4 1986

informatyka

Dr Brian Wichmann o konwersji oprogramowania

Procesor tekstowy dla Mery 400

Rekurencja w języku Forth

WYDAWNICTWO NOT SIGMA

Cena 120 zł

informatyka

4 1986

Nr 4

Miesięcznik Rok XXI

Kwiecień 1986

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

Materiałów nie zamówionych redakcja nie zwraca

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

Zakł. Graf. „Tamka". Zam. 0248-1300/86. Obj. 4,0 ark. druk. Nakład 8300 egz. P-85.

ISSN 0542-9951, INDEKS 36124

Cena egzemplarza 120 zł
Prenumerata roczna 1440 zł

WYDAWNICTWO NACZELNA ORGANIZACJA TECHNICZNA
SIGMA
CZASOPISM I KSIĄŻEK TECHNICZNYCH

00-950 Warszawa
skrytka pocztowa 1004
ul. Biała 4

W NUMERZE:

W NAJBLIŻSZYCH NUMERACH:

- Paweł Grzegorzewicz o mechanizmie półautomatycznego generowania poleceń dla systemu operacyjnego RSX-11M
- Piotr Zaskórski i Ewa Kasprzak o technologii wytwarzania oprogramowania użytkowego z komputerowym wspomaganiem
- Krzysztof Perycz o mikrokomputerowym systemie operacyjnym SI
- Andrzej Macioł i Adam Stawowy o środkach do przetwarzania tekstów
- Dines Bjørner o roli i zakresie formalnej definicji Ady
- Piotr Cofta o normie IEEE dla języków asemblerowych dla mikroprocesorów
- Krzysztof Michalik o programowaniu logicznym

BRIAN A. WICHMANN
J.G.J. MEIJERINK
National Physical Laboratory
Teddington, Middlesex
Wielka Brytania

Konwersja oprogramowania na język Ada

Biblioteka oprogramowania powinna zapewniać spełnienie możliwie szerokiego zakresu potrzeb użytkowników przez jak najmniejszą liczbę podprogramów. Rozwój i rozpowszechnienie biblioteki wymaga znacznego wysiłku, szczególnie pod względem dokumentacyjnym, i często potęguje trudności w doborze odpowiedniego podprogramu do określonego zastosowania. Kształt konkretnej biblioteki, jak np. NAG Fortran Library [6] (NAG, ang. Numerical Algorithms Group), jest ograniczony przez możliwości języka. Nie można zakładać więc, że optymalna postać podprogramu w Fortranie jest optymalna także dla innego języka.

W artykule tym rozważono konwersję krótkiego podprogramu z Fortranu i Pascala na Adę. Przykładowy podprogram generowania liczb losowych wybrano dlatego, że jest mały, a jednocześnie obejmuje wiele istotnych zagadnień spotykanych przy projektowaniu pakietów w Adzie. Nie musi być do tego potrzebna dokładna znajomość Ady, gdyż występujące problemy są względnie niezależne od języka, a poszczególne kroki konwersji wyjaśniono dość dokładnie.

KONWERSJA Z PASCALA NA ADĘ

Konwersji oprogramowania można dokonywać skutecznie bez dokładnej znajomości zastosowanych algorytmów. Zupełnie naturalna jest konwersja z języka wysokiego poziomu na kod maszynowy, przeprowadzana mechanicznie bez zrozumienia treści programów przez kompilatory. Poniższy przykład dotyczy generatora liczb losowych. Szczegóły jego implementacji omówiono w uprzednio opublikowanym artykule [8], a wyniki testowania losowości generowanych ciągów liczbowych przedstawiono w raporcie [9]. Ich znajomość nie jest wymagana do zrozumienia procesu konwersji. W pierwszej kolejności omówiono wersję napisaną w Pascalu [1], ze względu na oczywiste podobieństwo między Pascalem i Adą.

W programie przedstawionym na wydruku 1 zdefiniowano funkcję **random**, będącą generatorem liczb losowych. Zmienne ix, iy i iz stanowią tzw. posiew (ang. seeds) do właściwych obliczeń i muszą być zainicjowane przed pierwszym wywołaniem. Ich wartości są całkowite i muszą znajdować się w przedziałach, odpowiednio, [1, 30268], [1, 30306] i [1, 30322]. Służą one do wygenerowania 48-bitowych ciągów losowych na podstawie trzech ciągów 16-bitowych o postaci:

ix := (ix*171) mod 30269
iy := (iy*172) mod 30307
iz := (iz*170) mod 30323

Ponieważ 30269, 30307 i 30323 są liczbami pierwszymi, wszystkie ciągi muszą być maksymalnej długości (por. [5], str. 19). Ten krok nie może jednak być wykonany na komputerze 16-bitowym bez powstania nadmiaru. Można tego uniknąć wprowadzając modyfikację:

$iy = k * 176 + r$

gdzie k i r są liczbami całkowitymi o wartościach ograniczonych następująco:

$0 \leqslant k \leqslant 172$
$0 \leqslant r \leqslant 175$

```
program name(output);

var

   ix: 1 .. 30268;   iy: 1 .. 30306;   iz: 1 .. 30322;

function random: real;

   var
      ni, k, r: integer;
      x: real;

   begin
   { calculate k and r for ix }
   k := ix div 177;
   r := ix - k * 177;
   ni := - k - k + 171 * r;

   { -342 <= ni <= 30096, so now reduce range }
   if ni < 0 then
      ix := ni + 30269
   else
      ix := ni;

   { k and r for iy generator }
   k := iy div 176;
   r := iy - k * 176;
   ni := - 35 * k + r * 172;

   { reduce range, -6020 <= ni <= 30100 }
   if ni < 0 then
      iy := ni + 30307
   else
      iy := ni;

   { now the same for iz }
   k := iz div 178;
   r := iz - k * 178;
   ni := - 63 * k + r * 170;

   { reduce range, -10710 <= ni <= 30090 }
   if ni < 0 then
      iz := ni + 30323
   else
      iz := ni;


   x := ix/30269.0 + iy/30307.0 + iz/30323.0;
   random := x - trunc(x)
   end {random} ;


   ...

begin  {Main program}

   iz := 1;
   iy := 10001;
   ix := 4987;

   ...
   x := random;
   ...
end.
```

Wydruk 1. Oryginalny program w Pascalu

Wtedy otrzymuje się równości:

$172 * iy = k * 176 * 172 + r * 172 =$
$= k * 30272 + r * 172 =$
$= -k * 35 + r * 172 \mod 30307$

a postępując analogicznie dla pozostałych zmiennych — równości:

$iz = k * 178 + r$
$170 * iz = -k * 63 + r * 170 \mod 30323$
$ix = k * 177 + r$
$171 * ix = -k * 2 + r * 171 \mod 30269$

Nowe wartości są ograniczone do szesnastu bitów, a ich okres wynosi ok. 6.95E12. Część ułamkowa sumy tych

trzech wartości, będąca częścią ułamkową z dzielenia przez liczby pierwsze, stanowi rzeczywistą wartość losową.

Program w Pascalu można przekształcić na program w Adzie, zmieniając jedynie składnię jego tekstu. Jednakże jest oczywiste, że należy zmodyfikować użycie zmiennych globalnych ix, iy i iz w celu uniknięcia możliwości ich niewłaściwego użycia. Można to łatwo osiągnąć w Adzie, umieszczając zmienne w pakiecie, tak że będą one użyte tylko w części implementacyjnej pakietu, a nie w jego specyfikacji. W konsekwencji należy zapewnić sposób odczytywania i ustawiania wartości tych zmiennych, aby umożliwić powtarzalność ciągów, jeśli zajdzie potrzeba. Przy uwzględnieniu tych zmian, specyfikacja pakietu w Adzie ma następującą postać:

```
package Random_Numbers is
    function Random return Float;
      —— udostępnia wartość losową z zakresu 0.0 .. 1.0
    procedure Seeds_Are (X, Y, Z : out Integer);
      —— udostępnia wartość posiewu
    procedure Restart (X, Y, Z : in Integer);
      —— wznowienie od określonych wartości posiewu
         z zakresu 1 .. 30 000
end Random_Numbers;
```

Specyfikacja zawiera prosty i elegancki opis pakietu, wolny od szczegółów implementacyjnych. Pożądane jest także uwzględnienie specyfiki semantyki, lecz żaden z omawianych języków nie ma mechanizmów służących temu celowi. Proponuje się rozszerzenie Ady przez komentarze, umożliwiające uwzględnienie takiej informacji semantycznej, np. [3]. W rzeczywistości można określić jedną właściwość rozważanego pakietu, mianowicie — ograniczenie zakresu wartości funkcji Random do przedziału (0.0 .. 1.0).

```
package body Random_Numbers is

   IX, IY, IZ : Integer;


function Random return Float is
   W : Float;

   begin
      IX := 171 * (IX mod 177) - 2 * (IX/177);
      IY := 172 * (IY mod 176) - 35 * (IY/176);
      IZ := 170 * (IZ mod 178) - 63 * (IZ/178);
      if IX < 0 then
         IX := IX + 30269;
      end if;
      if IY < 0 then
         IY := IY + 30307;
      end if;
      if IZ < 0 then
         IZ := IZ + 30323;
      end if;
      W := Float(IX)/30269.0 + Float(IY)/30307.0 + Float(IZ)/30323.0;
      return W  - Float (Integer (W - 0.5));
   end Random;

procedure Seeds_Are (X, Y, Z : out Integer) is
   begin
      X := IX;
      Y := IY;
      Z := IZ;
   end Seeds_Are;


procedure Restart (X, Y, Z : in Integer) is
   begin
      IX := ( X - 1 ) mod 30269 + 1;
      IY := ( Y - 1 ) mod 30307 + 1;
      IZ := ( Z - 1 ) mod 30323 + 1;
   end Restart;

begin

   IX := 1;
   IY := 10001;
   IZ := 4987;

end Random_Numbers;
```

Wydruk 2. Ciało pakietu po konwersji w języku Ada

Ciało pakietu po konwersji na język Ada przedstawiono na wydruku 2. Jedyna różnica w porównaniu z wersją napisaną w Pascalu polega na zainicjowaniu zmiennych posiewu IX, IY i IZ. Dzięki umieszczeniu odpowiednich instrukcji w ciele pakietu, zmienne są inicjowane automatycznie bez żadnego udziału użytkownika. Należy zauważyć, że wskutek ukrycia zmiennych IX, IY i IZ można uprościć ciało funkcji Random, gdyż nie ma już konieczności umieszczania ścisłych ograniczeń na zakresy wszystkich wartości. Nowa procedura Restart mogłaby powodować powstanie wyjątku sygnalizującego, że określone liczby wychodzą poza dopuszczalny zakres. Zamiast tego jednak są one przekształcane na wartości wewnątrz zakresu.

UWZGLĘDNIENIE PRZENOŚNOŚCI

W oryginalnej wersji algorytmu zaprogramowanej w Fortranie, uwzględniono dwa różne warianty. Jeden z nich jest przeznaczony do arytmetyki 24-bitowej, a drugi — do 16-bitowej, choć zewnętrznie oba są identyczne. W Fortranie lub Pascalu dostarcza się obu wariantów pozostawiając użytkownikowi wybór odpowiedniejszego dla jego komputera. Opublikowana wersja fortranowska ma jeden z wariantów wbudowany jako komentarz (wydruk 3).

```
      FUNCTION RANDOM (L)
C
C        ALGORITHM AS 183  APPL. STATIST. (1982) VOL.31, NO.2
C
C        RETURNS A PSEUDO-RANDOM NUMBER RECTANGULARLY DISTRIBUTED
C        BETWEEN 0 AND 1.
C
C        IX, IY AND IZ SHOULD BE SET TO INTEGER VALUES BETWEEN
C        1 AND 30000 BEFORE FIRST ENTRY
C
C        INTEGER ARITHMETIC UP TO 30323 IS REQUIRED
C
      COMMON/RAND/IX, IY, IZ
      IX =  171 * MOD(IX, 177) - 2  * (IX/177)
      IY =  172 * MOD(IY, 176) - 35 * (IY/176)
      IZ =  170 * MOD(IZ, 178) - 63 * (IZ/178)
C
      IF (IX .LT. 0) IX = IX + 30269
      IF (IY .LT. 0) IY = IY + 30307
      IF (IZ .LT. 0) IZ = IZ + 30323
C
C      IF INTEGER ARITHMETIC UP TO 5212632 IS AVAILABLE,
C      THE PRECEDING 6 STATEMENTS MAY BE REPLACED BY
C
C      IX = MOD(171 * IX, 30269)
C      IY = MOD(172 * IY, 30307)
C      IZ = MOD(170 * IZ, 30323)
C
C      ON SOME MACHINES, THIS MAY SLIGHTLY INCREASE
C      THE SPEED. THE RESULTS WILL BE IDENTICAL.
C
      RANDOM = AMOD(FLOAT(IX)/30269.0 +
     $              FLOAT(IY)/30307.0 + FLOAT(IZ)/30323.0, 1.0)
      RETURN
      END
```

Wydruk 3. Opublikowana wersja procedury w Fortranie

W Adzie dąży się do osiągnięcia bardzo dużej przenośności zakładając, że kompilatory języka muszą być identyczne. Dlatego można opracować program w wielu wariantach nie wymagający zmian w tekście źródłowym. W rozważanym przypadku Ada zapewnia środki do określania długości reprezentacji liczb. Rozmiar reprezentacji liczb całkowitych w bitach jest określony przez stałą Integer'Size. Z tego względu wybór algorytmu może być dokonany automatycznie. W pewnych okolicznościach jednak wybór algorytmu może się nie udać. Jest to wykrywane przez sprawdzenie wartości Integer'Last, tak że może być zgłoszone powstaniem wyjątku Constraint_Error. Ciało pakietu uwzględniającego tę sytuację przedstawiono na wydruku 4. Znaczenie komentarzy w języku angielskim omówiono w podpisie do wydruku.

Istotne w tym pakiecie jest użycie operatora warunkowego „and then". Wartość wyrażenia logicznego (Integer'Last < 5212632) oblicza się tylko wtedy, gdy stała Simple_Case ma wartość TRUE. Wówczas obliczenie ma sens, lecz w przeciwnym przypadku próba konwersji liczby 5212632 na typ Integer spowodowałaby zgłoszenie wyjątku Constraint_Error.

DOBÓR STYLU

Choć proces konwersji można uznać za zakończony, gdyż uwzględniono w nim oba warianty fortranowskie, dokładny przegląd wykazuje braki w stylu programowania. Abstrakcyjnie, generator liczb losowych ma pojedynczy posiew określający kolejne liczby. W zaimplementowanym algorytmie posiew tworzą trzy wartości całkowite. W Adzie jednolitość posiewu można ująć jako typ danych, co ma następujące odbicie w specyfikacji (odpowiednie zmiany w ciele pakietu nie wymagają dodatkowych wyjaśnień):

```
package Random_Numbers is
    function Random return Float;
      —— udostępnia wartość losową z zakresu 0.0 .. 1.0
    type Seed is
        record
            X, Y, Z : Integer;
```

```
      end record;
   function Current_Seed return Seed;
      —— udostępnia wartość posiewu
   procedure Restart (Restart_Seed : in Seed);
      —— wznowienie od określonych wartości posiewu
         —— z zakresu 1 .. 30_000.
end Random Numbers;
```

```
package body Random_Numbers is

   IX, IY, IZ : Integer;   -- The three seeds.


   Simple_Case : constant Boolean := Integer'Size >= 24;
      -- Simple_Case determines which algorithm will be used,
      -- it should be evaluated by the compiler and in consequence
      -- no additional overhead will be placed on the running program.

function Random return Float is
   W : Float;

   begin
      if Simple_Case then
         IX := (171 * IX) mod 30269;
         IY := (172 * IY) mod 30307;
         IZ := (170 * IZ) mod 30323;
      else
         IX := 171 * (IX mod 177) - 2 * (IX/177);
         IY := 172 * (IY mod 176) - 35 * (IY/176);
         IZ := 170 * (IZ mod 178) - 63 * (IZ/178);
         if IX < 0 then
            IX := IX + 30269;
         end if;
         if IY < 0 then
            IY := IY + 30307;
         end if;
         if IZ < 0 then
            IZ := IZ + 30323;
         end if;
      endif;
      W := Float(IX)/30269.0 + Float(IY)/30307.0 + Float(IZ)/30323.0;
      return W - Float (Integer (W - 0.5));
   end Random;

procedure Seeds_Are (X, Y, Z : out Integer) is
   begin
      X := IX;
      Y := IY;
      Z := IZ;
   end Seeds_Are;


procedure Restart (X, Y, Z : in Integer) is
   begin
      IX := ( X - 1 ) mod 30269 + 1;
      IY := ( Y - 1 ) mod 30307 + 1;
      IZ := ( Z - 1 ) mod 30323 + 1;
   end Restart;

begin

   -- Check that Integer has sufficient range
   -- for the generator to work at all.
   if Integer'Last < 30323 then
      raise Constraint_Error;
   end if;

   -- Check that Simple_Case gives correct selection. Raise
   -- Constraint_Error in unlikely case that Simple_Case does
   -- not give correct distinction.
   if Simple_Case and then (Integer'Last < 5212632) then
      raise Constraint_Error;   -- unlikely
   end if;

   IX := 1;
   IY := 10001;
   IZ := 4987;

end Random_Numbers;
```

Wydruk 4. Ciało pakietu uwzględniającego sytuacje wyjątkowe. Stała Simple_Case obliczana w czasie kompilacji określa, który algorytm będzie użyty. Instrukcje if w aktywnej części pakietu zapewniają poprawne działanie generatora niezależnie od długości słowa komputera

DOMYŚLNE INICJOWANIE POSIEWU

W oryginalnym algorytmie inicjowano posiew przez zwykłe przypisanie stałych trzem zmiennym całkowitym. Taka metoda wystarcza do otrzymania powtarzalnych ciągów losowych. W praktyce, ciągi losowe zaczynające się od tej samej wartości, nie są typowe. Jeden ze sposobów pozwalających na uniknięcie powtarzalności polega na inicjowaniu danych wartościami uzyskanymi z wewnętrznego zegara komputera. W Adzie umożliwia to zdefiniowany pierwotnie pakiet Calendar. Odpowiednie zmiany, nie wpływające na specyfikację pakietu, lecz tylko na jego ciało, przedstawiono na wydruku 5.

Trzy zmienne całkowite tworzące posiew są inicjowane wartościami, którym odpowiada różna częstotliwość zmian czasu. Trzecia składowa jest — na przykład — zwiększana co milisekundę. W normie Ady określono rozdzielczość typu Duration co najmniej na 20 ms, nie podano w niej

```
with Calendar;
package body Random_Numbers is

   Simple_Case : constant Boolean := Integer'Size >= 24;

   S : Seed;

function Random return Float is
   W : Float;
   begin
      if Simple_Case then
         S := ((171 * S.X) mod 30269,
               (172 * S.Y) mod 30307,
               (170 * S.Z) mod 30323);
      else
         -- not Simple_Case
         S := (171 * (S.X mod 177) - 2 * (S.X / 177),
               172 * (S.Y mod 176) - 35 * (S.Y / 176),
               170 * (S.Z mod 178) - 63 * (S.Z / 178));
         if S.X < 0 then
            S.X := S.X + 30269;
         end if;

         if S.Y < 0 then
            S.Y := S.Y + 30307;
         end if;
         if S.Z < 0 then
            S.Z := S.Z + 30323;
         end if;
      end if;

      W := Float(S.X)/30269.0 +
           Float(S.Y)/30307.0 +
           Float(S.Z)/30323.0;
      return W - Float (Integer (W - 0.5));
   end Random;

function Current_Seed return Seed is
   begin
      return S;
   end Current_Seed;


   procedure Restart (Restart_Seed : in Seed) is
   begin
      Seed := ( ( Restart_Seed.X - 1 ) mod 30269 + 1,
                ( Restart_Seed.Y - 1 ) mod 30307 + 1,
                ( Restart_Seed.Z - 1 ) mod 30323 + 1 );
   end Restart;

begin

   if Integer'Last < 30323 then
      raise Constraint_Error;
   end if;

   if Simple_Case and then (Integer'Last < 5212632) then
      raise Constraint_Error;
   end if;

   -- Initialize the seed.
   declare
      use Calendar;
      T : Time := Clock;
      Count: Integer := Integer(Seconds(T)/3);
             -- in range 1 .. 28801, increases every 3 secs.
   begin
      S := (Month(T),
            Day(T) + Count,
            Integer(1000*Seconds(T) - 3000*Duration(Count)) );
   end;
end Random_Numbers;
```

Wydruk 5. Pakiet uwzględniający automatyczne inicjowanie posiewu

jednak wymaganej rozdzielczości funkcji Clock. W rzeczywistości, w jednej z użytych wersji zatwierdzonego kompilatora firmy ROLM stwierdzono rozdzielczość sekundową.

