zatrudnia 4300 pracowników, wytwarzając układy VLSI, maski, układy scalone (na zamówienie) oraz wykonuje prace projektowe; roczna wartość produkcji i usług — 40 mln dol. W tych dziedzinach będzie ono monopolistą i nie wynika to z jakiś arbitralnych decyzji, ale z faktu, że tak małego kraju nie stać na rozpraszanie środków. W sumie program będzie dotyczył 24 zakładów, zatrudniających 160 tys. osób łącznie.

Władze węgierskie podkreślają, że celem tego przedsięwzięcia nie jest osiągnięcie samowystarczalności w podzespołach elektronicznych, ale uzyskanie rozsądnej równowagi importu i eksportu. Przewiduje się specjalizację w produkcji projektowanych na zamówienie układów w seriach od kilkuset do kilku tysięcy sztuk oraz sprowadzanie masowo wytwarzanych układów uniwersalnych. Przyjęcie tego kierunku wynika z konkurencyjności węgierskich opracowań zawierających duży udział oprogramowania — dzięki jego wysokiej jakości i relatywnie niskim kosztom prac programistycznych w tym kraju.

Przypomnijmy, że w roku ubiegłym Węgry wyeksportowały do Europy Zachodniej oprogramowanie wartości 5 mln dol., a wtedy program rozwoju w zasadzie jeszcze nie wystartował. Już w 1985 roku przewiduje się eksport 30% produkcji, głównie na rynki zachodnie, a także specjalizację w dostawach układów na zamówienie krajów socjalistycznych. Z oczywistych przyczyn ekonomicznych, Węgrzy nie mogą rozwijać swojego przemysłu sami, wykorzystują więc wspólne plany w ramach RWPG, nastawiając się na szczególnie ścisłą współpracę z ZSRR, NRD i CSRS. Jak więc widać, możemy pozostać już nie tylko poza nurtem światowego rozwoju elektroniki, ale nawet poza najważniejszymi przedsięwzięciami krajów naszego obozu.

Oprac. MkS
na podstawie
ELECTRONICS z 31.05.1983

KONFERENCJE

Przestępczość komputerowa

O wadze problemu przestępczości komputerowej w krajach rozwiniętej informatyki świadczy fakt, że w dniach 17—18 listopada br. w Paryżu odbyło się już piąte, kolejne międzynarodowe seminarium na ten temat. Organizatorem seminarium był wspomniany już wiele razy na naszych łamach paryski instytut badawczy bankowości INSIG, któremu udało się pozyskać jako głównego prelegenta D. B. Parkera z Uniwersytetu Stanford, znanego amerykańskiego specjalistę i autora wielu publikacji na temat tego typu przestępstw.

W zapowiedzi seminarium akcentuje się szybki ilościowy wzrost przestępstw komputerowych oraz stosowanie przez ich sprawców metod coraz trudniejszych do wykrycia. Głównym akcentem programu seminarium były nowe, bardziej skuteczne sposoby prewencji.

Słownictwo z zakresu inżynierii oprogramowania

Po dwumiesięcznej przerwie powracamy do omawiania terminologii zawartej w normie „IEEE Standard Glossary of Software Engineering Terminology". Ważną grupę terminów z tego dokumentu stanowią pojęcia związane z niezawodnością oprogramowania. Choć zagadnienia te omawialiśmy częściowo jeszcze przed opublikowaniem normy (INFORMATYKA nr 8—9/1982), są na tyle ważne, że warto do nich powrócić.

Podstawowym terminem związanym z niezawodnością jest **błąd.** Jednakże podana w tym dokumencie (za ISO 2382/II) definicja określająca błąd (ang. error) jako rozbieżność między obliczoną, zaobserwowaną lub zmierzoną wartością (stanem) a wartością (stanem) prawdziwą, zdefiniowaną lub teoretycznie poprawną, ma niewiele wspólnego z programowaniem. Inna defnicja z tej samej normy (również analogiczna do ISO 2382/XIV) stwierdza, że błąd jest to niezamierzony stan powodujący, że jednostka funkcjonalna zaprzestaje wykonywania wymaganych funkcji. Tak rozumiany błąd, nazywany po angielsku fault (często też bug lub defect), a po francusku défaut, bardziej odpowiada poszukiwanemu przez nas znaczeniu; choć może tkwić zarówno w sprzęcie, jak i w oprogramowaniu.