W celu otrzymania powtarzalnych ciągów można nadal używać procedury Restart, np.:

```
Restart ( (1,10001,4987) );
```

ROŻNORODNOŚĆ CIĄGÓW I WIELOZADANIOWOŚĆ

Choć przedstawiona wersja generatora nadaje się do wielu celów, ma dwa poważne braki. Omówimy je rozważając użycie pakietu w programie symulacji złożonego problemu ze zdarzeniami przypadkowymi.

Każdemu zdarzeniu przypadkowemu może odpowiadać inny ciąg losowy. Ich rozkłady mogą być różne od równomiernego w przedziale [0,1], zapewnionego przez opracowany pakiet. Co więcej, w czasie testowania zazwyczaj zachodzi potrzeba zmiany parametrów określonego ciągu, przy ustaleniu parametrów pozostałych ciągów. Nie pozwala na to pojedynczy ciąg z jednym posiewem. Jedynym rozwiązaniem jest pozwolenie użytkownikowi na utworzenie nowego ciągu z własnym posiewem.

Można przypuszczać, że najłatwiejsze byłoby przekazywanie posiewu jako parametru funkcji Random. Niestety, nie jest to dopuszczalne w Adzie, ponieważ parametr funkcji musiałby być w trybie **inout**. W celu pokonania tej trudności należałoby przekształcić funkcję Random na procedurę. Choć takie rozwiązanie mogłoby odpowiadać purystom, w tej pracy uznano je za nieatrakcyjne i odrzucono.

Złożony problem symulacji można zaprogramować w Adzie bardzo elegancko, tworząc zadania odpowiadające poszczególnym symulowanym procesom. Stanowiłyby one bardziej realny model procesów. Jednakże zadania w Adzie nie powinny dzielić wspólnych zmiennych. Takie współdzielenie zachodzi wtedy, gdy dwa zadania używają tego samego generatora liczb losowych (współdzieloną zmienną jest wspólny posiew). Ten zakaz współdzielenia jest bardzo istotny, ponieważ w przeciwnym przypadku generator dostarczałby wyników nieokreślonych (a nie przypadkowych). W tej sytuacji należy pozwolić użytkownikowi na utworzenie nowego generatora w każdym zadaniu, tak aby nie zachodziła potrzeba współdzielenia posiewu.

W Adzie można utworzyć wiele konkretów (egzemplarzy) pakietu, jeżeli pakiet jest rodzajowy (ang. generic). Konieczność istnienia takiej konstrukcji podkreślano wielokrotnie [2]. Specyfikacja pakietu rodzajowego jest następująca:

generic
package Random_Numbers **is**
—— treść taka jak poprzednio
end Random_Numbers;

natomiast jego ciało pozostaje bez zmian w stosunku do pakietu zwykłego. Jednakże pakiet rodzajowy nie może być użyty bezpośrednio, lecz musi być poddany konkretyzacji:

package My_Random **is new** Random_Numbers;

Ciągi losowe otrzymuje się przez wielokrotne wywołania funkcji My_Random.Random.

Istotnym problemem w stosowaniu rodzajowej wersji pakietu jest konieczność jego konkretyzacji. Dla użytkowników nie wymagających wielokrotnych ciągów ani wielozadaniowości, czyli w najprostszych przypadkach, jest to bardzo niewygodne. Rozwiązanie tego problemu, przyjęte według zaleceń sformułowanych dla oprogramowania matematycznego [7], polega na utworzeniu dwóch pakietów — pakietu rodzajowego o nazwie Gen_Random_Numbers i jego standardowego konkretu (wydruk 6).

```
generic
package Gen_Random_Numbers is

   function Random return Float;
      -- Returns a random value in the range 0.0 .. 1.0.

   type Seed is
         record
            X, Y, Z : Integer;
         end record;

   function Current_Seed return Seed;
      -- Gives the value of the seed.

   procedure Restart ( Restart_Seed : in Seed );
      -- To restart from particular seeds or
      -- from any values in the range 1 .. 30_000.

end Gen_Random_Numbers;
```

Wydruk 6a. Specyfikacja pakietu rodzajowego

Nowy pakiet rodzajowy ma następującą specyfikację

generic
package Gen_Random_Numbers **is**
—— treść taka jak poprzednio
end Gen_Random_Numbers;

a jego ciało nie różni się od poprzedniego. Nierodzajową wersję pakietu tworzy się przez konkretyzację:

package Random_Numbers **is new** Gen_Random_Numbers;

Dzięki zapewnieniu tej samej specyfikacji wysiłek związany z przekształceniem programu o pojedynczym ciągu losowym na program z wieloelementowym zbiorem generatorów jest zminimalizowany.

Alternatywnym rozwiązaniem zagadnienia wielozadaniowości jest umieszczenie generatora wewnątrz zadania w Adzie. Wtedy posiew staje się zmienną chronioną w zadaniu. Ta metoda ma jednak dwie wady. Po pierwsze, wywołanie generatora (tj. wejścia w zadanie) byłoby wywołaniem procedury, a nie funkcji, a po drugie — efektywność generatora byłaby znacznie gorsza wskutek narzutu związanego z szeregowaniem. Podejście polegające na użyciu zadań jest bardzo naturalne dla języka Occam, zaprojektowanego dla mikroprocesora o nazwie Transputer, firmy Inmos [4]. Każde zadanie byłoby wykonywane przez inny procesor (transputer), a łącznie tworzyłyby one jeden program w języku Occam.

```
with Calendar;
package body Gen_Random_Numbers is

   Simple_Case : constant Boolean := Integer'Size >= 24;
      -- Simple_Case determines which algorithm will be used,
      -- it should be evaluated by the compiler and in consequence
      -- no additional overhead will be placed on the running program.

   S : Seed;

function Random return Float is
   W : Float;
   begin
      if Simple_Case then

         S := ((171 * S.X) mod 30269,
               (172 * S.Y) mod 30307,
               (170 * S.Z) mod 30323);
      else

         S := (171 * (S.X mod 177) - 2 * (S.X / 177),
               172 * (S.Y mod 176) - 35 * (S.Y / 176),
               170 * (S.Z mod 178) - 63 * (S.Z / 178));
         if S.X < 0 then
            S.X := S.X + 30269;
         end if;

         if S.Y < 0 then
            S.Y := S.Y + 30307;
         end if;
         if S.Z < 0 then
            S.Z := S.Z + 30323;
         end if;
      end if;

      W := Float(S.X)/30269.0 +
           Float(S.Y)/30307.0 +
           Float(S.Z)/30323.0;
      return W - Float (Integer (W - 0.5));
   end Random;

function Current_Seed return Seed is
   begin
      return S;
   end Current_Seed;


   procedure Restart (Restart_Seed : in Seed) is
   begin
      Seed := ( ( Restart_Seed.X - 1 ) mod 30269 + 1,
                ( Restart_Seed.Y - 1 ) mod 30307 + 1,
                ( Restart_Seed.Z - 1 ) mod 30323 + 1 );
   end Restart;

begin

   -- Check that Integer has sufficient range
   -- for the generator to work at all.
   if Integer'Last < 30323 then
      raise Constraint_Error;
   end if;

   -- Check that Simple_Case gives correct selection. Raise
   -- Constraint_Error in unlikely case that Simple_Case does
   -- not give correct distinction.
   if Simple_Case and then (Integer'Last < 5212632) then
      raise Constraint_Error;
   end if;

   -- Initialize the seed.
   declare
      use Calendar;
      T : Time := Clock;
      Count: Integer := Integer(Seconds(T)/3);
                -- in range 1 .. 28801, increases every 3 secs.
   begin
      S := (Month(T),
            Day(T) + Count,
            Integer(1000*(Seconds(T) - 3*Duration(Count))) + 2000 );
   end;
end Gen_Random_Numbers;
```

Wydruk 6b. Ostateczna postać programu poddanego konwersji w języku Ada — ciało pakietu

PODSUMOWANIE

Jak wynika z powyższych rozważań, istnieje zasadnicza różnica między prostym przekształceniem podprogramu a utworzeniem idealnego pakietu w Adzie. Ta różnica wynika z dużych możliwości Ady, pozwalających na zapewnienie większej funkcjonalności. W przedstawionym przykładzie, zwiększenie funkcjonalności, nieosiągalne przy programowaniu w Fortranie, polega na możliwości:

- automatycznego wyboru jednego z dwóch wariantów zgodnie z właściwościami użytego komputera,

- polepszenia ochrony posiewu przed niewłaściwym dostępem,
- automatycznego, domyślnego inicjowania posiewu dzięki użyciu zegara wewnętrznego (w sposób niezależny od komputera),
- tworzenia ciągów wielokrotnych przy zapewnieniu bezpieczeństwa w środowisku wielozadaniowym.

Należy sądzić, że konwersja innych programów ujawni dodatkowe możliwości tego rodzaju. Jedno jest pewne, że przynajmniej w konwersji na język Ada nie należy oczekiwać uzyskania dokładnie takiej samej funkcjonalności.

Autorzy wyrażają podziękowanie za szczegółowe komentarze dotyczące pierwszej wersji tego artykułu, które przygotował L. D. Hill, D. A. Watt, R. S. Scowen i David May przedstawili uwagi, które również przyczyniły się do ulepszenia tekstu. Dr L. D. R. Chandesris wykazał, że błąd w inicjowaniu posiewu może powodować powstanie nadmiaru, co zostało poprawione w obecnej wersji.

Tłumaczył i opracował:
JANUSZ ZALEWSKI

LITERATURA

[1] British Standards Institution: The Pascal Compiler Validation Suite. Version 4.0, October 1983
[2] Currie I. F., Peeling N. E.: Modular Compilation Systems for High Level Programming Languages. Algol Bulletin, No. 48, August 1982
[3] Hill A.: Towards an Ada-based specification and design language. Ada UK News, Vol. 4, No. 4, October 1983
[4] INMOS Ltd: Occam. Prentice-Hall, Englewood Cliff (NJ), 1984
[5] Knuth D. E.: Seminumerical Algorithms. Addison-Wesley, London, 1969
[6] Numerical Algorithms Group Ltd: NAG Fortran Library. 256 Banbury Road, Oxford, Wielka Brytania
[7] Symm G. T., Wichmann B. A., Kok J., Winter D. T.: Guidelines for the design of large modular scientific libraries in Ada. Report DITC 28/83, National Physical Laboratory, July 1983
[8] Wichmann B. A., Hill I. D.: Algorithm AS 183 — An efficient and portable pseudo-random generator. Applied Statistics, Vol. 31, No. 2, 1982
[9] Wichmann B. A., Hill I. D.: A Pseudo-Random Number Generator. Report DITC 6/82, National Physical Laboratory, Teddington, June 1982.

ROMAN FABER
MICHAŁ OSTROWSKI
Instytut Informacji Naukowej, Technicznej i Ekonomicznej
Warszawa

UPT — uniwersalny procesor do redagowania tekstów dla MERY 400

W uczelniach, instytutach naukowych, jak również w biurach i urzędach występuje potrzeba przygotowywania krótkich opracowań tekstowych, które często mają złożoną budowę wewnętrzną i bogatą szatę graficzną. Mogą to być raporty, sprawozdania, artykuły, pisma, ogłoszenia itp. Jeszcze silniejsze potrzeby tego rodzaju występują wśród pracowników służb informacji naukowej, gdzie niemal codziennym problemem jest opracowywanie różnego rodzaju wydawnictw informacyjnych.

Przygotowywanie takich opracowań tradycyjnymi metodami jest nadzwyczaj uciążliwe. Względy te przesądziły o podjęciu prac nad oprogramowaniem automatyzującym procesy technicznego redagowania tekstu. W ich wyniku powstał pakiet programów o nazwie UPT (Uniwersalny Procesor Tekstowy), przeznaczony do kształtowania typograficznego dowolnych tekstów, zarejestrowanych na nośniku magnetycznym w postaci rekordów znakowych tworzących pliki sekwencyjne. Pakiet UPT umożliwia określanie fizycznych rozmiarów strony, rozmiarów szpalty, sposobu redagowania wiersza, podział tekstu na rozdziały, sporządzanie spisu treści oraz skorowidza.

Pakiet UPT jest eksploatowany na standardowym zestawie minikomputera MERA-400 z jednostką pamięci dyskowej.

Do przedstawiania wyników przetwarzania może być używany monitor ekranowy, jednak zasadniczym urządzeniem wyjściowym, służącym do tych celów, jest drukarka mozaikowa. Nośnik magnetyczny, na którym zarejestrowane są przetwarzane pliki, stanowią kasety pamięci dysko-

Dr inż. ROMAN FABER w 1972 r. ukończył Wydział Elektroniki Politechniki Warszawskiej. Pracuje w Instytucie Informacji Naukowej, Technicznej i Ekonomicznej, gdzie zajmuje się projektowaniem oraz tworzeniem minikomputerowych i mikrokomputerowych systemów informacyjno-wyszukiwawczych.

Mgr inż. MICHAŁ OSTROWSKI w 1972 r. ukończył Wydział Elektroniki Politechniki Warszawskiej o specjalności maszyny matematyczne. Pracował w Ośrodku Informatyki Technicznej i Przetwarzania Danych w OLPiT. Zajmował się problematyką związaną z organizacją baz danych i automatyzacją procesów edycyjnych w zastosowaniach do automatyzacji pracy służb informacyjnych resortu łączności i automatycznej edycji spisów telefonicznych. Od 1984 r. pracuje w Instytucie Informacji Naukowej, Technicznej i Ekonomicznej. Zajmuje się zastosowaniami mikrokomputerów do celów informacyjnych.

wej. Plik będący przedmiotem przetwarzania określany jest mianem pliku źródłowego, zaś plik zawierający tekst zredagowany — nazwą pliku wynikowego. Podczas przetwarzania korzysta się z systemu operacyjnego SOM 3 oraz edytora tekstowego UPD.

Pakiet został zrealizowany w języku FORTRAN IV, co powinno umożliwić łatwe przenoszenie pakietu na inne komputery wyposażone w translator tego języka.

Istnieje możliwość drukowania znaków alfabetu polskiego w pliku wynikowym. Znaki te należy w pliku źródłowym zakodować w postaci określonych ciągów dwuznakowych, złożonych z litery i cyfry. Odpowiedni znak zostaje utworzony przez wydrukowanie w pierwszej kolejności znaku alfabetu łacińskiego, a następnie nadrukowanie odpowiedniego elementu uzupełniającego (takiego jak: przecinek, kreska ukośna i apostrof).

DYREKTYWY

Proces redagowania tekstu za pomocą pakietu UPT jest opisywany dyrektywami. Służą one do przekazania wartości numerycznych parametrów określających sposób redagowania tekstu, napisów, jak również do ustalenia trybów redagowania.

Dyrektywy mają następującą budowę:

: ⟨nazwa⟩ ⟨parametr$_1$⟩ ... ⟨parametr$_n$⟩

Dyrektywa rozpoczyna się znakiem „:" dosuniętym do lewego marginesu. Nazwa (w języku angielskim) służy do zidentyfikowania dyrektywy przez procesor. Dopuszcza się skracanie nazw dyrektyw. Po nazwie następuje grupa parametrów oddzielonych od siebie spacjami. Niektóre z parametrów mają być pominięte. Zostaną wtedy użyte wartości domyślne tych parametrów. Dyrektywy definiowania trybów redagowania nie zawierają parametrów.

FUNKCJE PAKIETU

Redagowanie rozmiarów strony polega na zdefiniowaniu formatu strony. Możliwe jest określenie długości strony wyrażonej liczbą wierszy oraz szerokości strony — jako liczby znaków w wierszu. Maksymalna długość strony nie jest ograniczona, zaś maksymalna szerokość wynosi 170 znaków (standardowo: 66 wierszy i 85 znaków). W ramach strony można definiować rozmiary szpalty poprzez definicję wielkości marginesów. Marginesy górny i dolny są określane jako liczba pustych wierszy (standardowa wartość marginesu górnego wynosi 7 wierszy, zaś marginesu dolnego — 5 wierszy). Marginesy lewy i prawy nie są rozróżnione, a więc nadaje się im tę samą wartość, określoną liczbą spacji (standardowa wartość dla tych marginesów wynosi 7 spacji). Szpalta tekstu może być podzielona na dwie kolumny. Odstęp pomiędzy nimi jest również traktowany jako margines, którego wielkość może być definiowana (standardowa wartość tego marginesu wynosi 5 spacji). Marginesy boczne mogą być tymczasowo przesuwane zarówno w lewo, jak i w prawo. Umożliwia to tworzenie wcięć w tekście, bądź lokalne wysuwanie tekstu poza margines.

Marginesy górny oraz dolny mogą być zaopatrzone w jednowierszowe napisy umieszczone pośrodku odpowiedniego marginesu. W przypadku marginesu górnego napis ten nosi nazwę nagłówka, zaś w wypadku marginesu dolnego — notki. Zarówno nagłówek, jak i notka mogą być podzielone na trzy napisy. Pierwszy z nich zostanie wówczas dosunięty do lewego marginesu, drugi napis zostanie umieszczony pośrodku wiersza, zaś trzeci — zostanie dosunięty do marginesu prawego. Na żądanie, nagłówek bądź notka mogą być automatycznie zaopatrzone w numery stron.

Redagowanie wiersza polega na określeniu sposobu przesyłania słów wierszy pliku źródłowego do wierszy pliku wynikowego. Używane do tego dyrektywy są podporządkowane dyrektywom redagowania strony w tym sensie, że format strony nie może być zmieniony za pomocą dyrektyw redagowania wiersza. Wyróżnia się dwie grupy omawianych dyrektyw: **dyrektywy generalne** i **dyrektywy lokalne.**

Wykaz dyrektyw pakietu UPT

ADJUST — tryb przesyłania słów
BMARGIN [n] — dolny margines
BREAK — przerwanie wypełniania wiersza
CMARGIN [m] — odstęp między kolumnami
COLUMNS [i] — liczba kolumn
DEFINE ⟨nazwa⟩⟨tekst⟩ — zdefiniowanie symbolu
DDOWN [tekst] — utworzenie nagłówka punktu
DDSUPPRESS [tekst] — utworzenie nagłówka
DINDENT L m1 m2 — wcięcie poziomu L
DLEVEL [l] — określenie poziomu nagłówka
DLIMIT l — ograniczenie nagłówków w indeksie
DNEXT [tekst] — kolejny nagłówek punktu
DNSUPPRESS [tekst] — kolejny nagłówek
DRESET [n] [m] — początkowy numer punktu
DSKIP L n1 n2 — odstępy nagłówka poziomu L
DUP [k] — zmniejszenie numeru poziomu nagłówka
EFOOTER /tekst1/tekst2/tekst3/ — notka strony parzystej
EHEADER /tekst1/tekst2/tekst3/ — nagłówek strony parzystej
EJECT — nowa strona
FILL — tryb przesyłania słów
FOOTER /tekst1/tekst2/tekst3/ — notka
HEADER /tekst1/tekst2/tekst3/ — nagłówek
INDENT [m] — wcięcie tekstu
INDEX ⟨napis⟩ — termin do skorowidza
LENGTH [n] — długość strony
NEED n — blok nagłówka
NFILL — tryb przesyłania słów
OFOOTER /tekst1/tekst2/tekst3/ — notka strony nieparzystej
OHEADER /tekst1/tekst2/tekst3/ — nagłówek strony nieparzystej
PARAGRAPH [m] [n] — akapit
QUIT — przerwanie redagowania
RINDENT [m] — wcięcie z prawej strony
RUNDENT [m] — zmniejszenie prawego marginesu
SKIP [n] — puste wiersze
SMARGIN [m] — marginesy boczne
SPACE [u] — interlinia
TAB ⟨znak⟩ $l_1\ l_2 \ldots l_i$ — tabulacja
TMARGIN [n] — margines górny
TOFC — żądanie spisu treści
UNDENT [m] — zmniejszenie lewego marginesu
WIDTH [m] — szerokość strony
) ⟨tekst⟩ — centrowanie tekstu
+ ⟨tekst⟩ — przesłanie verbatim
***** ⟨tekst⟩ — komentarz
/tekst1/tekst2/tekst3/ — rozrzucenie tekstu w wierszu
— numerowanie stron

Dyrektywy generalne określają zasady redagowania wiersza od momentu użycia dyrektywy do jej odwołania. Przesyłanie słów z pliku źródłowego do wierszy pliku wynikowego realizowane jest standardowo w ten sposób, że do każdego wiersza pliku wynikowego przesyłana jest największa (mieszcząca się) liczba słów. Pomiędzy wyrazami umieszczana jest jedna spacja, zaś po kropce, średniku, wykrzykniku, znaku zapytania i przecinku — dwie spacje. Jeśli ostatnie słowo danego wiersza nie jest dosunięte do prawego marginesu, w wierszu umieszcza się dodatkowe spacje, które zwielokrotniają już istniejące spacje. W ten sposób osiąga się dosuwanie tekstu do prawego marginesu.

Kolejny tryb przesyłania słów różni się od opisanego tym, że czynności wypełniania wiersza pliku wynikowego kończone są z chwilą umieszczenia w nim największej (mieszczącej się) liczby słów. Nie wykonuje się w tym wypadku dosuwania do prawego marginesu.

Ostatni ze zdefiniowanych sposobów przesyłania słów polega na tym, że wiersze pliku źródłowego są przesyłane do pliku wynikowego z zachowaniem swej postaci. Realizowana jest natomiast tabulacja, co jest szczególnie dogodne przy budowaniu tabel w tekście.

Dyrektywy generalne umożliwiają ponadto określenie liczby pustych wierszy umieszczonych pomiędzy kolejnymi wierszami tekstu (interlinia) oraz — tabulacji. Tabulację stanowi ciąg pozycji znakowych, odliczanych od lewego marginesu, który może być następnie wykorzystany do rozmieszczania tekstu w wierszu.

Dyrektywy lokalne dotyczą sposobu rozmieszczania tekstu w pojedynczym wierszu. Może to polegać na umieszczeniu wskazanego tekstu pośrodku wiersza (centrowanie tekstu), przesłaniu tekstu bez zmiany postaci (verbatim), bądź rozłożeniu trzech wskazanych tekstów składowych z lewej strony, pośrodku i z prawej strony wiersza. Przewidziano także możliwość umieszczania komentarza w pliku źródłowym.

W ramach lokalnego redagowania wiersza istnieje sposób rezerwowania miejsca — zwanego „dziurą" — na tekst, który zostanie włączony dopiero na etapie przetwarzania pliku źródłowego. Miejsce takie jest etykietowane nazwą ujętą w znaki „%". Nazwie można nadać wartość w postaci tekstowej, za pomocą odpowiedniej dyrektywy. Spowoduje ona, że „dziura" zostanie zastąpiona właściwym tekstem. Dyrektywę tę można podać w trybie nasłuchu dyrektyw lub też umieścić ją w pliku źródłowym. Przedstawiony sposób rezerwowania miejsca na tekst odpowiada częstym sytuacjom praktycznym, gdy trzeba na przykład przygotować nakład pism o takiej samej treści, lecz skierowanych do różnych adresatów. Wygodnie jest zarejestrować tekst pisma jeden raz, nie podając w nim nazwisk, adresów, dat itp., rezerwując natomiast miejsce na te dane. Na etapie przetwarzania tekstu miejsca takie mogą być automatycznie wypełnione wskazanymi tekstami.

Ważną z praktycznego punktu widzenia funkcją realizowaną przez dyrektywy lokalnego kształtowania tekstu jest tworzenie akapitu w tekście. Polega to na przerwaniu przesyłania słów do bieżącego wiersza, wygenerowaniu wskazanej liczby pustych wierszy, umieszczeniu w kolejnym wierszu żądanej liczby spacji, a następnie wznowieniu przesyłania słów.

Lokalne redagowanie wiersza obejmuje również możliwość podkreślania lub wytłuszczania tekstu. Odpowiedni jego fragment, przewidziany do podkreślenia, ujmuje się w podwójne nawiasy kwadratowe. Tekst należący do pliku zostanie wytłuszczony, zaś tekst wchodzący w skład nagłówka lub notki będzie podkreślony.

Kolejna funkcja umożliwia podział tekstu na rozdziały, punkty oraz podpunkty. Każdy z wymienionych fragmentów tekstu zaopatrywany jest w nagłówek o budowie:

⟨numer punktu⟩ [tekst]

gdzie przydzielony automatycznie numer złożony jest z liczb dziesiętnych i kropek, którego budowa zgodna jest z powszechnie stosowaną konwencją numerowania punktów. Podawanie tekstu nie jest obowiązkowe. Uzyskana w ten sposób hierarchiczna struktura może zawierać 8 poziomów bloków tekstu.

Użytkownik ma możliwość określenia postaci bloku tekstu na każdym z poziomów. Polega to na wskazaniu liczby pustych wierszy poprzedzających nagłówek oraz liczby pustych wierszy następujących po nagłówku. Można także regulować wielkość wcięcia nagłówka oraz tekstu.

Posługiwanie się aparatem podziału tekstu na rozdziały polega na wskazaniu miejsc w tekście, gdzie należy utworzyć kolejny blok. Chcąc zapewnić elastyczność tej procedury oddano w ręce użytkownika pewien zakres kontroli nad sposobem przydzielania numerów blokom tekstu. Umożliwia to — między innymi — oddzielne redagowanie rozdziałów.

Spis treści może powstać jedynie wtedy, gdy tekst zostanie podzielony na rozdziały. Automatyczna budowa spisu treści polega na zarejestrowaniu w oddzielnym pliku numerów nagłówków oraz związanych z nimi napisów dla kolejnych bloków tekstu. Każdy z takich wierszy zostaje uzupełniony do prawego marginesu ciągiem kropek oraz numerem strony, na której wystąpił nagłówek. Do decyzji użytkownika pozostawia się wybór największego poziomu bloków tekstu, których numery będą rejestrowane w spisie treści.

Tworzenie skorowidza polega na zarejestrowaniu w oddzielnym pliku terminów wytypowanych przez użytkownika. Każdy z nich zostaje zaopatrzony w numer strony, na której wystąpił.

TRYBY PRACY PAKIETU

Wyróżnia się dwa tryby pracy pakietu:
- nasłuch dyrektyw
- przetwarzanie tekstu.

Pierwszy z trybów rozpoczyna się z chwilą załadowania pakietu do pamięci operacyjnej. Realizacja procedury nasłuchu polega na wczytywaniu i analizowaniu kolejnych

```
SOURCE UPDATE                    23. 04. 1985          STRONA  2
 LISTING

 59    59  OBRAZ KONTROLNY O STRUKTURZE ZIARNISTEJ. ZAKL4OICENIA WIZJI I FONII.
 60    60  W UKL4ADZIE ANTENOWYM NIE STWIERDZA SIE2 USZKODZENIA.
 61    61  :DNSUP [[PRZYCZYNA]]
 62    62  1. ZE WZGLE2DU NA ZBYT MAL4E WZMOCNIENIE PRZEDWZMACNIACZA WIELKIEJ
 63    63  CZE2STOTLIWOS1CI (PODZESPO1L4 1 LUB 2) SZUMY ODBIORNIKA UWIDACZNIAJA2 SIE2 NA
 64    64  EKRANIE. W PRZYPADKU ZASTOSOWANIA W ODBIORNIKU GL4OWIC LAMPOWYCH
 65    65  POWODEM ZMNIEJSZENIA SIE2 WZMOCNIENIA MOZ3E BYC1 ZUZ3YCIE JEDNEJ Z LAMP.
 66    66   2.ZBYT MAL4E WZMOCNIENIE PIERWSZEGO STOPNIA WZMACNIACZA POS1REDNIEJ
 67    67  CZE2STOTLIWOS1CI (PODZESPO1L4 3). W REZULTACIE NA EKRANIE UWIDACZNIAJA2 SIE2
 68    68  SZUMY WZMACNIACZA POS1REDNIEJ CZE2STOTLIWOS1CI.
 69    69  :DNSUP [[POSTE2POWANIE]]
 70    70  W PZYPADKU ODBIORNIKO1W LAMPOWYCH NAPRAWA POLEGA NA WYMIANIE
 71    71  ZUZYTEJ LAMPY (PATRZ "TABELA WYPOSAZ3ENIA STANDARDOWEGO"). USUNIE2CIE
 72    72  BARDZIEJ SKOMPLIKOWANYCH USZKODZEN1 W STOPNIACH WIELKIEJ CZE2STOTLIWOS1CI
 73    73  ODBIORNIKO1W NALEZ3Y POWIERZYC1 PUNKTOWI NAPRAWCZEKU. WYMIANE2
 74    74  TRANZYSTORO1W W PODZESPOL4ACH WIELKIEJ CZE2STOTLIWOS1CI RO1WNIEZ3 POWINIEN
 75    75  PRZEPROWADZIC1 ZAKL4AD NAPRAWCZY, PONIEWAZ3 W NIEKTO1RYCH PRZYPADKACH
 76    76  KONIECZNE BE2DZIE PONOWNE ZESTROJENIE UKL4ADU.
 77    77  :DUP
 78    78  :DNEXT [[OBRAZ ZANIKA PRZY ODBIORZE STACJI UHF]]
 79    79  :DDSUP [[USTEKKA]]
 80    80  APARAT PRACUJE BEZ ZARZUTU W ZAKRESIE UHF PRZEZ KRO1TKI CZAS PO
 81    81  WL4A2CZENIU. NASTE2PNIE OBRAZ STOPNIOWO ZATRACA KONTRAST I W KONICU ZANIKA.
 82    82  EKRAN NADAL ROZS1WIETLONY. DZIWIE2K ZAKL4OICONY SL4ABYMI SZUMAMI.
 83    83  :DNSUP [[PRZYCZYNA]]
 84    84  NA SKUTEK NAGRZANIA SIE2 LAMPY HETERODYNY ZMIENIAJA2 SIE2 POJEMNOS1CI
 85    85  UKL4ADU. LAMPA JEST ZUZ3YTA. NAJPIERW MALEJE AMPLITUDA SYGNA4U HETERODYNY
 86    86  (POGORSZENIE KONTRASTU), A NASTE2PNIE OSCYLACJE CAL4KOWICIE ZNIKAJA2
 87    87  I ODBIO1R STAJE SIE2 NIEMOZ3LIWY.
 88    88  :DNSUP [[POSTE2POWANIE]]
 89    89  USTERKE2 USUWA SIE2 DROGA2 WYMIANY LAMPY MIESZACZA I HETERODYNY
 90    90  W GL4OWICY UHF (PATRZ "TABELA WYPOSAZ3ENIA STANDARDOWEGO").
```

Wydruk 1. Fragment tekstu pliku źródłowego

```
                                   STR. 2                    NAPRAWY UTV

SYGNAW TELEWIZYJNY. W  JEGO  SKLAD                  NIEKTORYCH          PRZYPADKACH
...                                                 KONIECZNE      BEDZIE      PONOWNE
                                                    ZESTROJENIE UKKADU.

                                                                3.2.OBRAZ  ZANIKA  PRZY  ODBIORZE
                                                                STACJI UHF
3.1.OBRAZ ZAKKOCONY

                                                                USTERKA

   USTERKA

                                                                APARAT PRACUJE BEZ ZARZUTU W
   OBRAZ KONTROLNY O STRUKTURZE                                 ZAKRESIE UHF PRZEZ KROTKI CZAS
   ZIARNISTEJ. ZAKKOCENIA WIZJI I                               PO WKACZENIU. NASTEPNIE OBRAZ
   FONII. W UKKADZIE ANTENOWYM NIE                              STOPNIOWO ZATRACA KONTRAST I W
   STWIERDZA SIE USZKODZENIA.                                   KONCU ZANIKA. EKRAN NADAL
                                                                ROZSWIETLONY. DZWIEK ZAKKOCONY
   PRZYCZYNA                                                    SKABYMI SZUKAMI.

   1. ZE WZGKEDU NA ZBYT MAKE                                   PRZYCZYNA
   WZMOCNIENIE PRZEDWZMACNIACZA
   WIELKIEJ CZESTOTLIWOSCI                                      NA SKUTEK NAGRZANIA SIE LAMPY
   (PODZESPÓK 1 LUB 2) SZUMY                                    HETERODYNY ZMIENIAJA SIE
   ODBIORNIKA UWIDACZNIAJA SIE NA                               POJEMNOSCI UKKADU. LAMPA JEST
   EKRANIE. W PRZYPADKU                                         ZUZYTA. NAJPIERW MAKEJE
   ZASTOSOWANIA W ODBIORNIKU                                    AMPLITUDA SYGNAKU HETERODYNY
   GKOWIC LAMPOWYCH POWODEM                                     (POGORSZENIE KONTRASTU), A
   ZMNIEJSZENIA SIE WZMOCNIENIA                                 NASTEPNIE OSCYLACJE CAKKOWICIE
   MOZE BYC ZUZYCIE JEDNEJ Z                                    ZNIKAJA I ODBIOR STAJE SIE
   LAMP.                                                        NIEMOZKIWY.

   2.ZBYT MAKE WZMOCNIENIE                                      POSTEPOWANIE
   PIERWSZEGO STOPNIA WZMACNIACZA
   POSREDNIEJ CZESTOTLIWOSCI                                    USTERKE USUWA SIE DROGA WYMIANY
   (PODZESPÓK 3). W REZULTACIE NA                               LAMPY MIESZACZA I HETERODYNY W
   EKRANIE UWIDACZNIAJA SIE SZUMY                               GKOWICY UHF (PATRZ "TABELA
   WZMACNIACZA POSREDNIEJ                                       WYPOSAZENIA STANDARDOWEGO").
   CZESTOTLIWOSCI.

   POSTEPOWANIE

   W PZYPADKU ODBIORNIKÓW
   LAMPOWYCH NAPRAWA POLEGA NA
   WYMIANIE ZUZYTEJ LAMPY (PATRZ
   "TABELA WYPOSAZENIA
   STANDARDOWEGO"). USUNIECIE
   BARDZIEJ SKOMPLIKOWANYCH
   USZKODZEN W STOPNIACH WIELKIEJ
   CZESTOTLIWOSCI ODBIORNIKÓW
   NALEZY POWIERZYC PUNKTOWI
   NAPRAWCZEKU. WYMIANE
   TRANZYSTORÓW W PODZESPUKACH
   WIELKIEJ CZESTOTLIWOSCI ROWNIEZ
   POWINIEN PRZEPROWADZIC ZAKKAD
   NAPRAWCZY, PONIEWAZ W

GIES, C.M. KIRSCH
```

Wydruk 2. Odpowiedni fragment tekstu pliku wynikowego

dyrektyw. Wprowadzanie dyrektyw kończy się wysłaniem znaku „?". Powoduje to przejście do drugiego trybu pracy. Tryb nasłuchu dyrektyw jest dogodny do określenia wartości parametrów oraz trybów redagowania, które będą obowiązywać przez cały czas przetwarzania tekstu.

Tryb przetwarzania tekstu polega na wczytywaniu kolejnych rekordów z pliku źródłowego, obróbce typograficznej oraz przesyłaniu zredagowanego tekstu do pliku wynikowego. Niektóre z rekordów pliku źródłowego mogą zawierać dalsze dyrektywy redagowania tekstu.

PRZYKŁAD ZASTOSOWANIA PAKIETU

W przykładzie przedstawiono sposób redagowania fragmentu instrukcji dokonywania prostych napraw odbiornika telewizyjnego [1]. Przykład ten dobrze ilustruje podział tekstu na rozdziały.

W trybie nasłuchu dyrektyw zdefiniowano format strony. Liczy ona 70 wierszy, z których każdy ma 90 znaków. Szpalta podzielona jest na dwie kolumny. Szerokość marginesów bocznych wynosi 1 spację, zaś odstęp pomiędzy kolumnami 3 spacje. Strona zaopatrzona jest w notkę informującą o autorach i wydawcy książki.

Na wydruku 1 zawarto plik źródłowy. Zamieszczono w nim dyrektywy redagowania tekstu. Dyrektywa

:HEAD

definiuje nagłówek strony. Występujący w nim znak # oznacza żądanie automatycznego numerowania stron. Grupa dyrektyw

:DSKIP 1 2 1
:DINDENT 1 0 5

definiuje postać bloków tekstu.

Pierwsza z dyrektyw oznacza, że każdy z nagłówków związanych z poziomem pierwszym będzie poprzedzony dwoma pustymi wierszami, zaś po nagłówku wystąpi jeden pusty wiersz.

Druga dyrektywa definiuje wcięcia stosowane na poziomie pierwszym. Przed nagłówkiem wcięcie nie wystąpi, zaś następujący dalej tekst zostanie wcięty o 5 spacji.

Dalsze dyrektywy służą do wskazania miejsc w tekście, w których należy utworzyć kolejne bloki tekstu. W szczególności dyrektywa

:DNEXT

oznacza żądanie utworzenia na tym samym poziomie bloku tekstu, wraz z odpowiednim numerem punktu. Dyrektywa

:DDOWN

spowoduje utworzenie bloku tekstu zaopatrzonego w odpowiedni numer — blok ten powstanie na kolejnym wyższym poziomie struktury bloków tekstu.

Wyniki przetwarzania przedstawiono na wydruku 2. Warto zwrócić uwagę na sposób tworzenia znaków charakterystycznych dla języka polskiego.

[1]) Tekst zaczerpnięto z książki: Kirsch C. M., Gies J.: Elementarne naprawy odbiorników telewizyjnych, WKiŁ, W-wa, 1977.

LITERATURA

[1] Faber R., Ostrowski M., Struk Z.: Implementacja pakietu UPT do automatycznego redagowania tekstów. Maszynopis, Instytut INTE, Warszawa, 1984.

ANDRZEJ T. JANCZURA
Instytut Budownictwa
Politechnika Wrocławska

Rozwiązywanie pasmowych układów równań liniowych na mikrokomputerach

W niniejszym artykule przedstawiono użytkownikom mikrokomputerów osobistych algorytm rozwiązywania pasmowych układów równań liniowych, który umożliwia znaczne zaoszczędzenie czasu obliczeń komputera. Załączono podprogramy napisane w języku Basic dla ZX Spectrum, które po prostej implementacji mogą być z powodzeniem wykorzystywane na innych mikrokomputerach, takich jak Commodore 64 czy bardziej profesjonalny IBM PC.

PODSTAWY TEORETYCZNE

Poszukuje się rozwiązania X liniowego układu równań
A * X = B

gdzie **A** jest macierzą symetryczną, dodatnio określoną, pasmową, nieosobliwą, wymiaru NxN, natomiast **B** jest wektorem N-elementowym.

Do rozwiązania równania wykorzystano standardową metodę Banachiewicza, w której macierz **A** jest dekomponowana na czynniki dolno- i górnotrójkątne. Biorąc pod uwagę symetrię macierzy **A**, można dowieść, że:

$$A = L * L^T$$

gdzie **L** jest macierzą dolnotrójkątną, L^T jest macierzą transponowaną do L.

Dalsze rozwiązanie przebiega według równań:

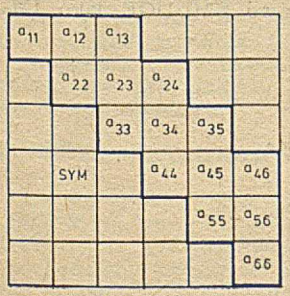
$$L * Y = B$$
$$L^T * X = Y$$

Z pierwszego równania, zwanego „podstawieniem do przodu", wyznacza się **Y**, a z drugiego, zwanego „podstawieniem do tyłu", oblicza się **X**.

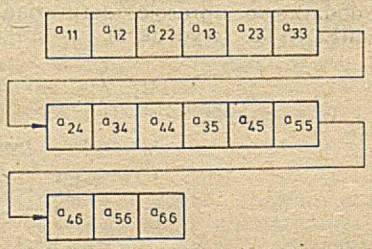
Przedstawiona procedura jest stosowana od dawna, a algorytmy rozkładu macierzy **A**, jak i rozwiązań powyższych równań, są powszechnie znane [1]. Metoda Banachiewicza jest silnie zbieżna [2] i wobec alternatywnego zmodyfikowania rozkładu macierzy **A** [1] ma mniejszą liczbę operacji mnożenia, co jest ważne przy obliczeniach komputerowych.

DZIAŁANIE PROGRAMU

W prezentowanym algorytmie macierz współczynników **A** jest zapamiętywana półpasmami w kierunku kolumn (rys.).



Macierz A



Sposób zapamiętywania macierzy A

Do określenia elementów diagonalnych macierzy **A** wykorzystano tablicę jednowymiarową **G(.)**, przy czym **G(1) = 0**. Przykładowo, dla macierzy **A** z rys. 1, elementy tablicy **G** przyjmują wartości: **G = {0, 1, 2, 5, 6, 11, 14}**
Dla wektora prawych stron **B** oraz rozwiązania **X** zarezerwowano miejsce w tablicy **B(.)**, która jest wykorzystywana sekwencyjnie.

Przed użyciem poniższych podprogramów, napisanych w języku Basic dla ZX Spectrum, należy:

1) wczytać liczbę równań **N**
2) zadeklarować wymiar tablic **B** i **G**, czyli:

DIM G(N + 1) : DIM B(N)

3) wczytać kolejno numery elementów diagonalnych macierzy **A** do tablicy **G(.)**
4) zadeklarować wymiary tablicy **A**, czyli:

DIM A(NMAX), gdzie **NMAX = G (N + 1)**

5) wczytać elementy macierzy **A** (półpasmami w kierunku kolumn) do tablicy **A(.)**
6) wczytać elementy wektora **B** do tablicy **B(.)**.