Oczywiście, nie jest korzystne, aby na oznaczenie dwóch zupełnie różnych pojęć w jednej dziedzinie (error i fault) stosować tę samą nazwę — **błąd.** Jest to zresztą dostrzegane. Coraz częściej w literaturze polskojęzycznej błąd w znaczeniu bliższym programowaniu nazywa się **defektem,** co jest — jak łatwo się przekonać — zgodne ze źródłosłowem angielskim i francuskim. Choć nie należy, z kolei, mnożyć nazw na oznaczenie tych samych pojęć, w tym przypadku byłbym za używaniem polskiego terminu **wada** lub **usterka.**

Na tym jednak nie kończą się kłopoty z nazywaniem błędów. Jeszcze innym rodzajem błędu (ang. mistake) nazywa się działanie ludzkie prowadzące do niepożądanego wyniku. W tym przypadku odpowiednikiem polskim może być termin **pomyłka.**

Defekt (ang. fault), jeżeli wystąpi, może spowodować awarię (ang. failure). Z kolei **awarią (uszkodzeniem)** w wymienionej normie (również w normie ISO 2382/XIV) nazywa się ograniczenie zdolności jednostki funkcjonalnej do wykonywania wymaganych funkcji.

Na co dzień każdy programista jest najbardziej oswojony z usuwaniem defektów w programie, tj. z uruchamianiem programu. W wymienionej normie **uruchamianiem** (ang. debugging) nazywa się proces umiejscowiania, analizowania i poprawiania przypuszczalnych defektów. Ciekawe, że po uwzględnieniu tego, co napisano powyżej o błędach oraz — definicji uszkodzenia, powyższa definicja uruchamiania nie różni się od podanej przez M. T. Jankowskiego (INFORMATYKA nr 2/1983).

Nie można tego natomiast powiedzieć o terminie **testowanie,** którego definicję, według projektu normy ISO, podano w numerze 8—9/1982. W normie IEEE nazewnictwo związane z tym terminem jest o wiele bardziej rozbudowane. W projekcie normy ISO 2382/VIII przez **testowanie** rozumie się — wykonywanie programu dla określonego zbioru danych w celu uzyskania przewidzianego wyniku, będącego podstawą do przyjęcia programu. Natomiast według normy IEEE — testowanie jest to proces ręcznego lub automatycznego sprawdzania i oceny systemu lub jego składnika w celu zweryfikowania, czy spełnia on narzucone wymagania lub — w celu określenia różnic między oczekiwanymi i rzeczywistymi wynikami.

Ponieważ ostatecznym celem testowania jest zawsze doprowadzenie do przyjęcia wyrobu, w normie IEEE zdefiniowano szereg innych pojęć z tego zakresu.

Testowaniem systemu (ang. sytem testing) nazywa się proces łącznego testowania sprzętu i oprogramowania w celu zweryfikowania, czy spełnia on narzucone wymagania.

Przez **testowanie formalne** (ang. formal testing) rozumie się proces przeprowadzania działań testujących i podawania wyników zgodnie z zatwierdzonym planem (tj. specjalnym dokumentem, ang. test plan). **Testowanie kwalifikacyjne** (ang. qualification testing) jest to testowanie formalne, przeprowadzone zwykle przez wytwórcę dla użytkownika, a mające na celu wykazanie, że oprogramowanie spełnia narzucone wymagania. **Testowanie akceptacyjne** (ang. acceptance testing) jest to testowanie formalne, przeprowadzane w celu określenia, czy system spełnia kryteria odbioru (ang. acceptance criteria) i pozwalające użytkownikowi przyjąć system.