Po wywołaniu:

GOSUB 200

macierz współczynników **A** jest rozkładana na czynniki trójkątne **L**, przy czym w tablicy **A(.)** początkowa jest macierz **A**, na końcu zaś macierz górnotrójkątna **L**. Rozwiązanie **X** jest otrzymywane po wywołaniu podprogramu:

GOSUB 259

Tabela 1. Półpasmo macierzy A

25	5	2	1					
	17	1	0	3				
		10	0	1	-2			
			20	2	3	0		
				14	0	5	1	
					18	-6	-7	2
SYMETRIA						20	0	-4	6
							16	1	0
								22	0
									10

Tabela 2. Rozwiązanie równania A*X=B

Nr	X	B	A*X
1	.090170009	1	1
2	-.065603115	0	4.6566129E-10
3	-0.55365405	-4	-4
4	0.18107345	3	3
5	0.40601898	0	-5.2386895E-10
6	-0.5078923	0	1.1990778E-8
7	-1.011259	-5	-5
8	-0.23965408	0	-1.5716068E-9
9	-0.12679989	0	4.6566129E-10
10	1.6067554	10	10

i umieszczone w tablicy **B(.)**. Do weryfikacji poprawności rozwiązania **X** służy podprogram:

GOSUB 300

który oblicza iloczyn **A*X**, przechowywany w tablicy **R(.)**.

```
200>LET Y4=.1E-20: IF A(1)<Y4THEN GO TO 229
204 LET A(1)=SQR (A(1))
206 FOR I=2 TO N: LET S2=0: LET S1=0: LET I2=G(I+1)
208 LET I1=G(I): LET I3=G(I-I2+I1+2)
212 FOR K=I1+1 TO I2-1: LET S1=0: LET I3=G(K+I+1-I2)
214 FOR J=1 TO K-I1-1
216 LET S1=S1+A(I3-J)*A(K-J)
218 NEXT J
219 LET A(K)=(A(K)-S1)/A(I3)
220 LET S2=S2+A(K)*A(K): NEXT K
226 LET S1=A(I2)-S2: IF S1>=Y4 THEN GO TO 230
227 IF S1<0 THEN PRINT "MACIERZ A JEST NIEDODATNIO OKRESLONA"
228 IF S1>0 AND S1<Y4 THEN PRINT "MACIERZ A JEST OSOBLIWA"
229 STOP
230 LET A(I2)=SQR (S1)
232 NEXT I: RETURN
259 LET B(1)=B(1)/A(1)
262 FOR I=2 TO N: LET S1=0: LET I2=G(I+1): LET I1=G(I)
264 FOR J=1 TO I2-I1-1
266 LET S1=S1+A(I1+J)*B(I+J-I2+I1)
268 NEXT J
270 LET B(I)=(B(I)-S1)/A(I2): NEXT I
272 LET I=N: FOR K=1 TO N: LET I2=G(I+1)
274 LET I1=G(I): LET B(I)=B(I)/A(I2)
276 FOR J=1 TO I2-I1-1
278 LET B(I-J)=B(I-J)-A(I2-J)*B(I)
280 NEXT J
282 LET I=I-1: NEXT K: RETURN
300 DIM R(N): FOR I=1 TO N: LET I2=G(I+1): LET I1=G(I)
301 LET R(I)=R(I)+A(I2)*B(I)
302 FOR L=1 TO I2-I1-1: LET J=I-L
303 LET R(I)=R(I)+A(I2-L)*B(I-L)
304 LET R(I-L)=R(I-L)+A(I2-L)*B(I)
305 NEXT L: NEXT I: RETURN
```

Przed wywołaniem tego podprogramu należy do tablicy **A(.)** podstawić oryginalną macierz **A**.

Zaprezentowane podprogramy umożliwiają rozwiązanie maksymalne:

N = INT ((RAMTOP — D + 5*M↑2/2 — 5*M/2 — 60) / / (10 — 5*M))

równań, gdzie **D** oznacza obszar pamięci operacyjnej, w bajtach, zajmowanej przez cały program. Na przykład, dla półpasma o szerokości **M = 40** oraz dla **D = 10 KB**, z powyższego równania otrzymuje się **N = 280 równań.**

Podprogramy są również efektywne czasowo, gdyż układ 10 równań przy półpaśmie o szerokości 4 (tabele 1 i 2) jest rozwiązywany na ZX Spectrum ok. 7,1 sekundy, a po zastosowaniu kompilatora zmiennoprzecinkowego FP 48 K V1,7 ok. 1,8 sekundy.

Należy zauważyć, że dla kolejnych wektorów prawych stron rozwiązanie układu równań zaczyna się od wywołania **GOSUB 259.**

LITERATURA

[1] Westlake I. R.: A Handbook of Numerical Matrix Inversion and Solution of Linear Equations. Wiley and Sons, New York, 1968

[2] Wilkinson J. H.: Błędy zaokrągleń w procesach algebraicznych. PWN, Warszawa, 1967.

ZBIGNIEW SZKARADNIK
Gliwice

Rekurencja w języku Forth

Zdolność do definiowania procedur rekurencyjnych [1]) jest uważana przez wielu informatyków za bardzo istotną cechę języka programowania. Pokaźna grupa funkcji matematycznych, a także operacji o charakterze nienumerycznym może być bardzo elegancko zdefiniowana przy użyciu opisu rekurencyjnego [3]. W niektórych językach programowania (Lisp, Algol, Pascal, C) istnieje możliwość stosowania procedur rekurencyjnych. W innych (Basic, Fortran) jest to formalnie niedopuszczalne, choć czasami możliwe.

A jaki jest pod tym względem Forth? Podręczniki języka [1] milczą na ten temat. Naturalne użycie rekurencji jest niedopuszczalne ze względu na proces „zasłaniania" (ang. smudging), który czyni nazwę słowa niewidoczną w czasie jego definicji. Tym niemniej cechy wirtualnej maszyny realizującej Forth (stos danych i stos powrotów), wprost zachęcają do użycia tego mechanizmu.

```
: thiscode  current @ @ pfa cfa ! immediate

: factorial
  dup 0= if
        drop 1
      else
        dup 1- thiscode literal execute *
      endif
```

Wydruk 1

Jedno z rozwiązań tego problemu podał Glyn Emery [2]. Opiera się ono na użyciu kompilatora THISCODE umieszczającego na stosie adres pola kodu aktualnie definiowanego słowa. Rekurencyjną definicję funkcji silnia (ang. factorial), zrealizowaną tą metodą, przedstawiono na wydruku 1.

```
: factorial
  dup 0= if
        drop 1
      else
        dup 1- [ smudge ] factorial [ smudge ] *
      endif
;
```

Wydruk 2

Osobiście stosuję inną metodę. Aby można było skompilować adres słowa aktualnie definiowanego, wystarczy na chwilę „odsłonić" go (słowem SMUDGE), zawieszając na ten okres kompilację, a po skompilowaniu ponownie „zasłonić", w celu umożliwienia poprawnego zakończenia definicji. Rekurencyjną definicję silni zrealizowaną przy użyciu tej metody, przedstawiono na wydruku 2. Można ją dodatkowo uprościć, definiując słowo SMUDGE jako IMMEDIATE (wydruk 3).

```
: ! smudge ! immediate

: factorial
  dup 0= if
        drop 1
      else
        dup 1- ! factorial ! *
      endif
```

Wydruk 3

[1]) Rekurencją nazywa się zdolność do definiowania procedury (bezpośrednio lub pośrednio) przy użyciu niej samej. Klasycznym przykładem rekurencji jest definicja funkcji silnia, którą można przedstawić następująco:

$$n! = \begin{cases} 1 & \text{dla } n = 0 \\ (n-1)!*n & \text{dla } n > 0 \end{cases}$$

```
: assign [compile] ' cfa [compile] ' ! ;

: factorial 0 ;

: factorial'
  dup 0= if
        drop 1
      else
        dup 1- factorial *
      endif
;

assign factorial' factorial
```

Wydruk 4

. Istnieje również i trzecia metoda [2] realizowania rekurencji. Jeżeli należy rekurencyjnie zdefiniować słowo A, to definiuje się w tym celu słowo A, nie wywołujące żadnej akcji. Następnie definiuje się słowo A" wywołujące słowo A, po czym, przy użyciu słowa ASSIGN, przyporządkowuje się słowo A" słowu A. Odbywa się to przez wpisanie adresu cfa słowa A" w pole pfa słowa A. Wydruk 4 przedstawia definicję funkcji silnia zrealizowaną w ten sposób. Warto zauważyć, że metoda ta nadaje się również do realizacji pośredniej rekurencji.

Względne czasy realizacji słów factorial (wydruki 1, 2, 3, 4) wynoszą odpowiednio 1, 12/1, 0/1, 0/1, 16.

LITERATURA

[1] Brodie L.: Starting FORTH. Prentice-Hall, Englewood Cliffs (NJ), 1981

[2] Emery G.: Recursion in FORTH. Dr Dobbs Journal, No. 54, February 1982

[3] Wirth N.: Algorytmy + Struktury danych = Programy. WNT, Warszawa, 1980.

MICHAŁ KLEIBER
MACIEJ LEŚNY
ROMUALD SZUNIEWICZ
Warszawa

Komputery osobiste w zastosowaniach profesjonalnych (2)

Do najciekawszych programów, których rozwój następuje równolegle z rozwojem sprzętowym komputerów osobistych, należą programy graficznego przedstawiania wyników obliczeń, programy przechowywania i wyszukiwania informacji oraz pakiety zintegrowane.

GRAFICZNE PRZEDSTAWIENIE WYNIKÓW OBLICZEŃ

Graficzne przedstawienie wyników obliczeń jest podstawą dużej grupy programów, bardzo przydatnych w zastosowaniach profesjonalnych. Danymi wejściowymi do tych programów są zbiory danych lub zależności funkcyjne, a wyniki przedstawiane są w postaci wykresów liniowych, kołowych, słupkowych, a także przestrzennych, w różnym układzie i różnej formie graficznej. W niektórych programach istnieje możliwość wygładzania przebiegu krzywych, według różnych algorytmów i rysowania linii regresji.

PROGRAMY PRZECHOWYWANIA I WYSZUKIWANIA INFORMACJI

Przykładem możliwości przechowywania i wyszukiwania informacji w zastosowaniach profesjonalnych, bez potrzeby głębokiej znajomości problematyki baz danych, jest program 1-2-3. Wiersze i kolumny tablicy można interpretować jako rekordy i pola zbioru danych (pliku). Można więc w ten sposób — na przykład — zapisać 256 różnych informacji dotyczących 2 tys. osób. Program 1-2-3 umożliwia sortowanie danych w układzie alfabetycznym lub chronologicznym oraz wyszukiwanie rekordów spełniających podane przez użytkownika warunki w poszczególnych polach (na przykład, wyszukanie wszystkich osób z wyższym wykształceniem i znajomością określonego języka).

PAKIETY ZINTEGROWANE

Istotną sprawą przy korzystaniu z różnych programów tekstowych, programów obliczeń tablicowych i programów graficznych jest możliwość wymiany zbiorów między tymi programami. Zbiory te budowane są jednak w różny sposób w poszczególnych programach. Najbardziej popularnymi strukturami zbiorów danych są DIF (program Visicalc), SYLK (program Multiplan) i WKS (program 1-2-3). Niektóre programy pozwalają korzystać z różnych struktur zbiorów.

Problemy przenoszenia informacji między różnymi zastosowaniami rozwiązane są w pakietach zintegrowanych, w których różne programy opierają się na jednolitych strukturach zbiorów. Oprócz przetwarzania tekstów, obliczeń tablicowych i graficznej prezentacji wyników, pakiety zintegrowane mają na ogół programy przechowywania i wyszukiwania informacji (bazy danych) oraz programy komunikacyjne (do przekazywania informacji między użytkownikami lub komputerami).

Obok tych podstawowych programów, profesjonalni użytkownicy mikrokomputerów mogą korzystać z dużej liczby programów specjalizowanych (np. w zakresie planowania przedsięwzięć, analizy statystycznej, księgowości, zarządzania bazami danych) oraz programów pomocniczych (np. służących jako kalendarz, notatnik, kalkulator, wywoływanych na ekran w postaci tzw. okien podczas pracy z innymi programami). Trzeba też wymienić programy wspomagania decyzji, a wśród nich systemy ekspertowe, które znajdują zastosowanie m. in. w medycynie.

JĘZYKI PROGRAMOWANIA

W coraz większej liczbie zastosowań użytkownik zwolniony jest z konieczności opracowywania własnych programów i może skoncentrować się na merytorycznej stronie problemów. Jeśli jednak oferowane programy nie spełniają specjalistycznych wymagań użytkownika, wówczas może on zapisać i rozwiązać swój problem przy użyciu jednego z kilkudziesięciu języków programowania, których kompilatory są dostępne na mikrokomputerach. Obok najbardziej popularnych języków wysokiego poziomu, takich jak: Basic, Fortran, Cobol, Pascal, APL i języków niskiego poziomu, jak makroasemblery lub C, użytkownik mikrokomputerów ma też do dyspozycji proste języki specjalizowane, oferowane do konkretnych zastosowań. Odpowiednim przykładem może być zbiór komend jednego z najbardziej

popularnych i uniwersalnych programów relacyjnych baz danych — dBase II firmy Ashton-Tate.

Rozliczne zastosowania mikrokomputerów bezpośrednio dostępne rzeszom użytkowników, nie mających żadnego przygotowania informatycznego, nie powinny stwarzać wrażenia niemożności wykorzystywania mikrokomputerów do rozwiązywania złożonych zagadnień naukowo-technicznych, realizowanych do niedawna wyłącznie na dużych maszynach cyfrowych. Sytuacja jest bowiem wręcz przeciwna — wykonywanie na mikrokomputerach wielu zadań, wymagających stosowania języków programowania, jest jak najbardziej możliwe, stwarzając niekiedy znacznie mniej kłopotów w porównaniu z pracą na sprzęcie tradycyjnym.

Na rynku dostępne są obecnie w wersjach mikrokomputerowych praktycznie wszystkie znane kompilatory i interpretatory języków programowania, mające znaczenie użytkowe. Istnieją możliwości pracy w tych językach pod nadzorem większości typowych dla współczesnych mikrokomputerów systemów operacyjnych, takich jak CP/M czy DOS.

Należy podkreślić, że intensywny rozwój oprogramowania mikrokomputerów dla potrzeb naukowo-technicznych rozpoczął się stosunkowo niedawno (2—3 lata temu). Przyczyną tego był oczywiście fakt, że początkowo użytkownicy tego rodzaju stanowili zdecydowaną mniejszość w porównaniu z rynkiem gier i oprogramowania na użytek małych i średnich przedsiębiorstw. W ostatnich 2—3 latach rozwój sprzętu mikrokomputerowego, a szczególnie upowszechnienie pamięci RAM (o pojemnościach co najmniej 0,5 MB) oraz pamięci na dyskach sztywnych (o pojemności rzędu 10—20 MB) stworzył zasadniczo nową sytuację. Możliwości zastosowań naukowo-technicznych okazały się tak duże, że opracowanie kompilatorów języków pod kątem takich właśnie zastosowań stało się bardzo atrakcyjne finansowo i błyskawicznie przyciągnęło wiele, zarówno renomowanych, jak i nowych firm produkujących oprogramowanie.

Istniejące obecnie oprogramowanie osiągnęło już znaczny stopień doskonałości. Obecnie obserwowany jego dalszy lawinowy rozwój pozwala upewnić się co do wiarygodności i tendencji rozwojowej w dziedzinie komputeryzacji prac naukowo-technicznych.

Jedną z najlepszych ilustracji możliwości zastosowań mikrokomputerów w technice jest rozwój oprogramowania dotyczącego tzw. metody elementów skończonych. Metoda ta jest powszechnie uważana za najbardziej wszechstronną i skuteczną metodę rozwiązywania dużych układów liniowych i nieliniowych równań różniczkowych. Jako taka, metoda elementów skończonych znalazła szerokie zastosowanie w zagadnieniach typowych dla wielu problemów inżynierskich, jak np. problemy mechaniki ciał odkształcalnych, mechaniki cieczy i gazów, przewodnictwa cieplnego, pól sprzężonych, projektowania układów konstrukcyjnych itp. Metoda elementów skończonych jest wykorzystywana od ok. 20 lat; istnieje i jest w praktycznym użyciu wiele programów (napisanych prawie wyłącznie w Fortranie) o ogólnym lub problemowym charakterze, uruchamianych na wszystkich możliwych typach komputerów. W początkowej fazie rozwoju mikrokomputerów zaczęto tworzyć oprogramowanie metody elementów skończonych w jedynym dostępnym wtedy języku, tj. Basicu. Ze względu na brak oprogramowania pomocniczego oraz wady Basica — związane z trudnością tworzenia dużych programów o modułowej strukturze — powstające w ten sposób programy nie gwarantowały dostatecznej efektywności. Pojawienie się Fortranu w wersji mikrokomputerowej zasadniczo zmieniło sytuację. Najnowsza wersja tego języka o nazwie MS.Fortran 3.30 umożliwia tworzenie oprogramowania w sposób nie odbiegający od metod wypracowanych przez lata doświadczeń z dużymi komputerami. Powstające oprogramowanie mikrokomputerów w zakresie metody elementów skończonych ma olbrzymi potencjał użytkowy — już dzisiaj na IBM PC, z pamięcią RAM o pojemności 640 KB, można rozwiązywać skomplikowane zadania z dziedziny nieliniowej mechaniki konstrukcji, otrzymując rozwiązania mające bezpośrednią przydatność praktyczną. Posiadanie sztywnego dysku zwiększa nasze możliwości do poziomu porównywalnego z osiąganym na największych komputerach produkowanych w kraju.

Istniejące kompilatory Fortranu nie są z pewnością ostatecznym osiągnięciem. Sytuację tę zilustrowano w tabeli, wskazując na różnice czasów osiąganych w trakcie wykonania prostych zadań arytmetycznych przez różne konkurencyjne wersje kompilatorów Fortranu.

Porównanie czasu obliczeń (w sekundach) na podstawie programu generowania liczb pierwszych z wykorzystaniem operacji zmiennoprzecinkowych

	MS. FORTRAN	SS. FORTRAN	IBM FORTRAN
Bez koprocesora 8087	46	18	429
Z koprocesorem 8087	3	13	—

Mówiąc o językach stosowanych do obliczeń numerycznych, nie można ominąć znaczenia i wpływu na efektywność procesu obliczeniowego, jakie ma tzw. koprocesor arytmetyczny 8087. Koprocesor ten, zwany także zmiennoprzecinkowym, kosztuje obecnie ok. 150 dolarów i może być bez trudu zainstalowany w IBM PC. Wpływ tego procesora na przyspieszenie procesu obliczeń zilustrowano także w tabeli. Można bezpiecznie założyć, że obecność koprocesora 8087 przyspiesza wykonanie przeciętnego programu obliczeniowego, wykorzystującego operacje zmiennoprzecinkowe o rząd wielkości — autorzy dysponują przykładami, z których wynika, że zwiększenie efektywności programu może być niekiedy nawet stukrotne.

Mówiąc o metodzie elementów skończonych, należy wspomnieć o zastosowaniach mikrokomputerów w niezwykle aktualnej dziedzinie prac inżynierskich, zwanej projektowaniem wspomaganym komputerowo (CAD). Metoda elementów skończonych jest bowiem często jednym z zasadniczych modułów w nowoczesnym podejściu do projektowania. W systemie CAD podstawową rolę odgrywa mikrokomputer, który umożliwia interaktywną pracę związaną z dokonaniem wstępnych obliczeń projektowych, przeprowadzeniem modyfikacji i optymalizacji projektu wstępnego, wykonaniem kompletnych obliczeń i sporządzeniem wymaganej dokumentacji. Obecny poziom rozwoju mikrokomputerów umożliwia tworzenie zautomatyzowanych stanowisk projektowania (tzw. work stations), mających perspektywicznie fundamentalne znaczenie dla efektywności projektowania we wszystkich dyscyplinach techniki. Tworzenie oprogramowania do CAD jest obecnie z pewnością jednym z najbardziej potrzebnych i opłacalnych działań w komputeryzacji prac naukowo-technicznych.

Oddzielną dziedziną, której nawet skrótowe omówienie wymagałoby osobnego artykułu, jest wspomagane komputerowo wytwarzanie (CAM ang. Computer-Aided Manufacturing).

Gwałtowny rozwój naukowo-techniczny zastosowań mikrokomputerów następował głównie w obszarach zdominowanych przez języki do obliczeń numerycznych, takie jak Fortran czy Pascal. Należy zdawać sobie sprawę, że w ramach możliwości odpowiednich kompilatorów tych języków cała praca koncepcyjna oraz pewna część obliczeń musi być realizowana w sposób ręczny. Dotyczy to w szczególności obliczeń realizowanych na symbolach typu różniczkowania czy całkowania. Uzyskane drogą ręczną formuły są wykorzystywane w ramach oprogramowania, np. fortranowskiego, do obliczania ich wartości dla konkretnych układów danych liczbowych. Zasadniczą, jakościową zmianę systemów pracy z mikrokomputerem osiąga się wykorzystując języki do przetwarzania informacji symbolicznych. Języki te już niedługo odgrywać będą istotną rolę w wielu zagadnieniach współczesnej nauki i techniki, takich jak: problematyka sztucznej inteligencji, robotyka, analiza języków naturalnych, zasady łamania i budowy szyfrów, tworzenie różnorodnych programów samodoskonalących się itp.

Dla mikrokomputera IBM PC istnieją obecnie możliwości wykorzystywania języków symbolicznych, takich jak: LISP, PROLOG czy muSIMP. Głównym celem stosowania tych języków jest prowadzenie obliczeń numeryczno-symbolicznych lub czysto symbolicznych. W obliczeniach naukowo-technicznych języki te pozwalają na:

- wykonywanie obliczeń i operacji na wyrażeniach arytmetycznych w zakresie dodawania, odejmowania, mnożenia, dzielenia, potęgowania, pierwiastkowania, operacji na

funkcjach trygonometrycznych i wielomianach, rachunku różniczkowego i całkowego, rachunku macierzowego itp.,
- definiowanie przez użytkownika własnych operacji symbolicznych, realizujących niestandardowe obliczenia matematyczne.

Nowe podejście do formułowania i rozwiązywania problemów przy użyciu komputerów znajduje odbicie w klasyfikacji języków programowania (np. zaproponowanej przez Alana Kay'a, Scientific American, 1984, Vol. 251, No. 3, p. 52), według której języki asemblerowe są zaliczane do języków niskiego poziomu, FORTRAN czy ALGOL — do języków wysokiego poziomu, PROLOG — do języków bardzo wysokiego poziomu, a programy obliczeń tablicowych stanowią formę języków ultra wysokiego poziomu.

Przy pisaniu tego artykułu autorzy opierali się na własnym doświadczeniu nabytym przy użytkowaniu komputerów osobistych w rozwiązywaniu problemów ekonomicznych, naukowych i technicznych oraz redagowaniu tekstów, a także na literaturze firmowej i bieżących numerach czasopism poświęconych komputerom osobistym. Artykuł został zredagowany na takich komputerach, a wymiana fragmentów tekstu między autorami odbywała się za pomocą dyskietek. Tekst o objętości ok. 23,5 strony maszynopisu (30 wierszy po 60 znaków) zajmuje ok. 42 KB pamięci.

JERZY KARCZMARCZUK
Instytut Fizyki
Uniwersytet Jagielloński
Kraków

Język programowania Icon (2)

W tej części artykułu omówiono proste i złożone struktury danych oraz najbardziej charakterystyczne mechanizmy Iconu, jakim są generatory.

PIERWOTNE STRUKTURY DANYCH

Operacje liczbowe w Iconie są podobne do spotykanych w innych językach programowania. Icon ma wbudowane operatory pięciu działań na liczbach całkowitych i rzeczywistych (zmiennopozycyjnych). Konwersja liczb całkowitych na rzeczywiste jest automatyczna. Symbolem dzielenia jest znak backslash „\" [1]), zarówno w przypadku całkowitym, jak i rzeczywistym. Wynik dzielenia jest całkowity (dzielenie z obcięciem), jeśli oba argumenty są całkowite, a rzeczywisty w przeciwnym wypadku. Resztę z dzielenia całkowitego można otrzymać stosując operator reszty oznaczany znakiem „%". Jedyną wbudowaną funkcją liczbową jest abs — udostępniająca wartość bezwzględna argumentu.

Można operować liczbami całkowitymi w dowolnym systemie pozycyjnym od 2 do 36. Cyfry większe od 9 są reprezentowane przez litery a, b, c ... z. Stałe w dowolnym systemie piszemy w postaci: **irj**, gdzie **i** jest liczbą dziesiętną oznaczającą bazę, zaś **j** — ciągiem cyfr o tej bazie. Tak więc **2r1101** ma wartość dziesiętną 13, zaś **36rcat** — wartość 15941.

Operatory porównania liczb są dość typowe $<$, $<=$, $=$, $>=$, $>$, za wyjątkiem operatora nierówności $\sim=$, który różni się oznaczeniem od częściej spotykanego $\langle\rangle$. Jak łatwo się domyślić, w przypadku niespełnienia sprawdzanego warunku, wynikiem operacji jest niepowodzenie. Interesującą cechą Iconu jest jednak semantyka tych operacji, w przypadku powodzenia. Wtedy wynikiem jest wartość prawego argumentu. Przykładowo, następujące wyrażenie złożone:

ALFA < = BETA < GAMMA

ma prostą, intuicyjną interpretację: jego ewaluacja zakończy się powodzeniem, jeśli wartość zmiennej BETA jest nie mniejsza od ALFA, a mniejsza od GAMMA. Zarówno operatory arytmetyczne, jak i operatory relacji mogą wystąpić w rozszerzonej operacji przypisania, np.:

ALFA > := BETA

przypisanie zmiennej ALFA wartości BETA tylko wtedy, gdy relacja przed przypisaniem była spełniona.

ZBIORY ZNAKÓW I NAPISY

W odróżnieniu od Snobolu, który dopuszcza pewną dowolność, Icon operuje zestawem 256 znaków, niezależnie od komputera. Pierwsze 128 znaków jest zgodne z kodem ASCII. Znakowy typ danych (char lub character) nie występuje, natomiast istnieją dwa inne typy danych złożonych ze znaków.

Pierwszym typem są tzw. C-zbiory (ang. csets), pełniące rolę teoriomnogościowych zbiorów znaków, w których kolejność lub powtarzanie się znaków nie ma znaczenia. W tekście programu stałe tego typu zapisuje się jako ciągi znaków w apostrofach, np. 'aeiou', co jest równoważne m. in. 'iaeouia'. Znaków nie mających odpowiedników tekstowych oraz apostrofu można używać wewnątrz C-zbiorów, wykorzystując specjalny znak „!/" (escape character). C-zbiory służą do przeszukiwania tekstów pod względem wystąpienia pojedynczych znaków, w naturalny sposób reprezentują też relacje między znakami, które najwygodniej opisuje się jako przynależność znaku do odpowiedniego C-zbioru. Icon ma kilka wbudowanych C-zbiorów, które są wartościami pewnych słów kluczowych. Słowo &cset odpowiada zbiorowi wszystkich 256 znaków, &ascii zawiera 128 pierwszych znaków kodu ASCII, a &lcase i &ucase zawierają odpowiednio małe i duże litery.

Istnieje 5 wbudowanych operacji na C-zbiorach. Wartością wyrażenia *C jest rozmiar c-zbioru o nazwie C, tj. liczba znaków zawarta w C. Wartością $\sim$C jest jego uzupełnienie do pełnego zbioru &ceset. Wyrażenia C1**C2, C1+ +C2 i C1— —C2 oznaczają przecięcie, sumę i różnicę zbiorów C1 i C2.

Strukturą danych częściej używaną niż C-zbiory są napisy. Ich sekwencyjna i uporządkowana struktura pozwala na wyrażenie znacznie bardziej skomplikowanych relacji między znakami. Stałe, będące napisami, zapisuje się jako ciągi znaków ograniczone cudzysłowami. Napis "aeiou" jest istotnie różny od "iaeouia". Konwersja zbiorów na napisy i odwrotnie, na ogół nieodwracalna, jest w miarę potrzeby wykonywana automatycznie.

Jednoargumentowy operator * służy do obliczania długości napisu. Najczęściej spotykaną operacją dwuargumentową jest spinanie (konkatenacja) oznaczana symbolem „||". Wartością wyrażenia „witaj" || „smutku" jest napis „witajsmutku". Napisy są pierwotnym typem danych, ale

[1]) Drukarnia nie ma w składzie znaku backslash (lewa kreska ukośna), który w tym artykule zastępujemy symbolem !/.

mogą być również uważane za jednowymiarowe tablice znaków. Można z nich wyodrębniać pojedyncze znaki, a także wycinki przy użyciu indeksowania. Jeśli wartością zmiennej X jest „witajsmutku", to wartością X[3] jest „t", a wartością X[6 : 11] napis „smutku".

Aby właściwie zrozumieć sposób indeksowania napisów i inne operacje związane z ich przeszukiwaniem, należy zdać sobie sprawę, że podstawowym pojęciem nie jest tu wartość indeksu znaku, lecz pozycja znaku, której obrazowo odpowiada strzałka umieszczona między znakami. Numer pozycji przed pierwszym znakiem napisu jest równy 1, zaś numer pozycji za ostatnim znakiem umownie wynosi zero. Charakterystycznym rozszerzeniem systemu indeksowania jest możliwość oznaczania pozycji lub indeksów liczbami ujemnymi. W Iconie przyjęto konwencję, według której ujemne indeksy oznaczają odliczanie od końca napisu w lewo. X[—1] jest ostatnim znakiem X, a wyrażenie X[—5 : 0] ma również wartość "smutku".

Wyrażenia oddzielone dwukropkiem nie muszą występować w kolejności [mniejsze : większe]. Dozwolona jest dowolna kolejność i oznacza to samo. Można stosować również indeksowanie względne, wykorzystując operatory: +: (w prawo) oraz —: (w lewo). Wartością wyrażenia X[3 +: 5] jest wycinek o długości 5 znaków, począwszy od trzeciego znaku X. Przekroczenie granic napisu przez wyrażenie indeksowe nie jest błędem, lecz powoduje niepowodzenie. Użyteczny w niektórych zastosowaniach może być jednoargumentowy operator oznaczany znakiem zapytania, będący generatorem losowego znaku. Wartością wyrażenia ?X jest jednoznakowy napis losowo wybrany z X.

Napisy mogą być ze sobą porównywane i porządkowane leksykograficznie. Operatory odpowiednich relacji są oznaczone symbolami $\ll$, $\ll=$, ==, $\gg=$, $\gg$ oraz $\sim==$. Poniższy program drukuje leksykalne maksimum i minimum przeczytanych napisów, po jednym w wierszu.

```
procedure main ()
    min := max := read ()
    while line := read () do
        if max ≪ := line then next
        else min ≫ := line
    write ("max = ",max)
    write ("min = ",min)
end
```

FUNKCJE OPERUJĄCE NAPISAMI

Funkcje operowania napisami można podzielić na dwie klasy: funkcje formatowania i funkcje przeszukiwania. Typową funkcją formatowania jest left. Wywołanie:

left(s1, i, s2)

gdzie wartościami s1 i s2 są napisy, a i jest liczbą całkowitą, powoduje dostarczenie lewego wycinka s1 o długości i — jeśli napis s1 jest dłuższy. W przeciwnym wypadku następuje uzupełnienie napisu s1 z prawej strony o znaki należące do s2. Opuszczenie trzeciego argumentu jest równoważne spacji.

Podobnie działają i podobnie są wywoływane funkcje **right** oraz **center**. Wywołanie funkcji **trim (c, s)** powoduje dostarczenie lewego wycinka s po odcięciu wszystkich znaków należących do C-zbioru c. Wywołanie funkcji **repl (s, i)** powoduje powielenie napisu s i-krotnie, zaś **reverse(s)** odwrócenie napisu s. Ostatnią funkcją formatowania jest odpowiednik REPLACE w Snobolu. Wywołanie **map (s1, s2, s3)** powoduje dostarczenie napisu powstałego z s1 przez wymianę wszystkich jego znaków, które znajdują się w napisie s2, na znaki w napisie s3, zajmujące tę samą pozycję. Przykładowo:

map ("12345678", "87654321", s)

gdzie wartością s jest dowolny napis 8-znakowy, jest równoważne wywołaniu **reverse(s)**.

Ważniejsze z punktu widzenia analizy tekstów są funkcje, które pozwalają w danym napisie znaleźć określony znak lub dłuższy fragment. Wynikiem wywołań wszystkich omówionych wyżej funkcji jest liczba będąca numerem pozycji. Podstawowymi funkcjami służącymi do wyszukiwania wyników napisów są find oraz match. Wywołanie **find(s1, s2)** powoduje dostarczenie numeru pozycji w s2, od której rozpoczyna się wycinek równy s1. W wypadku większej liczby wystąpień s1, wartością dostarczaną jest pierwszy znaleziony numer pozycji. Funkcja **find** jest generatorem i pewne szczegóły jej działania zostaną omówione później. Jeśli wycinek s1 nie zostanie znaleziony, to wywołanie **find** zakończy się niepowodzeniem. Wywołanie funkcji **match(s1, s2)** kończy się powodzeniem, jeśli s1 jest początkowym wycinkiem napisu s2. Dostarczany jest wtedy numer pozycji w napisie s2, po ostatnim znaku należącym do s1. Te funkcje, a także inne omówione poniżej, dopuszczają dwa dodatkowe argumenty. Wywołanie **find(s1, s2, i1, i2)** ogranicza zakres przeszukiwania napisu s2 do pozycji zawartych między i1 a i2.

Do wyszukiwania pojedynczych znaków wygodniejsze są inne funkcje: **upto, any** oraz **many**. Jeśli napis s zawiera znak lub znaki należące do C-zbioru c, to wywołanie **upto(c, s)** powoduje dostarczenie numeru pozycji w s przed pierwszym znalezionym znakiem. Jeśli pierwszy znak s należy do c, to wywołanie **any(c, s)** powoduje dostarczenie numeru pozycji po tym znaku. Funkcja **any** przypomina więc funkcję **match**. Dostarczany wynik na ogół nie jest interesujący, ważne jest natomiast, czy wywołanie kończy się powodzeniem. Rozszerzeniem funkcji **any** jest funkcja **many**, wywoływana podobnie. Jeśli napis s zaczyna się od znaków należących do C-zbioru c, to wywołanie **many** powoduje dostarczenie numeru pozycji po odpowiednim wycinku. Funkcja **many** jest więc w pewnym sensie odwrotnością funkcji **upto**.

Przykładem wykorzystania tych funkcji jest program, który odczytuje wiele wierszy tekstu i przedrukowuje początkowe słowa każdego wiersza, tj. wycinki złożone wyłącznie z małych i dużych liter.

```
procedure main ()
    wchar :=&lease ++&ucase
    while line := read () do
    if line := line[upto(wchar, line) : 0]
    then write(line[1 : many(wchar,line)])
end
```

Należy wspomnieć także o funkcji pomocnej w analizie i przetwarzaniu wyrażeń algebraicznych i innych napisów, które zawierają pary znaków pełniących rolę nawiasów, np. (...), [...] itp. Wywołanie funkcji **bal(c1, c2, c3, s)** przypomina nieco **upto(c1, s)**, przy czym c2 i c3 określają nawias otwierający i zamykający, między którymi znak należący do c1 nie zostanie znaleziony w s. Przykładowo, wyrażenie:

bal('+', '(', ')', "((2*X) +3)+(5*Y)")

powoduje dostarczenie liczby 10.

ZŁOŻONE STRUKTURY DANYCH

Icon ma dość uniwersalne struktury danych, które w zależności od sposobu dostępu użytego przez programistę mogą pełnić rolę list, rozszerzalnych i „rozciągliwych" tablic, stosów lub kolejek.

Listę tworzy się ujmując w nawiasy kwadratowe ciąg wyrażeń, oddzielonych przecinkami, np.:

["Caesar","in","urbe","sua","deus","est"]

Wartości wyrażeń mogą być dowolnego typu, w szczególności mogą być listami. Jednoargumentowy operator gwiazdki służy do określania aktualnej długości listy. Indeksowanie list, a więc traktowanie ich jak jednowymiarowych tablic, odbywa się w typowy sposób. Można również używać konstrukcji z dwukropkiem, w celu wyodrębnienia wycinków list. Operator spinania list ma postać |||. Listy można traktować jak stosy lub kolejki, wykorzystując następujące funkcje wbudowane: **put, get, push** i **pull**. Wywołanie **put(l, x)** powoduje dołączenie wartości x do końca listy l, zwiększając jej długość o 1. Wyrażenie **get(l)** dostarcza wartości pierwszego elementu listy l, jednocześnie usuwając go z listy. Z kolei funkcja **push(l, x)** dołącza

wartość x do początku listy l, a **pull(l)** usuwa ostatni element listy. Icon ma również wbudowaną funkcję sortowania listy.

Tabele

Tabela jest typową asocjacyjną strukturą danych, z którą mamy do czynienia również w Snobolu. Składniowo odpowiada jednowymiarowej tablicy, przy czym indeks nie musi być liczbą całkowitą, lecz wartością dowolnego typu, np. napisem, a nawet tabelą. Tabelę (początkowo pustą) tworzy się wywołując wbudowaną funkcję **table(x)**. Wartość x jest wartością początkową nadawaną elementom tabeli w chwili ich tworzenia i służy jako wartość zastępcza przy odwołaniu do nieistniejącego elementu. Elementy tabeli tworzy się przypisując im wartości.

Załóżmy, że procedura **readword** dostarcza kolejnego słowa ograniczonego znakami interpunkcyjnymi (czytelnik może w ramach ćwiczenia sam napisać tę procedurę). Program:

```
procedure main ( )
    wds := table (0)
    while wds[readword( )] + := 1
end
```

zlicza liczbę wystąpień różnych słów w tekście. Program jest niekompletny, gdyż należałoby jeszcze zamienić tabelę wds na listę, posortować ją i wydrukować.

Rekordy

Rekordy w Iconie są nieco podobne do list, lecz zawierają stałą liczbę elementów. Dostęp do elementów nazywanych polami umożliwia standardowa notacja kropkowa, choć rekordy w Iconie można również indeksować kolejnym numerem pola w deklaracji rekordu. Deklaracje są globalne i nie są zanurzane w procedurze. Przykładem zastosowania rekordów jest funkcja, operująca liczbami zespolonymi:

```
record complex(re, im)
procedure mult(z1, z2)
    return complex(z1.re*z2.re — z1.im*z2.im,
                   z1.re*z2.im + z1.im*z2,re)
end
```

Nazwa rekordu jest wykorzystywana jako konstruktor w tworzeniu nowych rekordów danego typu, a nazwy pól służą jako selektory.

GENERATORY

Generatory są najbardziej charakterystycznym mechanizmem sterowania w Iconie, którego brak w innych popularnych językach programowania, choć występuje w Prologu (stanowiąc jego istotę), w Snobolu (podczas operacji porównywania napisu z wzorcem), a także w wyspecjalizowanych systemach, jak np. PLANNER. Mówiąc krótko, polega on na tym, że można odmówić przyjęcia wartości obliczonej przez wyrażenie i zażądać dostarczenia następnej, jeśli jest to możliwe.

W celu zilustrowania tego mechanizmu posłużę się analogią wojskową. Oficer na wojnie, aby posunąć się naprzód, musi zdobyć pewne wzgórze opanowane przez wroga. Czy traktuje to jako zadanie do wykonania, czy jako cel (a raczej — podcel) do osiągnięcia? A może jest mu wszystko jedno? Podejście do zagadnienia może zależeć od rangi oficera. Bezpośrednio kierujący akcją musi ją zaprogramować, natomiast generał na ogół nie tylko nie interesuje się szczegółami, ale nawet liczbą bezskutecznych prób.

Ideą przewodnią Iconu było ułatwienie programowania na „generalskim" poziomie. Przykład bliższy informatyce może polegać na znalezieniu jednego lub wszystkich rozwiązań problemu 8 hetmanów, który polega na takim rozstawieniu ich na szachownicy, aby wzajemnie się nie szachowały. Charakterystyczną, nie wymagającą wyrafinowanych koncepcji ze strony programisty, choć na ogół niezbyt efektywną metodą służącą do tego celu, są nawroty. Rozwiązanie tą metodą polega na składaniu na wszystkie możliwe sposoby wariantowych rozwiązań podproblemów, które w ten sam sposób rozszczepia się dalej. Jeśli dana kombinacja wariantów prowadzi program w ślepą uliczkę, to należy się wycofać, odtworzyć stan programu poprzedzający wybór wariantu i wybrać następny. Zalecana strategia może polegać na wykonaniu wariantu 1, ewentualnie wariantu 2 itd. Takie podejście nosi nazwę niedeterministycznego.