Na zakończenie podamy jeszcze, jak rozumie się w normie IEEE-729 pozostałe pojęcia omówione w numerze 8—9/1982.

Weryfikacją nazywa się proces określania, czy w danej fazie cyklu rozwojowego oprogramowania wyrób spełnia wymagania ustalone w poprzedniej fazie. Natomiast **uwiarygodnieniem** (ang. validation) nazywa się proces oceny oprogramowania na końcu procesu (cyklu) rozwojowego w celu stwierdzenia zgodności z wymaganiami. **Niezależna weryfikacja i uwiarygodnienie** (ang. independent verification and validation) jest to weryfikacja i uwiarygodnienie wyrobu (oprogramowania) przeprowadzona przez instytucję technicznie i organizacyjnie niezależną od instytucji odpowiedzialnej za opracowanie wyrobu. I wreszcie — **zatwierdzenie** (ang. certificcation) jest to pisemna gwarancja, że system lub program komputerowy spełnia narzucone wymagania.

Do wymienionych określeń warto dodać dwie uwagi. Po pierwsze — niektóre występujące tu pojęcia, np. acceptance criteria, test plan, software development process czy software development cycle, zostały oddzielnie zdefiniowane. Po drugie — wiele podanych tu definicji stanowi jedynie podstawowe znaczenie opisywanego terminu, jako że istnieje ich więcej. Nie zostały tutaj przywołane żeby nie wprowadzać większego zamętu.

JANUSZ ZALEWSKI

SAMOKLAP

Propozycja Pana Drwinżyniera (INFORMATYKA nr 4/83, s. 38), aby wyraz **komputer** zastąpić słowem SAMOLICZ budzi poważne zastrzeżenia. Wprawdzie określenie SAMO- jest przyjęte w naszym języku i dobrze się kojarzy z innymi pojęciami, z wyjątkiem wyrazu **samotrzeć,** który nasza młodzież niesłusznie utożsamia z grzechem biblijnego Onana, ale SAMOLICZ jest zwyczajną próbą przemycenia do języka — nadziei; typowych dla epoki propagandy sukcesu. Dziś komputery nie liczą, bo ich już prawie nie ma, a i n a n i e też nikt nie liczy! Generalnie rzecz biorąc polska informatyka ponosi klapę i dlatego proponuję, aby wyraz komputer zastąpić raczej słowem SAMOKLAP. W ten sposób w naszej terminologii znajdzie trwałe odbicie nie mniej trwały kryzys naszej branży.

Przy sposobności proponuję, aby nieznane u nas jeszcze urządzenie WORD PROCESSOR ochrzcić słowem SAMO-PIS, co byłoby przydatne w tłumaczeniach.

ODROSŁAW RIADOMĘCKI
(adres nie znany redakcji)

Błaszczak S., Iszkowski W.: Język ADA w testach

INFORMATYKA 1983, nr 11, s. 2

Opis zestawu ćwiczeń umożliwiających samodzielne sprawdzenie zrozumienia tych szczegółów konstrukcji języka ADA, które nie występują w innych językach programowania.

Блащак С., Ишковски В.: Язык ADA в контрольных упражнениях

INFORMATYKA 1983, № 11, стр. 2

Описание набора упражнений, позволяющих самостоятельно проверять понимание тех подробностей конструкций языка ADA, которые не выступают в других языках программирования.

Grabowski J.: Porównanie właściwości mikroprocesorów 16-bitowych (1)

INFORMATYKA 1983, nr 11, s. 6

Porównanie mikroprocesorów ZILOG 8001/2, INTEL 8086, MOTOROLA 68000, TEXAS INSTRUMENTS 9900, LSI 11 i NATIONAL SEMICONDUCTOR 16032 z punktu widzenia architektury i listy rozkazów, z podkreśleniem cech interesujących programistę.

Грабовски Й.: Сравнение свойств 16-битовых микропроцессоров (1)

INFORMATYKA 1983, № 11, стр. 6

Сравнение микропроцессоров ZILOG 8001/2, INTEL 8086, MOTOROLA 68000, TEXAS INSTRUMENTS 9900, LSI 11 и NATIONAL SEMICONDUCTOR 16032 с точки зрения архитектуры и набора команд. Подчеркнуты свойства, интересующие программиста.