Omawiane dotąd wyrażenia dostarczały określonego wyniku, bądź zawodziły. Niektóre wyrażenia są jednak potencjalnie zdolne do dostarczenia kilku wyników, np. wynikiem wywołania:

```
upto ('r',"czarna krowa w kropki bordo")
```

jest liczba 4. Można jednak spowodować, aby wywołanie **upto** dostarczyło innych numerów pozycji, na których występuje litera r, tj. 9, 17 i 25. Wyrażenia zdolne do dostarczania większej liczby wyników noszą nazwę generatorów, a uporządkowany zbiór tych wyników nazywa się sekwencją generatora. Sekwencja nie istnieje jako obiekt. Jej elementy nie współistnieją, gdyż są generowane kolejno.

Iterator

Wygenerowanie całej sekwencji wyników generatora można spowodować explicite przez użycie struktury sterowania zwanej iteratorem. Jeśli wyrażenie E1 jest generatorem, to konstrukcja

```
every E1 do E2
```

powoduje, że E1 generuje kolejno całą swoją sekwencję, a dla każdego jej elementu ewaluowane jest E2. Przykładowo:

```
every i := upto('aeiou', x) do write i
```

powoduje wydrukowanie wszystkich numerów pozycji samogłosek w napisie będącym wartością x. Właściwość generowania przenosi się z funkcji **upto** na całą operację przypisania, a ogólnie przenosi się na każde wyrażenie zawierające generator. Klauzula E2 jest opcjonalna. To samo zostanie wydrukowane, jeśli użyjemy zapisu:

```
every write(upto('aeiou', x))
```

Konstrukcje generatorów

Icon ma bogaty repertuar operatorów i struktur sterowania tworzących generatory z prostszych elementów.

Najprostszym generatorem złożonym jest alternatywa generatorów złożona z wyrażeń połączonych operatorem |, który nazywa się operatorem wyboru. Wyrażenie E1|E2 generuje sekwencję E1, po której następuje sekwencja E2. Przykladowo:

```
1 | 2 | 5
```

jest generatorem o sekwencji 1, 2 i 5. Elementy tej alternatywy są banalnymi generatorami, których sekwencje są jednoelementowe. Tak należy traktować wszystkie „zwykłe" wyrażenia. Często można wykorzystać alternatywę zamiast instrukcji **if ... then ... else.**

Jednoargumentowy operator | oznacza nieograniczone powtórzenie, np. wyrażenie:

```
| ( 1 | 2 | 5)
```

generuje sekwencję: 1, 2, 5, 1, 2, 5, ...
Długość sekwencji można ograniczyć przez zadaną liczbę. Jeśli E jest generatorem, to wyrażenie:

```
E \ i
```

gdzie wartością i jest dodatnia liczba całkowita, jest generatorem, który po dostarczeniu co najwyżej i wyników ostatecznie zawodzi.

Jedną z najczęściej spotykanych w innych językach konstrukcji jest pętla, nakazująca wykonanie pewnych operacji dla wszystkich liczb całkowitych należących do pewnej sekwencji. W Iconie służą do tego celu generatory. Wyrażenie:

i1 to i2 by i3

generuje wszystkie liczby całkowite od i1 po i2 z krokiem i3. Krok jest opcjonalny, a jego wartość domyślna wynosi 1. Tak więc w Iconie tradycyjną pętlę **for** zapisuje się w postaci **every i := i1 to i2 do ...**
Trzeba jednak mieć świadomość faktu, że iterator jest zupełnie niezależny od generatora.

Dość typowym i bardzo użytecznym mechanizmem jest przetwarzanie listy lub napisu na generator, który generuje wszystkie elementy odpowiedniej struktury danych. Do tego celu służy operator generowania elementu oznaczany znakiem „!". Wyrażenie:

every write(!s)

powoduje wydrukowanie wszystkich znaków napisu **s** po jednym w wierszu, natomiast:

every !x := wyrażenie

wymianę wszystkich elementów **x** (listy lub napisu) na wartości dostarczane przez prawą stronę przypisania. Unika się w ten sposób jawnego indeksowania.

W następnym odcinku omówimy dość nietradycyjny, teleologiczny (ang. goal-oriented) aspekt programowania w Iconie, a także konstruowanie własnych procedur, będących generatorami, takich jak: **find** lub **upto**.

LESZEK KOTULSKI
Katedra Informatyki
Uniwersytet Jagielloński
Kraków

Monitor jako narzędzie strukturalizacji programów współbieżnych (2)

Koncepcja monitora rozwinięta przez Hoare'a [9] i Brinch Hansena [1, 2] została omówiona w poprzedniej części artykułu. Poniżej zajęto się analizą jego funkcjonowania. Jedna z podstawowych właściwości monitora polega na tym, że wykonanie jego procedur jest wzajemnie wykluczone w czasie. To ograniczenie jest warunkiem koniecznym dla zapewnienia poprawnej obsługi danych i zasobów administrowanych przez monitor.

BADANIE POPRAWNOŚCI MONITORA

Z punktu widzenia poprawności programów — wzajemne wykluczanie powoduje, że wywołania procedur są obsługiwane kolejno po sobie, podobnie jak w programowaniu sekwencyjnym, co pozwala zastosować w dowodach poprawności monitora reguły dowodzenia, oparte na asercjach związanych z reprezentacją danych [9]. W przypadku programu sekwencyjnego programista może z daną reprezentacją danych związać niezmiennik początkowy i badać, jak zmienia się on wskutek wykonania kolejnych operacji programu aż do osiągnięcia założonego niezmiennika końcowego. W przypadku monitora zakłada się, że programista musi wyznaczyć początkowy niezmiennik **Ip** wspólny dla wszystkich procedur monitora. Ponieważ każdy niezmiennik końcowy **Ik** staje się jednocześnie niezmiennikiem początkowym dla następnej wywoływanej procedury, to **Ik** musi implikować spełnienie **Ip**. Reasumując, musi istnieć taki niezmiennik **I**, że jest on spełniony po zainicjowaniu danych, a dla każdej należącej do tego monitora procedury **P**, posiadającej atrybut zewnętrzności, musi być prawdziwe:

I{P}I

Zezwolenie wzajemnego wykluczania może nastąpić również wskutek wykonania operacji synchronizujących. Załóżmy, że niezmiennik **I** musi być spełniony bezpośrednio przed wykonaniem tych operacji. Jeżeli ponadto z daną zmienną warunkową **b** programista zwiąże asercję **B**, to wykonanie tych operacji będzie w monitorze Hoare'a [9] opisane przez następujące reguły dowodzenia:

I&B (b.signal) I
I (b.wait) I

Zgodnie z oczekiwaniami proces odwieszający wykona operację **signal** tylko wtedy, gdy będą spełnione warunki do odwieszenia procesu zawieszonego w kolejce zmiennej warunkowej **b**. Semantyka operacji **signal** gwarantuje, że monitor natychmiast podejmie obsługę procesu odwieszonego, czyli że niezmienniki spełnione przed wykonaniem operacji **signal** będą również spełnione przed wykonaniem instrukcji następującej po **wait**. Dokładniej, monitor podejmie obsługę procesu odwieszającego dopiero wtedy, gdy zakończy on obsługę procesu odwieszonego lub ponownie zawiesi ten proces. W obu przypadkach można stwierdzić, że na pewno spełniony jest jedynie podstawowy niezmiennik monitora **I**.

Omówiona metoda pozwala zbadać zgodność specyfikacji monitora z jego implementacją. Pomijając znane trudności przeprowadzania dowodów asercyjnych, należy stwierdzić, że użyteczność tej metody maleje z powodu braku możliwości wykluczenia przy jej użyciu ryzyka wystąpienia zakleszczenia. Tak więc, nie zapewnia ona możliwości całkowitej weryfikacji poprawności monitora.

MODULARYZACJA SYSTEMU

Korzyści metodyczne i praktyczne wynikające z modularyzacji dużych programów sekwencyjnych są dość oczywiste. Prawdopodobnie nikogo nie trzeba przekonywać o potrzebie zastąpienia monolitycznego monitora przez hierarchiczną strukturę monitorów [21, 22]. Pewnego uzasadnienia wymaga jedynie stwierdzenie, że wszystkie moduły zastępujące monitor monolityczny muszą być również monitorami oraz założenie o hierarchicznej budowie struktury monitorów. W pierwszym przypadku, gdy dany moduł jest monitorem, to można odwoływać się do niego z kilku monitorów lub procesów. Drugim argumentem jest łatwość testowania systemów hierarchicznych, jak również możliwość uniknięcia zakleszczenia przy cyklicznym wywoływaniu monitorów (gdy operacja wywołania monitora nie zwalnia wzajemnego wykluczania).

IMPLEMENTACJA ZAGNIEŻDŻONYCH WYWOŁAŃ MONITORÓW

Wiele kontrowersji wzbudza sposób implementacji operacji wywołania procedury monitora z innego monitora, przy czym dotyczą one rozwiązania problemu wzajemnego wykluczania. Należy zawsze podjąć decyzję czy monitor, który wywołał procedurę innego monitora, musi być bezczynny w czasie, gdy wywołany monitor obsługuje to wywołanie. Wykonanie operacji **wait** może zawiesić tę obsługę na dłuższy czas.

Haddon [8] i Lister [23] wymieniają cztery możliwe rozwiązania:

1. W wypadku wykonania w monitorze **M** operacji **wait** możliwy jest tylko dostęp do tego monitora, bez zwolnienia wykluczenia jakimkolwiek z monitorów, które wywołały ten monitor.

2. W wypadku wykonania w monitorze **M** operacji **wait** możliwy jest dostęp do monitora **M** i do wszystkich monitorów, które obsługiwały ten proces, a podczas tej obsługi wywołały monitor **M**. Po odwieszeniu wykluczenie zostaje przywrócone tylko w monitorze **M**, a później jest sukcesywnie przywracane w monitorach, do których wraca obsługa procesu (po wykonaniu procedury w wywołanym monitorze). Jeżeli podczas obsługi procesu monitor nie wykona operacji **wait**, to w monitorach wywołujących monitor **M** wykluczanie zostanie utrzymane.

3. Wykluczenie jest zwalniane przy każdym wywołaniu monitora przez monitor i przywracane po obsłudze tego wywołania (analogicznie jak w przypadku wykonania operacji **wait** w poprzedniej propozycji).

4. Wywołanie monitora przez monitor jest w ogóle zabronione.

Za pierwszym rozwiązaniem przemawia fakt, że gdy wykluczanie wykonania zostanie utrzymane w mocy w momencie wywołania procedury innego monitora, to nie ma możliwości zmiany struktury danych monitora wywołującego. W rezultacie, niezmiennik spełniony przed wykonaniem operacji wywołania monitora będzie również spełniony po wykonaniu tej procedury, co jest bardzo korzystne z punktu widzenia formalnego dowodzenia twierdzeń. Rozwiązanie to, najpowszechniej stosowane, przyjęto po raz pierwszy w języku CONCURRENT PASCAL [2]. Jego wadą jest podatność na występowanie zakleszczenia. Klasycznym schematem zakleszczenia jest przypadek, gdy dwa procesy zostaną zawieszone, oczekując nawzajem na sygnał odwieszenia siebie, co jest spowodowane błędną algorytmicznie dystrybucją zasobów.

W zależności od sposobu implementacji monitora mogą też występować dwa dodatkowe przypadki zakleszczenia. Pierwszy polega na tym, że monitor wywołujący zostaje wskutek tej akcji ponownie wywołany, zarówno bezpośrednio, jak i pośrednio, w ramach cyklicznego wywołania. Zakleszczenia tego rodzaju można uniknąć stosując hierarchiczną strukturę wywołań monitorów. Drugi przypadek zakleszczenia występuje wtedy, gdy informacja o odwieszeniu pewnego procesu **P** jest przekazywana przez jeden z monitorów, w którym wykluczenie obsługi innego procesu jest utrzymane z powodu zawieszenia procesu **P**. Monitor ten jest jednym z monitorów, które wywołały monitor zawieszający proces **P**.

Możliwość wystąpienia zakleszczeń spowodowała przyjęcie przez Wettsteina [29] i Listera [21, 22] drugiej z omawianych propozycji implementacji monitora. Główny zarzut wysuwany przeciwko tej implementacji wynika stąd, że utrzymanie lub zwolnienie wykluczenia nie zależy od algorytmu w nim zapisanego, lecz od algorytmu wywoływanego monitora. Jest to sprzeczne z aksjomatyczną definicją modułu [27], gdyż: *znaczenia jednostki programowej względem jej układu odniesienia* **NIE** *można wnioskować na podstawie analizy samego tekstu jednostki.*

Dodatkowo należy zwrócić uwagę na trudności wynikające z faktu, że powrót z wywoływanej procedury następuje automatycznie, gdy wykluczanie zostało utrzymane lub przez kolejkę wejściową, gdy wykluczenie zostało zwolnione. Trudność tę Lister proponuje ominąć, stosując mechanizm globalnego wykluczania. Oznacza to, że proces wywołując monitor uzyskuje dostęp do całej hierarchicznej struktury monitorów, a wtedy żaden inny proces nie może być obsługiwany przez żaden z monitorów. W takim wypadku wywołanie innego monitora, jak również powrót z wywoływanego monitora do monitora wywołującego, nie zmieniają stanu procesu. Wadą tego rozwiązania jest wydłużony czas reakcji, ponieważ proces nie może być obsłużony przez wolny monitor, gdy jakiś inny proces jest obsługiwany przez inny monitor. Szczególnie w przypadku systemów wieloprocesorowych mechanizm globalnego wykluczania może stać się wąskim gardłem systemu procesów współbieżnych powodując okresową bezczynność kilku procesów.

Bardzo ciekawe — zdaniem autora — jest trzecie rozwiązanie [16]. Oczywiste stają się w nim trudności związane z weryfikacją poprawności, gdzie przed każdą operacją wywołania monitora należy zagwarantować spełnienie asercji monitora. W takim wypadku z każdym wywołaniem innego monitora należy związać następującą regułę dowodzenia:

I (wywołanie monitora) I

Takie rozwiązanie musi budzić pewne wątpliwości, a nawet prowadzić do stwierdzenia [19], że jest „beznadziejnie uciążliwe". Warto jednak zwrócić uwagę, że chcąc przeprowadzić asercyjny dowód poprawności monitora w wypadku poprzednio omówionej implementacji — dysponując jedynie tekstem tego monitora — uzyska się identyczne rozwiązanie.

Na szczęście metoda asercji nie jest jedyną metodą pozwalającą na formalne wykluczenie błędów uwarunkowanych czasowo. Innym rozwiązaniem jest wykorzystanie pojęcia atomu oznaczającego podoperację monitora, która musi być wykonana niepodzielnie. Jednocześnie podanie reguł programowania pozwala automatycznie wykluczyć na etapie kompilacji wystąpienie błędów uwarunkowanych czasowo [7, 8]. Jeżeli nie brać pod uwagę trudności związanych z asercyjną weryfikacją poprawności programów, to do oczywistych zalet tego rozwiązania należą:

- zlikwidowanie możliwości powstania zakleszczenia spowodowanego implementacją wywołania monitora,
- uniezależnienie czasu podjęcia obsługi procesu przez monitor jedynie od szybkości wykonania procedur tego monitora oraz aktualnej długości kolejki wejściowej.

W przypadku zastosowania kolejki priorytetowej, maksymalny czas podjęcia obsługi procesu o najwyższym priorytecie nie przekroczy czasu najdłuższego, nieprzerwalnie wykonywanego fragmentu monitora.

W implementacji języka CCPASCAL [12] przyjęto rozwiązanie o zwalnianiu wykluczania w momencie wykonywania operacji **wait** tylko w monitorze aktualnie obsługującym proces (rozwiązanie pierwsze). Wprowadzenie w tym języku nowej konstrukcji programowej **pure class** (której zmiennymi globalnymi mogą być tylko monitory i inne obiekty **pure class**) pozwala na efektywne projektowanie systemów współbieżnych bez użycia zagnieżdżonych wywołań monitorów.

Warto odnotować, że czwarta propozycja została również zaimplementowana. W języku SIMONE [13] zagnieżdżone wywołania monitorów nie są w ogóle dozwolone.

MODYFIKACJE FUNKCJONOWANIA MONITORA

Praktyczne wykorzystanie istniejących implementacji monitora, przy pisaniu dużych programów, wykazało wiele ich niedogodności. Z drugiej strony — wymagania formalne, by monitor był modułem, w którym:

- zdefiniowano pewną strukturę danych
- udostępniono zbiór dobrze zdefiniowanych operacji na niej
- zabezpieczono wzajemne wykluczanie wykonania tych operacji w tym samym czasie

pozwalają na jego liczne modyfikacje eliminujące przedstawione wady i umożliwiające zachowanie spójności koncepcji.

W porównaniu z implementacją monitora w języku CONCURRENT PASCAL [2] najliczniejszą grupą zmian są nowe definicje operacji synchronizacyjnych oraz operacji wywołania monitora przez monitor. Większość z nich nie

zmienia jednak klasy problemów rozwiązywalnych przy użyciu zmodyfikowanego monitora i jest raczej próbą dostosowania monitora do indywidualnego gustu użytkownika. Propozycja autora [15], by wykonanie operacji **delay** pozwalało na zawieszenie procesu jednocześnie w kilku kolejkach, zwiększa obszar zastosowań. Efektywne zakodowanie informacji o przyczynie zawieszenia umożliwia sygnałom zdarzeń logicznie związanych z poszczególnymi kolejkami (czyli operacji **continue**) odwieszenie procesu oczekującego na spełnienie alternatywy lub koniunkcji zdarzeń. Brak takiej możliwości jest jednym z głównych zarzutów przedstawionych przez Keedy'ego [14], gdyż nie pozwala, na przykład, na obsługę wejścia-wyjścia.

Kolejny zarzut dotyczy braku możliwości obsługi przez monitor sytuacji wyjątkowych. Nie dyskwalifikuje on jednak samej koncepcji, gdyż nie naruszając jej można łatwo uzupełnić monitor o procedury specjalne, podejmujące obsługę takich sytuacji [16]. Nieco inaczej rozwiązano obsługę sytuacji wyjątkowych w języku MESA [19], gdzie ewentualna obsługa jest deklarowana dla każdej procedury wejściowej osobno.

Innym problemem jest potrzeba bezpośredniego powiązania informacji zgłaszanych przez sprzęt z algorytmem monitora. W języku MESA definiowano w tym celu specjalny typ zmiennej warunkowej, bazującej na fizycznym adresie przerwania. Proces zawieszony na takiej zmiennej jest odwieszany po nadejściu przerwania, analogicznie jak po wykonaniu operacji **notify**. Monitor jest informowany o konieczności obsługi odwieszonego procesu i podejmuje tę obsługę w najbliższym czasie, ustalonym ze względu na długość i charakter wcześniejszych wołań. W monitorach stosujących semantykę instrukcji synchronizacyjnych zgodną z językiem CONCURRENT PASCAL, rozwiązanie tego problemu wymaga odmiennego od **delay** rozkazu zawieszającego w oczekiwaniu na spełnienie zdarzeń zależnych od sprzętu.

Ostatnim powszechnie badanym problemem jest sposób bezpiecznego i efektywnego dynamicznego tworzenia procesów oraz monitorów. Rozwiązanie przyjęte w języku MESA jest również najbardziej elastyczne. Pozwala ono nie tylko na realizowanie monitorów tego samego typu, jako procedury czystej (tzn. kod tych monitorów jest wspólny dla wszystkich monitorów danego typu, a generowana jest tylko struktura danych). Dopuszczalne jest także, aby globalna struktura danych była w całości przekazywana, jako fragment rekordu, będącego parametrem wywołania procedury (łącznie ze wszystkimi informacjami dotyczącymi wzajemnego wykluczania czy stanu kolejek). W tym ostatnim przypadku należy zwrócić uwagę, że reguły zasięgu wprowadzone w języku MESA nie zapewniają pełnego bezpieczeństwa systemu. Przykładowo dopuszczalne jest, aby wskutek działania operacji przypisania — mającej jako argument rekord tego samego typu — zmianie uległa zawartość rekordu, będącego parametrem, co w rezultacie — z powodu zniszczenia stanu kolejek — może doprowadzić do kompletnego chaosu w systemie.

PORÓWNANIE MONITORA Z INNYMI NARZĘDZIAMI SYNCHRONIZACJI

Użyteczność monitora została potwierdzona licznymi przykładami implementacji: RC4000 [1], SOLO [3], TRIO [5], PILOT [24]. Jedynym mechanizmem, który również stosowano do tego celu, jest semafor. Klasycznym przykładem systemu operacyjnego, zrealizowanego przy jego użyciu, jest system THE [7].