Przemysłowy FORTRAN czasu rzeczywistego. Część 2

INFORMATYKA 1983, nr 11, s. 10

Druga część charakterystyki projektu międzynarodowej normy rozszerzającej zastosowanie języka FORTRAN, zawierająca szczegółowe omówienie wywołań procedur funkcyjnych i niefunkcyjnych.

Промышленный FORTRAN реального времени. Часть 2

INFORMATYKA 1983, № 11, стр. 10

Вторая часть характеристики проекта международного стандарта, расширяющего применение языка FORTRAN, содержащая подробное обсуждение вызовов функций и процедур.

Warski W.: Alternatywa dla struktury blokowej w językach czasu rzeczywistego

INFORMATYKA 1983, nr 11, s. 14

Omówienie wad struktury blokowej, stanowiącej podstawę budowy obecnie stosowanych języków czasu rzeczywistego. Zaprezentowano koncepcję rozwiązania upraszczającego konstrukcję i działanie tej kategorii języków programowania.

Варски В.: Альтернатива для блочной структуры в языках реального времени

INFORMATYKA 1983, № 11, стр. 14

Обсужение недостатков блочной структуры, являющейся основой строения теперь применяемых языков реального времени. Представлена концепция решения, упрощающая конструкцию и действие этой категории языков программирования.

Stasiewicz P., Stępaniec J.: Automatyka zdalnej obsługi użytkownika w systemie operacyjnym GEORGE-3

INFORMATYKA 1983, nr 11, s. 17

Prezentacja rozwiązań programowych zrealizowanych w Warszawskim Przedsiębiorstwie Informatyki Przemysłu Budowlanego „ETOB", zapewniających automatyzację zdalnej obsługi użytkownika oraz zwiększających efektywność wykorzystania systemu komputerowego ODRA 1305.

Стасевич П., Степанец Й.: Автоматизация дистанционного обслуживания пользователя в операционной системе GEORGE-3

INFORMATYKA 1983, № 11, стр. 17

Представление программных решений, осуществлённых в Варшавском Предприятии Вычислительной Техники Строительной Промышленности „ETOB", обеспечивающих автоматизацию дистанционного обслуживания пользователя и увеличивающих эффективность использования вычислительной системы ODRA 1305.

Abramowicz A. W.: System automatycznego redagowania tekstów literackich

INFORMATYKA 1983, nr 11, s. 20

Charakterystyka systemu automatycznego redagowania tekstów SART, zaprojektowanego w Instytucie Maszyn Matematycznych w Warszawie na potrzeby przemysłu poligraficznego. Omówiono rozwiązania wersji pilotowej systemu, przeznaczonej do składania tekstów literackich.

Абрамович А. В.: Система автоматического редактирования литературных текстов

INFORMATYKA 1983, № 11, стр. 20

Характеристика системы автоматического редактирования текстов SART, запроектированной в Институте Математических Машин в Варшаве для потребностей полиграфической промышленности. Обсуждаются решения ведущей версии системы, предназначенной для составления литературных текстов.

Błaszczak S., Iszkowski W.: ADA-language in tests

INFORMATYKA 1983, No. 11, p. 2

Description of exercises for selfchecking of understanding such ADA-language construction details, which in other programming languages not exist.

Błaszczak S., Iszkowski W.: ADA-Sprache in Testen

INFORMATYKA 1983, Nr. 11, S. 2

Eine Beschreibung der Übungen, die eine Selbstprüfung von Verständnis solcher Konstruktionseinzelheiten der ADA-Sprache, die in anderen Programmiersprachen nicht bestehen, ermöglicht.

Grabowski J.: 16-bit-microprocessor features comparison (1)

INFORMATYKA 1983, No. 11, p. 6

Comparison of ZILOG 8001/2, INTEL 8086, MOTOROLA 68000, TEXAS INSTRUMENTS 9900, LSI 11 and NATIONAL SEMICONDUCTOR 16032 microprocessors from architecture's and instruction list's point of view with emphasis on features, which are interesting for programmer.

Grabowski J.: Eigenschaftenvergleich von 16-Bits-Mikroprozessoren (1)

INFORMATYKA 1983, Nr. 11, S. 6

Ein Vergleich der Mikroprozessoren ZILOG 8001/2, INTEL 8086, MOTOROLA 6.000, TEXAS INSTRUMENTS 9900, LSI 11 und NATIONAL SEMICONDUCTOR 16032 von dem Standpunkt der Architektur und Befehlsliste aus, mit Betonung der für Programmierer bedutenden Eigenschaften.

Real-time FORTRAN for industrial applications. Part 2

INFORMATYKA 1983, No. 11, p. 10

Second part of international standard proposal characteristics for real-time FORTRAN, which includes detailed discussion of functional and nonfunctional procedure calls.

Echtzeit-FORTRAN für industrielle Awendungen. Teil 2

INFORMATYKA 1983, Nr. 11, S. 10

Zweiter Teil der Charakteristik eines internationalen Normentwurfes für Echtzeit-FORTRAN Erweiterung, der genaue Besprechung der funktionellen und nicht funktionellen Prozedurabrufe umfasst.

Warski W.: Alternative for the block structure in real-time languages

INFORMATYKA 1983, No. 11, p. 14

Disadvantages of the block structure, which is a base for construction of to-day applied real-time languages, are discussed. Solution idea for construction and work simplification of such programming language category is presented.

Warski W.: Eine Alternative für die Blockstruktur in Echtzeitprogrammiersprachen

INFORMATYKA 1983, Nr. 11, S. 14

Nachteile der Blockstruktur, die eine Grundlage für den Bau der gegenwärtig angewendeten Echtzeitprogrammiersprachen bilden. Es wurde die Konzeption einer Lösung vorgestellt, die die Struktur und Wirkung solcher Programmiersprachen vereinfacht.

Stasiewicz P., Stepaniec J.: Automatic user's remote access in GEORGE-3 operating system

INFORMATYKA 1983, No. 11, p. 17

Presentation of software solutions realized in Warsaw Data Processing Enterprise of Building Industry ETOB, which ensure automation of user's remote access and improve operation effectivity of the ODRA 1305 computer system.

Stasiewicz P., Stepaniec J.: Automatisierung des Benutzersfernzugriffes in GEORGE-3 Betriebssystem

INFORMATYKA 1983, Nr. 11, S. 17

Eine Vorstellung der in Warschauer EDV-Unternehmen für Bauindustrie ETOB realisierten Softwarelösungen, die den automatisierten Fernzugriff für Benutzer sichern und Ausnutzungseffektivität des ODRA 1305 Rechnersystems verbessern.

Abramowicz A. W.: A system for automatic editing of literary text

INFORMATYKA 1983, No. 11, p. 20

Characteristics of the SART automatic editing system, which was designed in the Institute of Mathematic Machines in Warsaw for printing industry needs. Solutions of the pilot version of the system intended for literary typesetting are discussed.

Abramowicz A. W.: Ein System für automatisiertes Redigieren der literarischen Texte

INFORMATYKA 1983, Nr. 11, S. 20

Eine Charakteristik des automatisierten Systems SART, das im Institut für Mathematische Maschinen in Warschau für Bedürfnisse der polygraphischen Industrie entworfen wurde. Es wurden Lösungen einer Pilotversion für Drucksatz der literarischen Texte besprochen.