Używając semaforów można efektywnie rozwiązać złożone problemy współbieżności — ich implementacja wymaga jednak od programisty dużej dyscypliny w programowaniu i nie istnieje żadna możliwość wspomagania pracy przez kompilator. Z tego względu nasuwa się pewna analogia z programowaniem sekwencyjnym. Choć wiadomo, jak zwarte i efektywne są programy pisane w językach asemblera, w praktyce nieomal zawsze dąży się do użycia języków wysokiego poziomu. Jednak z faktu, że pewne błędy w programowaniu współbieżnym można wykryć dopiero na etapie testowania — wynikają groźniejsze konsekwencje niż w przypadku programowania sekwencyjnego. W programie sekwencyjnym raz zasygnalizowany błąd można stosunkowo łatwo zlokalizować, wykorzystując bogaty zestaw programów śledzących. W programie współbieżnym dołączenie programów śledzących zmienia realizacje czasowe wykonań poszczególnych procesów i błąd, który poprzednio się pojawił, może nie wystąpić ponownie. Reasumując, monitor jest jedynym godnym polecenia narzędziem programowym, pracującym w oparciu o wspólną strukturę danych.

Jak udowodnili Lauer i Needham [20], schemat synchronizacji w monitorze — przy zachowaniu pewnych restrykcji dotyczących stylu programowania — jest równoważny w sensie klasy zastosowań schematowi synchronizacji wymiany komunikatów. Należałoby więc przeanalizować, czy mechanizmy językowe bazujące na wymianie komunikatów, nie są efektywniejszym narzędziem synchronizacji. W chwili obecnej znany jest jeden taki mechanizm — procesy komunikacyjne Hoare'a [9]. Jak wykazały jego pierwsze implementacje [26], ze względu na złożoność, dla komputerów opartych na pamięci monolitycznej, monitor jest narzędziem efektywniejszym. W przypadku systemów wieloprocesorowych z pamięcią rozproszoną (ang. distributed systems) wzrasta natomiast złożoność implementacji monitora i w rezultacie przewagę uzyskują procesy komunikacyjne. Podobnie wygląda porównanie monitora z mechanizmami synchronizacji stosowanymi w języku Ada [28]. Szereg krytycznych uwag wygłoszonych na temat efektywności mechanizmu randezvous. [25] wydaje się wskazywać na potrzebę znacznej modyfikacji tego rozwiązania.

Stosunkowo krótki okres, jaki minął od wprowadzenia pierwszych implementacji konstrukcji językowych bazujących na koncepcji wymiany komunikatów, nie pozwala w pełni zweryfikować ich przydatności. Z drugiej strony, interesujące wydają się próby lepszego dostosowania koncepcji monitora do architektury z pamięcią rozproszoną, czego przykładem jest koncepcja procesów rozproszonych Brinch Hansena [4]. Jest to istotne, przede wszystkim z uwagi na możliwość znalezienia koncepcji uniwersalnej dla obu architektur. Jak na razie monitor wydaje się być ogólniejszy, tylko w klasie komputerów z pamięcią wspólną.

LITERATURA

[1] Brinch Hansen P.: Operating Systems Principles. Prentice Hall, Englewood Cliffs (NJ), 1973

[2] Brinch Hansen P.: The programming language Concurrent Pascal IEEE Trans. on Software Engineering. Vol. SE-1, No. 2, pp. 199—207, 1975

[3] Brinch Hansen P.: The SOLO Operating System. Software Practice and Experience, Vol. 6, No. 2, pp. 141—205, 1976

[4] Brinch Hansen P.: Distributed processes — a concurrent programming concept. Comm. of the ACM, Vol. 21, No. 11, pp. 934—941, 1978

[5] Brinch Hansen P.: The TRIO operating system. Software Practice and Experience, Vol. 10, pp. 943—948, 1980

[6] Dijkstra E. W.: Cooperating sequential processes. Programming Languages. Genuys F. (ed.) Acaademic Press, New York, 1968

[7] Dijkstra E. W.: The structure of „THE" multiprogramming system. Comm. of the ACM, Vol. 11, No. 5, pp. 341—346, May 1968

[8] Haddon B. K.: Nested Monitor Calls. Operating System Review, Vol. 11, No. 4, pp. 18—23, October 1977

[9] Hoare C. A. R.: Monitors — an operating system structuring concept. Comm. of. ACM, Vol. 17, No. 10, pp. 549—557, 1974

[10] Hoare C. A. R.: Communicating sequential processes. Comm. of the ACM, Vol. 21, No. 8, pp. 666—677, 1978

[11] Iszkowski W., Maniecki M.: Programowanie współbieżne. WNT, Warszawa, 1982

[12] Joseph M., Prasad V. R.: More on Nested Monitor Calls. Operating Systems Review, Vol. 12, No. 2, p. 21—25, April 1978

[13] Kaubisch W. H., Perrot R. H., Hoare C. A. R.: Quwasiparallel programming. Software Practice and Experience, Vol. 6, pp. 341—364, 1976

[14] Keedy J. L.: On structuring Operating Systems For Monitors. The Australian Computer Journal, Vol. 10, No. 1, pp. 23—27, 1978

[15] Kotulski K.: On the Use of monitors for Operating Systems. Zeszyty Naukowe Uniwersytetu Jagiellońskiego, Seria Informatyka, nr 1, 1983

[16] Kotulski L.: Metoda eliminacji błędów uwarunkowanych czasowo w systemach procesów współbieżnych. Rozprawa doktorska, Uniwersytet Jagielloński, Kraków, 1984

[17] Kotulski L.: Time depending errors exclusion method in monitors structures. Zeszyty Naukowe Uniwersytetu Jagiellońskiego, Seria Informatyka, nr 3, 1985

[18] Kotulski L.: Monitor jako narzędzie strukturalizacji programów współbieżnych. Informatyka, nr 2—3, 1986

[19] Lampson B. W., Redell D. D.: Experience with Processes and Monitors in Mesa. Comm. of the ACM, Vol. 23, No. 2, pp. 105—117, 1979

[20] Lauer H. C., Needham R. M.: On the duality of Operating Systems Structures. Proc. Second International Symposium on Operating Systems, IRIA Rocquencout, France, October 1978
[21] Lister A. M., Maynard P. J.: An Implementation of Monitors. Software Practice and Experience, Vol. 6, pp. 377—385, 1976
[22] Lister A. M., Sayers P. J.: Hierarchical Monitors. Software Practice and Experience, Vol. 7, pp. 613—623, 1977
[23] Lister A. M.: The Problem of Nested Monitor Calls. Operating Systems Review, Vol. 11, No. 3, pp. 5—7, July 1977
[24] Redell D. et al.: Pilot — An operating system for personal computer. Comm. of the ACM, Vol. 23, No. 2, February 1980
[25] Roberts E. S., Ewans jr A., Morgan C. R., Clarke E. M.: Task management in ADA — A critical evaluation for real-time multiprocessor. Software Practice and Experience, Vol. 11, pp. 1019—1051, 1981
[26] Roper T. J., Batter C. J.: A communicating sequential language and implementation. Software Practice and Experience. Vol. 11, pp. 1215—1234, 1981
[27] Turski W. M.: Metodologia programowania. WNT, Warszawa, 1978
[28] U. S. Department of Defence: Programming Language ADA — Reference Manual. Vol. 106, Lecture Notes in Computer Science, Springer-Verlag, New York, 1981
[29] Wettstein H.: The Problem of nested monitor calls revsed. Operating Systems Review, Vol. 12. No. 1, pp. 19—23, January 1978.

Sterownik sieci Ethernet w systemie Multibus

Sieć Ethernet powstała w końcu lat siedemdziesiątych w głównej mierze jako rezultat prac Roberta Metcalfe'a i Davida Boggsa z firmy Xerox. Wkrótce potem firmy Intel, Digital Equipment i Xerox formalnie uzgodniły współpracę w zakresie tworzenia koncepcji Etherneta i wspólnie opracowały jego specyfikację, opublikowaną we wrześniu 1980 r. Specyfikacja Etherneta opisuje charakterystyki parametrów fizycznych i elektrycznych sieci od strony sprzętowej, a także strukturę funkcjonalną warstwowej architektury sieci. Tylko dwie pierwsze warstwy: warstwa fizyczna i warstwa sterowania łączem, zostały opisane dostatecznie szczegółowo. Specyfikacja pozostałych, wyższych warstw nie wykracza poza charakterystykę funkcjonalną.

Fizycznie, Ethernet dokonuje transmisji szeregowej komunikatów po 50-omowym przewodzie koncentrycznym. Segment przewodu może mieć do 500 m długości i łączyć do 100 stacji. Każda stacja jest dołączona do głównego przewodu przez kabel z nadajnikiem/odbiornikiem (ang. transceiver). Komunikaty są formatowane w standardowe ramki (ang. frame) złożone z oktetów, każdy o długości 8 bitów. Tworzenie ramki polega na zestawieniu adresu przeznaczenia (6 oktetów), źródła (6 oktetów), typu komunikatu (2 oktety), danych (od 46 do 1500 oktetów) i sekwencji kontrolnej (4 oktety). Komunikaty mogą być adresowane do pojedynczej stacji, do wszystkich stacji (ang. broadcast, rozgłaszanie) lub do pewnej liczby wybranych stacji (ang. multicast, rozgłaszanie częściowe). Sygnały są transmitowane przy użyciu kodowania bifazowego (ang. Manchester encoding), najbardziej odpowiedniego do transmisji przez kanał szeregowy.

Komunikaty danych są przesyłane z szybkością 10 MB/s. W określonej chwili dane może przesyłać tylko jedna stacja. Dlatego w sieci Ethernet zdefiniowano metodę wykrywania kolizji, chroniącą magistralę przed równoczesnym dostępem z wielu stacji. Metoda ta zwana dostępem wspólnym z wykrywaniem kolizji (ang. carrier-sense multiple access with collision detection, CSMA/CD) zapewnia wykrywanie dostępu z wielu stacji i ignorowanie jednoczesnych transmisji.

Z powodu zastosowania metody CSMA/CD, dostęp do magistrali ściśle odpowiada regułom FCFS (ang. first-come first-served). Zaleta tego podejścia polega na tym, że każda stacja ma indywidualny, zdecentralizowany dostęp do przewodu przez swój sterownik, bez potrzeby używania osobnego systemu sterowania magistralą. W konsekwencji system jest odporny na awarie spowodowane utratą sterowania siecią, a odpowiednie układy elektroniczne są równie łatwe do realizacji.

Sieć Ethernet rozwinęła się bardzo szybko od koncepcji do popularnego systemu komputerowych sieci lokalnych LAN (ang. Local Area Networks). Wiele firm z USA, a także z Europy rozpoczęło prace nad systemami bazującymi na Ethernecie, a zainteresowanie nim ciągle wzrasta. Jest to jedna z przyczyn, dla której Intel opracował sterownik tej sieci. Inną przyczyną jest to, że sieć Ethernet można potencjalnie zastosować w wielu różnych systemach komputerowych. Wydaje się, że doprowadzenie Etherneta do kompatybilności z magistralą Multibus stwarza bazę do tworzenia sieci lokalnych.

BUDOWA STEROWNIKA

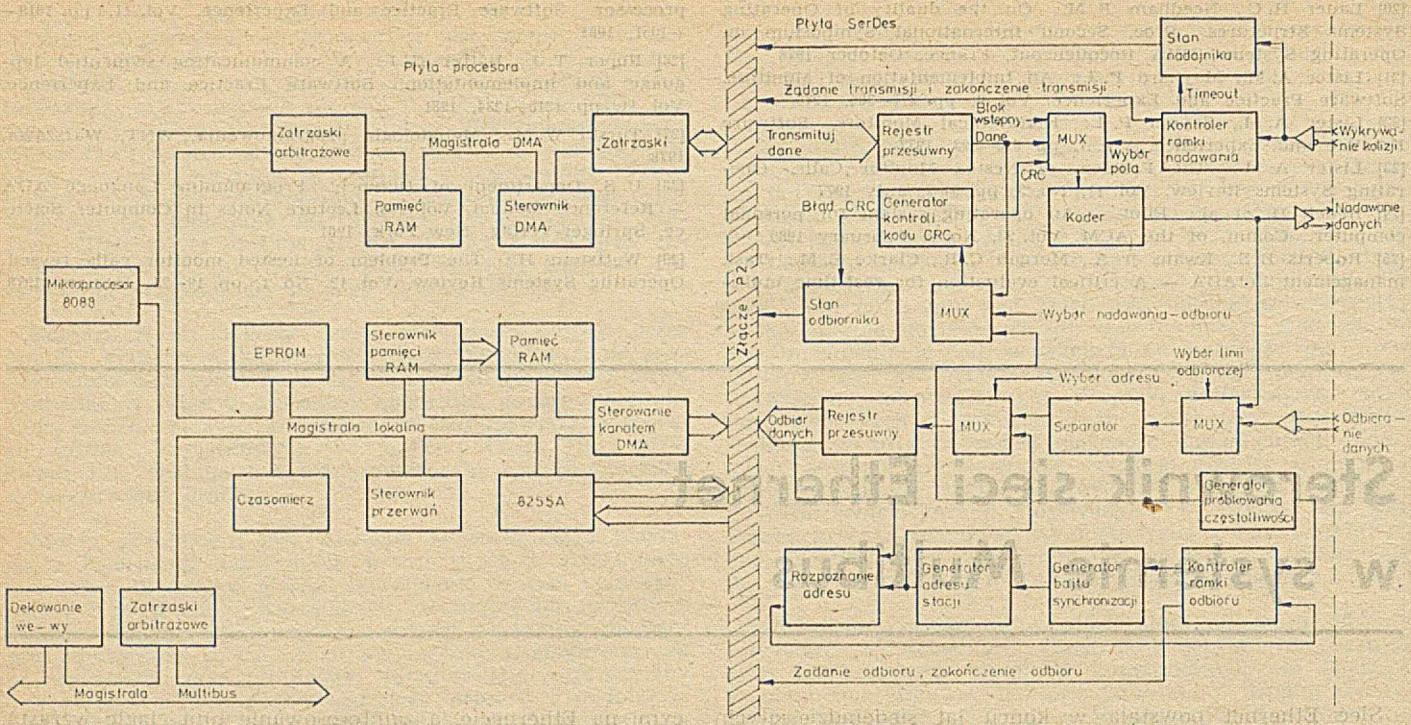
Sterownik iSBC 550 składa się z dwóch płyt — płyty procesora opartej na mikroprocesorze 8088 oraz płyty dokonującej przekształcenia danych do (i z) postaci szeregowej (ang. serialization/deserialization board), nazywanej płytą SerDes (rys. 1). Płyta procesora zawiera układ przetwarzający, buforujący i przenoszący wiadomości do pamięci systemu. Płyta SerDes odpowiada specyfikacji firm DEC, Intel i Xerox w zakresie sterowania łączem (ang. data link control) oraz na poziomie fizycznym (ang. physical link control). Dokonuje ona konwersji 8-bitowych danych równoległych na dane szeregowe, tworzy ramki (ang. frame) dla komunikatów, oblicza kod CRC (ang. cyclic redundancy check) i sprawdza, czy ta wartość jest zgodna z otrzymaną w komunikacie. Dokonuje także kodowania i dekodowania bifazowego komunikatów, rozpoznaje adres przeznaczenia i realizuje wykrywanie kolizji. Obie płyty mają typowe wymiary i złącza krawędziowe rozmieszczone zgodnie ze standardowym sprzęgiem Multibus.

Płyta procesora zawiera wersję 5 MHz mikroprocesora 8088, 16 KB dynamicznej pamięci RAM przeznaczonej do wykonywania programów sieci Ethernet, 8 KB pamięci PROM dla oprogramowania układowego (ang. firmware), na której opiera się sterowanie łączem i międzyprocesorowy protokół magistrali Multibus, tzw. MIP (ang. Multibus Interprocessor Protocol). Dodatkowe 8 KB szybkiej pamięci statycznej RAM używa się do buforowania nadawanych i odbieranych danych. Płyta procesora steruje pracą płyty SerDes, odbiera od niej sygnały stanu i dane. Wszystkie transmisje danych przez płytę SerDes na poziomie sterowania łączem sieci Ethernet są buforowane w 8 KB pamięci statycznej RAM.

Płyta SerDes jest oparta na układach o małym i średnim stopniu scalania. Jest sprzęgnięta z magistralą Multibus przez płytę mikroprocesora i przyjmuje dane nadawane z procesora oraz wysyła do niego dane otrzymane z sieci (rys. 2). Do komunikacji z płytą procesora służą trzy sygnały uzgodnienia (ang. handshaking signals): „żądanie transmisji spełnione", „żądanie odbioru spełnione", „kolizja wykryta".

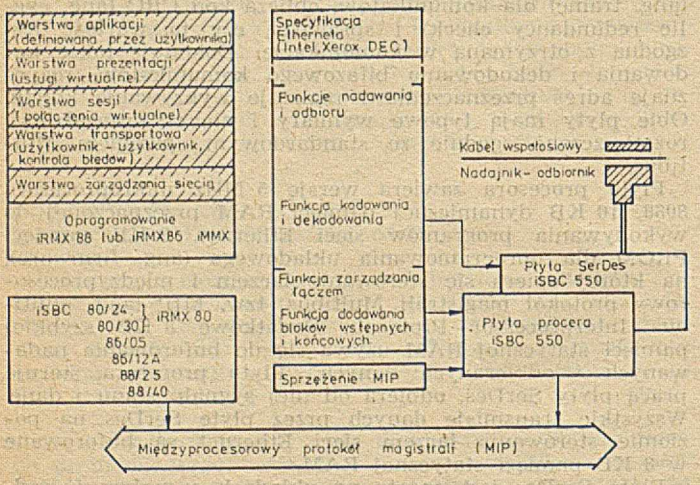
OPROGRAMOWANIE I REALIZACJA SPRZĘŻENIA

Sterownik sieci Ethernet iSBC 550 zawiera oprogramowanie układowe, wspomagające sprzężenie przez protokół MIP. MIP definiuje pseudoarchitekturę, w której procesy



Rys. 1. Schemat blokowy sterownika iSBC 550. Płyta procesora oparta na mikroprocesorze 8088 buforuje komunikaty i przenosi je do pamięci systemu. Płyta SerDes przekształca 8-bitowe dane równoległe na postać szeregową, tworzy ramki i kontroluje wystąpienie błędów (pierwsze słowo w ostatnim wierszu, z prawej strony rysunku, to: Żądanie)

wykonywane na różnych jednopłytowych komputerach mogą komunikować się wzajemnie w sposób niezawodny i kontrolowany. Dotyczy to systemów złożonych z heterogenicznych zbiorów procesorów, wykonujących heterogeniczny zestaw programów nadzorczych czasu rzeczywistego, bądź programów aplikacyjnych. Tak więc MIP umożliwia efektywną współpracę wielu procesorom rezydującym na wspólnej magistrali Multibus. Dzięki komunikacji po magistrali Multibus, zapewnianej przez ujednolicony zestaw sprzężeń systemów operacyjnych, możliwa jest realizacja wielu, stosunkowo słabo powiązanych ze sobą, procesów. Wymianą komunikatów steruje moduł programowy iMMX 800, który zapewnia adresowanie, przesyłanie danych i zarządzanie pamięcią.



Rys. 2. Schemat konfiguracji systemu. Płyta iSBC 550 może współpracować z różnymi procesorami. Dzięki zastosowaniu modułu iMMX 800 możliwe jest programowanie wyższych warstw komunikacji

Sterownik iSBC 550 i pakiet programowy iMMX 800 mogą być używane w połączeniu z dowolną z płyt mikrokomputerów iSBC 80/24, 80/30, 86/05, 86/12A, 88/25 lub 88/45 i tworzyć konfiguracje przeznaczone do typowego przetwarzania, a także do specjalnych obliczeń numerycznych. Zestaw złożony z centralnego procesora, pamięci, płyt dodatkowych i sterownika iSBC 550 w jednej kasecie tworzy kompletny system.

System operacyjny iRMX, który obejmuje moduł oprogramowania iMMX 800, spełnia większość wymagań programowych. To, co pozostaje do zrobienia użytkownikowi, polega na opracowaniu wyższych warstw oprogramowania komunikacyjnego, zgodnie ze specyfikacją Etherneta.

Oprogramowanie układowe sterownika składa się z poleceń i danych, które biorą udział w sterowaniu przepływem komunikatów, rozpoznawaniu typu komunikatu i rozgłaszaniu komunikatów. Do przykładowych poleceń sterujących przepływem komunikatów należą: RECEIVE A MESSAGE i TRANSMIT A MESSAGE. Inne polecenia dotyczą statystyk sieci, typów akceptowanych komunikatów i adresowania przy rozgłaszaniu. Oprogramowanie układowe rozpoznaje osiem typów komunikatów i maksymalnie osiem adresów przy rozgłaszaniu.