■ Raport przygotowany przez BOOZ, ALLEN AND HAMILTON (nowojorską firmę badania rynku) przewiduje, że w połowie lat dziewięćdziesiątych 30 mln gospodarstw domowych w USA będzie korzystało z systemów informacyjnych, do których dostęp zapewnią komputery osobiste. Proponowany zakres usług obejmie rezerwację, zakupy, zarządzanie budżetem domowym, gry, kalendarze, nauczanie czy wspomaganie operacji finansowych oraz wyszukiwanie informacji. Według ocen zaprezentowanych przez Jamesa Farleya, szefa firmy, 60% wszystkich domowych wydatków dokonywać się będzie za pośrednictwem tych systemów. Obecnie będą wykorzystane dwie metody ich realizacji — za pomocą sieci kablowej albo telefonicznej. Eksperci z BOOZ, ALLEN AND HAMILTON twierdzą, że brak standardów w tej dziedzinie nie wniesie istotnego opóźnienia w rozwoju. Przewidywany koszt budowy — 150 mld dol. przy rocznym rynku usług — 30 mld.

*

■ Być może walka o panowanie na rynku systemów operacyjnych dla mikrokomputerów 16-bitowych (przypominamy, że dla 8-bitowych panuje niepodzielnie CP/M) rozegra się nie poprzez zwycięstwo jednego z konkurentów, lecz poprzez zacieranie różnic. Świadczy o tym nowa wersja 2.0 znanego systemu MS-DOS firmy MICROSOFT, która pozwala, by programy napisane pod jego kontrolą były wykonywane pod kontrolą systemu XENIX (czyli wersji sławnego UNIXA). Oczywiście wymagało to zastosowania hierarchicznej struktury zbiorów oraz mechanizmów wejścia/wyjścia niezależnych od urządzeń peryferyjnych. W niedalekiej przyszłości MICROSOFT opracuje wspólną bibliotekę funkcji dla obu systemów oraz wzbogaci MS-DOS o możliwość pracy wieloprogramowej. MS-DOS 2.0 stanowi standardowe wyposażenie nowego modelu IBM PERSONAL COMPUTER XT.

*

■ Na wiosennej wystawie COMDEX 83 (26—29 kwietnia, Atlanta, USA) firma VERTIMAG SYSTEMS pokazała prototyp jednostki pamięci na dysku elastycznym, wykorzystującą nową technikę zapisu pionowego. Dzięki temu udało się uzyskać pojemność zapisu 5 M bajtów na dyskietce o średnicy 5,25 cala. Całe urządzenie składa się ze zmodyfikowanej jednostki firmy SHUGART oraz nośnika kobaltowo-chromowego na podłożu z tworzywa Mylar pozwalającego osiągnąć gęstość zapisu 30 tys. bitów/cal, czyli ok. 4 razy większą niż dotychczas. Uruchomienie produkcji w rozmiarach umożliwiających normalną sprzedaż jest oczekiwane w połowie roku 1984.

*

■ Znany japoński koncern NEC zaprezentował w końcu kwietnia dwa nowe superkomputery SX-1 i SX-2. Według danych firmowych, pierwszy z nich osiągnie prędkość 570 mln operacji zmiennoprzecinkowych na sekundę (flops), zaś drugi — przełamie barierę miliarda — 1,3 G flops. W komputerach tych zastosowano chłodzone cieczą pakiety o dużej gęstości elementów, 1000-bramkowe układy logiczne z opóźnieniem 250 ps/bramkę i statyczne pamięci RAM 64 K wykonane w technologii MOS z czasem dostępu 3,5 ns. Dzięki temu cykl maszynowy wynosi 6 ns. Przewiduje się, że komputery serii SX będą mogły mieć pamięć operacyjną rozszerzaną aż do 2 G bajtów. Pierwsze dostawy — na początku 1985 r.; opłata za dzierżawę SX-2 — 383 tys. dol. miesięcznie.

*

■ Radykalny postęp dotyczy nie tylko mini-dysków, lecz także urządzeń większych. Firma CDC wprowadziła na rynek nowy model jednostki z dyskami o średnicy 14 cali nazwany 9771 XMD. Urządzenie przy wymiarach zewnętrznych ok. 50×75×25 cm ma pojemność 825 M bajtów. W stosunku do poprzedniego modelu (9775 FMD, 675 MB) zwiększono gęstość promieniową zapisu z 800 ścieżek/cal i liniową z 6400 do 15 400 bitów/cal oraz zmniejszono objętość jednostki do jednej trzeciej, liczbę dysków z 12 do 5, czas dostępu z 25 do 16 ms, liczbę płytek z układami sterującymi z 27 do 4 oraz koszt z 22 do niecałych 12,5 dol. za 1 M bajt. Jednocześnie podniesiono istotnie poziom niezawodności — średni czas międzyawaryjny wynosi obecnie 11 tys. godzin, zamiast dotychczasowych 6 tys. Parametry te wsakzują, że dyski 9771 XMD będą mogły być stosowane z powodzeniem w mini-, a może nawet mikrokomputerach.

*

■ Od 12 do 17 września br. odbyła się w Warszawie jedna z narad specjalistów w ramach prac prowadzonych przez Radę ds. Zastosowań Środków Techniki Obliczeniowej MKETO. Organizatorem jej było Centrum Projektowania i Zastosowań Informatyki, a uczestniczyły w niej delegacje ZSRR (główny wykonawca), NRD, Bułgarii, CSRS oraz krajowych ośrodków. Spotkanie poświęcono uzgodnieniu założeń technicznych dla zautomatyzowanych systemów obróbki danych w dziedzinie finansów i ubezpieczeń społecznych. Wygłoszone referaty prezentowały rozwiązania opracowane w poszczególnych krajach, a dotyczące przede wszystkim baz danych i systemów zarządzania nimi ukierunkowanych na ten typ zastosowań. W ramach narady zorganizowano również praktyczny pokaz osiągnięć CPIZI w tej dziedzinie.

*

■ W roku 1983 zostaną po raz pierwszy przyznane dwie nowe nagrody naukowe dla autorów najwybitniejszych osiągnięć informatycznych — ufundowane przez Międzyrządowe Biuro Informatyki w Rzymie, znane pod skrótem IBI. Mniejszą z tych nagród — 5000 dol. — przyznaje hiszpańska delegatura IBI w Madrycie w dziedzinie zastosowań komputerów w szkolnictwie wyższym, opisanych (wymóg kategoryczny) w języku hiszpańskim. Intencją nagrody, która ma być przyznawana dorocznie przynajmniej przez pięć najbliższych lat, jest zwiększenie zainteresowania dydaktyką informatyki na uczelniach latyno-amerykańskich. Oprócz wspomnianej nagrody, przewidziano kilka 1000 — dolarowych wyróżnień. Natomiast „wielką' 'nagrodę IBI — 25 tys. dol. — będzie się przyznawać za szczególną działalność przyczyniającą się do upowszechniania informatyki i jej rozwoju. Utworzono już międzynarodowe jury, w skład którego wchodzi Generalny Dyrektor IBI oraz czterech przedstawicieli krajów członkowskich. Ogólny regulamin wielkiej nagrody uchwalono w listopadzie 1982, a na 48 sesji Ogólnego Zgromadzenia IBI, w kwietniu 1983 w Rabacie (Maroko), zatwierdzono warunki przyznawania tego wyróżnienia po raz pierwszy. Niewątpliwie pogoń za „informatycznym noblem" jest szlachetna, można jednak żywić obawy, iż Wielka Nagroda IBI będzie przyznawana wyłącznie przedstawicielom krajów członkowskich. PRL w IBI nie bierze udziału.

*

■ Trwa dalszy rozwój 16-bitowej wersji systemu operacyjnego CP/M — oczywiście, by sprostać wyzwaniu Laboratorium Bella (system UNIX) i firmy MICROSOFT (MS-DOS). DIGITAL RESEARCH, wspólnie z HITACHI (Japonia) opracowała ostatnio nowe mutacje dla procesów MOTOROLA 68000 oraz iAPX 286 (INTEL), zapewniając jednocześnie, że dzięki hierarchicznej strukturze zbiorów i wieloprogramowości z komunikacją między procesami — jest to nowa generacja systemów operacyjnych.

*

■ Od 19 do 23 września 1983 Paryż gościł najważniejszą światową konferencję w dziedzinie informatyki — Kongres Międzynarodowej Federacji Przetwarzania Informacji (IFIP). Organizowany co trzy lata, powrócił w tym roku do swego miejsca urodzenia — I Kongres, na którym powołano IFIP, odbył się również w Paryżu, w roku 1959. Inauguracji obrad w Pałacu Kongresów dokonał premier Francji Pierre Mauroy. Uczestniczyło w nim prawie 200 osobistości i ekspertów ze świata nauki i rozwoju informatyki. Program został podzielony na 10 głównych działów, a złożyło się nań 40 referatów zamówionych przez organizatorów oraz 98 (z 400 nadesłanych) komunikatów pochodzących z 21 krajów. Odbyły się także 32 dyskusje (tzw.okrągłe stoły) poświęcone syntezie najważniejszych aktualności informatyki. W najbardziej prestiżowej grupie osobistości zaproszonych do wygłoszenia referatu, znalazł się również profesor Andrzej Blikle z Instytutu Podstaw Informatyki PAN. Był on również w grupie dziewięciu ekspertów, wśród których francuska prasa fachowa przeprowadziła ankietę na temat przewidywanych kierunków rozwoju informatyki w ciągu najbliższych dziesięciu lat. Pośród zaproszonych referentów znaleźli się także m.in. J. D. Ichbiah (ADA), J. D. Ullman (teoretyczne podstawy baz danych), G. M. Amdahl (architektura superkomputerów), T. Moto-Oka (piąta generacja) oraz A. Jerszow (teoretyczne podstawy informatyki).

Zasady prenumeraty

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

Jednostki gospodarki uspołecznionej, instytucje i organizacje przesyłają zamówienia (w 1 egz.) zawierające: tytuł czasopisma, liczbę zamawianych egzemplarzy, okres prenumeraty i pełny adres zamawiającego z kodem pocztowym, a także numer konta bankowego zamawiającego oraz (ewentualnie) adresy odbiorców, którzy na zlecenie zamawiającego mają prasę otrzymywać.

Dopisując w zamówieniu PRENUMERATA STAŁA, zamawiający nie będzie musiał corocznie ponawiać zamówienia, a jedynie dokonywać przedpłaty według aktualnie obowiązujących cen.

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

Prenumeratorzy indywidualni dokonują wpłaty przekazem na ww. konto, podając na odwrocie odcinka dla adresata-posiadacza rachunku: tytuł czasopisma, liczbę zamawianych egzemplarzy oraz okres prenumeraty.

Zamówienia i wpłaty przyjmowane są w terminach:

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

Uwaga: przy podawaniu kodu pocztowego i numeru konta bankowego obowiązuje bardzo czytelne pismo.*

Prenumerata kwartalna — 225 zł, półroczna — 450 zł, roczna — 900 zł. Prenumerata ze zleceniem wysyłki za granicę jest dwukrotnie droższa.

Od 1 lipca br. wprowadzona została prenumerata ulgowa dla członków stowarzyszeń naukowo-technicznych NOT, studentów wyższych uczelni, uczniów szkół zawodowych. Warunkiem jej uzyskania jest poświadczenie blankietu dla nabywcy indywidualnego (na odcinku dla adresata) przez właściwe stowarzyszenie NOT, wyższą uczelnię lub szkołę zawodową. Cena prenumeraty ulgowej: kwartalnie — 150 zł, półrocznie — 300 zł, rocznie — 600 zł.

Dodatkowych informacji o prenumeracie udziela: Zakład Kolportażu, tel. 40-00-21 w. 293, 295, 217 oraz 40-35-89. Egzemplarze archiwalne można nabywać w Klubie Prasy i Informacji Technicznej w Warszawie, ul. Mazowiecka 12, tel. 27-43-65.

W numerze 12/83 przedstawimy

W najbliższych numerach:

- MODULA-2
- Interakcyjny system przetwarzania obrazów
- mikroKLAN — nowa wkładka dla „mikroamatorów"