Specyfikacja Etherneta określa maksymalną długość kabla koncentrycznego i kabla nadajnika-odbiornika. Nadajnik-odbiornik musi być bardzo blisko (rzędu centymetrów) od kabla koncentrycznego, aby spełniać wymagania przepustowości. Płyta SerDes sterownika iSBC 550 łączy się fizycznie z kablem o długości 56 cm, złożonym z czterech skręconych par przewodów. Kabel ten jest identyczny z kablem nadajnika-odbiornika i jest do niego dołączony przez odpowiednie złącze. Pierwsza para przewodów służy do przenoszenia danych do nadajnika-odbiornika, druga — przenosi dane z nadajnika-odbiornika, trzecia — umożliwia wykrywanie kolizji, a ostatnia służy do zasilania nadajnika-odbiornika.

Wysyłane komunikaty przechodzą od płyty procesora do płyty SerDes. Po utworzeniu ramki i obliczeniu oraz sprawdzeniu kodu CRC, dane w postaci szeregowej są wysyłane do nadajnika-odbiornika. W tym momencie dane są już zakodowane bifazowo, a dalej nadajnik-odbiornik dokonuje ich konwersji na sygnały elektryczne, których charakterystyki (czas narastania i opadania, amplituda itd.) odpowiadają specyfikacji Etherneta.

Dane skierowane do określonej stacji w sieci Ethernet są odbierane przez nadajnik-odbiornik, wysyłane z powrotem do płyty SerDes do odtworzenia z postaci szeregowej i przenoszone do płyty procesora iSBC 550. Następnie płyta SerDes na podstawie adresu przeznaczenia i informacji o typie komunikatu (wysłanej razem z komunikatem), kieruje dany komunikat do odpowiedniego procesu.

Umieszczenie sterownika Etherneta na płycie magistrali Multibus jest pierwszym krokiem do zintegrowania systemów modularnych z lokalnymi sieciami komputerowymi.

Oprac. ROMAN GRABOWICZ

na podst. Computer Design Vol. 20, No. 9, September 1981

COMPUTER STUDIO KAJKOWSCY

ul. Balladyny 3B, 81-524 Gdynia
Tel.: 29-00-18, 24-01-50

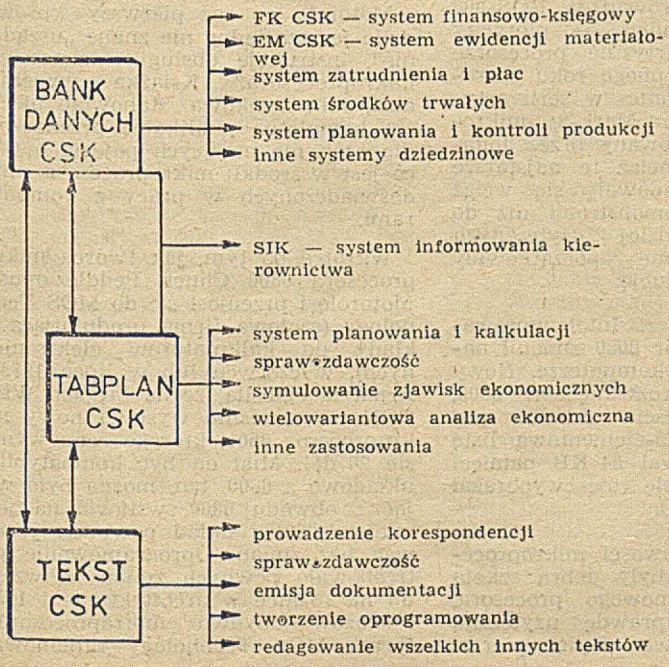
Oprogramowanie z CSK dla potrzeb zarządzania

(dla mikrokomputerów 8- i 16-bitowych)

Oferta programowa CSK obejmuje programy, które służą do automatyzacji przetwarzania danych we wszystkich dziedzinach funkcjonowania przedsiębiorstw:

- do prowadzenia korespondencji, ewidencji i rozliczeń finansowych,
- do wspomagania decyzji na szczeblu dyrektora.

Miejsce oprogramowania CSK w zautomatyzowanym systemie przetwarzania danych w przedsiębiorstwie przedstawia rysunek.



PROFESJONALNE OPROGRAMOWANIE MIKROKOMPUTERÓW

CSK oferuje następujące oprogramowanie użytkowe i systemowe dla mikrokomputerów 8- i 16-bitowych

Nazwa programu	8-bitowe	16-bitowe
Oprogramowanie użytkowe		
BANK-DANYCH CSK — system zarządzania bazą danych	+	+
TABPLAN CSK — komputerowy arkusz kalkulacyjny	+	+
TEKST CSK — pakiet redagowania tekstów	+	+
TRANSCOM CSK — program komunikacji z ODRĄ	+	+
TRANSCOM/M CSK — program komunikacji między mikrokomputerami	+	+
BANK-CSK — graficzny system komunikacji z bazą danych	—	+
BGRAF CSK — pakiet grafiki prezentacyjnej	—	+
FK CSK — system finansowo-księgowy	+	+
EM CSK — system ewidencji materiałowej	+	+
PL TEKST CSK — pakiet redagowania tekstów (polskie znaki)	—	+
Oprogramowanie systemowe		
SOMIK — rozbudowa systemu operacyjnego CP/M 2.0	+	—
W/SYS CSK — system operacyjny wielozadaniowy/wielostanowiskowy	+	+
GSK CSK — pakiet procedur graficznych wg normy GKS	—	+

+ tak; — nie

Oprogramowanie CSK może być eksploatowane **na mikrokomputerach 8-bitowych:**

- ELWRO seria 500 i 600
- ROBOTRON 5110/20/30, 1715
- MK 4101/02
- ComPAN
- IMP-85
- innych (z systemem CP/M)

mikrokomputerach 16-bitowych:

— LIDIA II/XT
— MAZOWIA
— ELWRO 800
— M24 (Olivetti)
— innych (zgodnych z IBM PC XT i AT).

Cena obejmuje: dyskietki z programem (wraz z kopią), jeden egzemplarz dokumentacji użytkowej. Termin realizacji — 7 dni od daty otrzymania zamówienia.
CSK organizuje kursy użytkowania oprogramowania.

EO/81/K/86

Historia i rozwój mikroprocesorów

Protoplastą dzisiejszych mikroprocesorów był powstały w 1959 r. w firmie Texas Instrument pierwszy układ scalony SSI (ang. Small Scale of Integration, mały stopień scalenia) zawierający względnie mało tranzystorów. Rozwój postępował dynamicznie i ... w 1969 r. istniejąca zaledwie rok firma Intel wprowadziła na rynek mikroukład o kilobitowej pamięci RAM. Chociaż nie można było jeszcze zbudować mikrokomputera, nowy układ znalazł dużo zastosowań, znacznie więcej niż układy produkowane poprzednio.

Mniej więcej w tym samym czasie, latem 1969 r., japońska wytwórnia kalkulatorów Busicom zleciła Intelowi wyprodukowanie serii mikroukładów do nowych kalkulatorów. Kalkulatory zaprojektowane przez japońskich inżynierów miały zawierać kilka mikroukładów złożonych z 3000—5000 tranzystorów każdy.

Marcian Hoff, wyznaczony do współpracy z Japończykami, ocenił projekt Busicomu jako zbyt skomplikowany i nieefektywny ekonomicznie. Mając doświadczenie w pracy nad PDP-8, minikomputerem, który miał bardzo małą listę rozkazów, wydedukował, że używając małego, uniwersalnego procesora można uprościć budowę kalkulatora. Taka konstrukcja, wykorzystująca raczej oprogramowanie niż elektronikę, oczywiście bardzo zwiększała wymagania stawiane pamięci, ale z tym problemem firma Intel mogła sobie poradzić. Hoff zdawał sobie sprawę, że taki procesor może znaleźć bardzo dużo zastosowań i cały pomysł sprzedał szefom swojej firmy. Postanowiono pracować jednocześnie nad dwoma projektami. Japończycy uprościli swój projekt i efektem końcowym miał być kalkulator o 12 układach, z których każdy mieścił ponad 2000 tranzystorów. Zespół dowodzony przez Hoffa przedstawił procesor uniwersalny, mieszczący 1900 tranzystorów, który został wytypowany do dalszej produkcji zamiast japońskiego. Busicom zlecił Intelowi produkcję układu znanego później jako 4004.

W 1970 r. do Intela przeszedł Federico Faggin, późniejszy założyciel firmy Zilog. W dziewięć miesięcy później powstała krzemowa wersja mikroprocesora 4004, który początkowo sprzedawano tylko Busicomowi, ale latem 1971 r. Intel uzyskał prawo sprzedaży innym klientom. W listopadzie 1971 r. Intel reklamował 4004 jako czterobitowy procesor wykonujący 60 000 operacji na sekundę.

NARODZINY PROCESORÓW OŚMIOBITOWYCH

W tym samym czasie, gdy trwały dalsze prace nad udoskonaleniem mikroprocesora 4004, firma CTC (Computer Technology Corporation, obecnie Datapoint) zleciła Intelowi oraz innemu znanemu producentowi — Texas Instruments — zaprojektowanie układów dużego stopnia scalenia do budowy nowych terminali inteligentnych. Obydwie firmy zaproponowały uniwersalne, ośmiobitowe procesory.

Ciekawe spostrzeżenie dotyczy długości słowa mikroprocesorów. Układ czterobitowy zastosowano do kalkulatorów, bo pracują w kodzie BCD, a ośmiobitowy do terminala operującego znakami ASCII. Interesujące jest, że firma CTC nie wybrała żadnej z propozycji, budując terminal na standardowych układach scalonych. Oczywiście tak Intel, jak i Texas Instruments dopracowały swoje projekty — TI uzyskał prawa patentowe na swój procesor, a projekt Intela przekształcił się w mikroprocesor 8008.

Mikroprocesor 8008 wprowadzony w kwietniu 1972 r. był pierwszym ośmiobitowym mikroprocesorem na rynku. Wymagał wprawdzie co najmniej 20 dodatkowych układów i miał 45-elementową listę rozkazów, wykonywał jednak 300 000 rozkazów na sekundę, adresując całe 16 KB pamięci.

Przed pojawieniem się mikroprocesorów 8008 i 4004 w elektronice królowała tzw. logika przypadku (ang. random logic). Tą nazwą określono złośliwie duże ilości połączonych ze sobą układów SSI i MST. Komputery budowane z takich układów nadawały się bardziej do demonstrowania zasad działania niż do wykonywania jakichkolwiek zadań.

W 1973 r. firma Scelbi Computer Consulting Inc. wprowadziła RGS-008 firmy RGS Electronics — pierwszy mikrokomputer oparty na procesorze 8008. W lipcu następnego roku czasopismo Radio Electronics w serii artykułów przedstawiło Mark-8, mikrokomputer zaprojektowany przez Jonathana Titusa. I chociaż te najstarsze mikrokomputery nadawały się wciąż raczej tylko do demonstracji niż do wykonywania bardziej użytecznych zadań, to one właśnie zapoczątkowały komputerową rewolucję.

W kwietniu 1974 r. Intel wprowadzając mikroprocesor 8080 zmienił nasze wyobrażenia o komputerze. Nowy procesor, będący rozwinięciem 8008, wymagał tylko sześciu układów pomocniczych, miał 75-elementową listę rozkazów i adresował 64 KB pamięci (wówczas nikt sobie nie wyobrażał dłuższego programu).

Ograniczone możliwości mikroprocesorów 4004 i 8008 były dobrą szkołą dla projektantów nowego procesora, który stał się naprawdę użyteczną maszyną obliczeniową. Będąc pierwszym mikroprocesorem zaprojektowanym nie tylko w celu zastąpienia elementów logicznych, 8080 był bardzo wygodny w użyciu z punktu widzenia sprzętowego.

Początek 1975 r. przyniósł serię artykułów w czasopismie Popular Electronics na temat tzw. minikomputera Altair 8800, opartego na procesorze 8080. Altair zaprojektowano w firmie MITS i sprzedawano w częściach do złożenia. Za 395 dol. klient dostawał mikroprocesor 8080, płytę centralną, obudowę i zasilacz oraz 256-bajtową pamięć. Sam procesor kosztował wtedy 360 dol., ale Intel zawarł z MITS specjalną umowę, której warunki pozwalały sprzedawać Altaira z godziwym zyskiem.

Oczywiście Altair odegrał wielką rolę w sukcesie samego mikroprocesora 8080. Programiści mieli teraz okazję i silniejszą motywację do pisania oprogramowania. Ponadto, otwarta architektura (lepsza wersja tego, co później stało się znane jako magistrala S-100/IEEE 696) pozwoliła na dołączanie dowolnych urządzeń zewnętrznych do komputera. Jednym z nich była stacja dysków firmy Digital Microsystems z nowym systemem operacyjnym dla mikroprocesora 8080, nazwanym CP/M (ang. Control Program for Microcomputer). W rezultacie większość opracowanego do tej pory oprogramowania mikrokomputerów powstała opierając się na tej samej albo rozszerzonej liście rozkazów co 8080.

RODZINA MIKROPROCESORÓW FIRMY MOTOROLA

W odpowiedzi na sukces Intela firma Motorola opracowała mikroprocesor 6800. Motorola jako pierwsza wprowadziła dodatkowe, specjalne układy przeznaczone do współpracy z mikroprocesorem. Zapewniły one równoległe i szeregowe funkcje we-wy, znacznie ułatwiające pracę projektantom systemów.

Motorola po raz pierwszy wprowadziła jeszcze jedno nie znane „urządzenie", instrukcję obsługi i zastosowań mikroprocesora. Książka wypełniła dużą lukę rynkową, stanowiąc pierwszą kompletną publikację przenaczoną dla osób nie mających pojęcia o tym, co jest w środku mikroprocesora i niedoświadczonych w pracy z komputerami.

Wkrótce po tym, jak twórca mikroprocesora 6800 Chuck Peddle opuścił Motorolę i przeniósł się do MOS Technology (czołowa firma produkująca układy do kalkulatorów elektronicznych), w czerwcu lub w lipcu 1975 r. firma ta ogłosiła, że na targach WESCON we wrześniu wystawi nowy mikroprocesor 6501, którego cena wyniesie 20 dol. Miał on być kompatybilny układowo z 6800, tzn. można było wyjąć z obwodu 6800, wstawić na jego miejsce 6501 i układ powinien pracować bez zmian. Oprogramowanie potrzebowało pewnych zmian ze względu na różnice w architekturze i liście rozkazów obydwu mikroprocesorów. Firma MOS Technology zaplanowała

również wersję 6502 z zegarem, kosztującą 25 dol.

Spowodowało to duże poruszenie w branży, zważywszy fakt, że mikroprocesory 8080 i 6800 sprzedawano po 179 dol. W Dolinie Krzemowej panował zgody pogląd, że był to prosty chwyt reklamowy i nie ma powodu, aby układy mikroprocesorowe miały przejść taką samą ewolucję cenową jak kalkulatorowe, od setek do kilku dolarów. Ludzie z branży twierdzili, że cena pojedynczego mikroprocesora nie spadnie poniżej 100 dol. Mimo to z niecierpliwością czekano na targi.

W stoisku MOS Technology nie było zapowiadanych układów. Okazało się, że wystawcy nie mogli niczego sprzedawać w swoich stoiskach, więc wszyscy zainteresowani (m. in. Steve Woźniak, Chuck Peddle) przenieśli się do pobliskiego hotelu. Oprócz długo oczekiwanych mikroprocesorów firma MOS Technology sprzadawała podręczniki oprogramowania i sprzętu oraz pierwszy zewnętrzny układ wielofunkcyjny (RAM, ROM, zegar i we-wy w jednym układzie). Jedna z wersji tego układu nazywała się TIM (ang. Terminal Interface Monitor) i zawierała kompletny monitor do współpracy z klawiaturą, a druga KIM (ang. Keyboard-Input Monitor), razem z centralnym procesorem, pamięciami ROM i RAM, klawiaturą i równoległymi układami we-wy stanowiła kompletny system, sprzedawany za 245 dol. Nikt poprzednio nie sprzedawał niczego podobnego — rewolucja nabierała rozmachu.

Wkrótce potem Motorola oskarżyła MOS Technology i Chucka Paddle o kradzież technologii i produkcja 6501 została wstrzymana. Komputery zbudowane na bazie 6502, to już teraz legenda: KIM-1, Apple I i II, cała seria Atari, PET i VIC-20 firmy Commodore.

NARODZINY Z80 I REAKCJA INTELA

Na przełomie lat 1975—1976 Federico Faggin opuścił Intela i założył firmę Zilog. Postawił przed sobą i swoim młodym, ale już znanym współpracownikiem, Masatoshi Shimą, ambitne zadanie, zbudowanie superukładu 8080. Miał to być mikroprocesor Z80, rozbudowana wersja 8080, pracujący na większości programów napisanych dla swego poprzednika, lecz z częstotliwością 4 MHz, czyli dwa razy szybciej. Przewidziano dla niego 176-elementową listę rozkazów, co brzmiało jak bajka i spowodowało, że początkowo odnoszono się do Z80 z takim samym sceptycyzmem jak do 6502.

Los i tym razem był łaskawy dla konstruktorów-marzycieli. Z80 został kolejnym przebojem rynkowym i jego popularność nie zmalała do dziś.

Wkrótce potem pojawiły się karty Z80 do magistral S-100. Chociaż Z80 miał dużo większe możliwości niż 8080 (wystarczy porównać listy rozkazów), niewielu użytkowników pisało oprogramowanie wykorzystując pełne możliwości nowego mikroprocesora. Przyczyna była prosta — większość zainstalowanych wówczas mikrokomputerów wykorzystywała 8080 i jeśli ktoś opracowałby program tylko dla Z80, automatycznie ograniczałby sobie rynek. Jest to zresztą problem aktualny do dziś.

Oczywiście nowe mikrokomputery nie wykorzystywały już procesora 8080. Z80 był dużo łatwiejszy w użyciu, nie wymagał żadnych dodatkowych układów wspomagających i miał jedno napięcie zasilania. Nawet gdy nie używa się jego pełnej listy rozkazów, jest dużo szybszy.

Wraz z Z80 wprowadzono również nową ideę — wbudowanie układu odświeżania pamięci dynamicznej RAM. Pamięci dynamiczne zawsze kosztowały cztery razy mniej niż ich odpowiedniki statyczne, co przesądzało o ich atrakcyjności. Z powodu wymogu ciągłego odświeżania, pamięci dynamiczne są jednak trudne w użyciu i projektanci systemów zawsze patrzyli na nie niechętnie. Pojawienie się Z80 rozwiązało 90% ich problemów i pozwoliło projektować systemy dużo tańsze niż poprzednio. Dobrym przykładem jest mikrokomputer TRS-80, zaprojektowany przez Steve'a Leiningera.

W tym czasie Intel rozwijał swoje wyroby, wprowadzając mikroprocesor 8085. Nowy mikroprocesor miał kilka dodatkowych rozkazów, ale ogólnie udoskonalenia wprowadzone przez Intela były podobne do rozwiązań Zilog. Z punktu widzenia sprzętowego 8085 był wygodniejszy niż konkurencyjny Z80, lecz miał jedną wadę — częstotliwość zegarową 3 MHz w porównaniu do 4 MHz dla Z80. Skłoniło to Intela do skierowania wysiłków w innym kierunku — wybór padł na procesory 16-bitowe.

Nie była to nowość. Firma National Semiconductor zaczęła prace nad procesorem IMP-16 już w 1972 r. W oparciu o pojedynczy mikroukład (nazwany Pace), firma Godbout Electronics w 1975 r. wyprodukowała komputer zaprojektowany przez George'a Morrowa. Nie nazwany system z pamięcią RAM i z wbudowanym sprzęgiem do magnetofonu, reklamowano w pierwszym numerze czasopisma Byte. Ani Pace, ani oparty na nim mikrokomputer nie odniosły sukcesu. Bill Godbout twierdził, że rynek nie był jeszcze wtedy przygotowany na przyjęcie mikrokomputerów 16-bitowych. Pojawiły się za to następne: LSI-16 firmy General Automation, LSI-11 firmy DEC, WD-11 firmy Western Digital, CP1600 firmy General Instruments i TMS 9900 firmy Texas Instruments. Zauważono, że wszystkie nowe procesory 16-bitowe były niewypałami, ponieważ nie było dla nich łatwo osiągalnego oprogramowania. Po prostu były to pierwsze mikroprocesory oferowane przez wytwórcę.

Intel zwietrzył szansę powodzenia w coraz szerszej bazie oprogramowania mikroprocesora 8080. Firma zadecydowała, że nowy 16-bitowy procesor ma być prostym rozwinięciem 8080. Rejestry mikroprocesora 16-bitowego były odpowiednikami rejestrów 8-bitowego, co umożliwiało łatwe translowanie kodu z 8080 na 8086. Odpowiedzią na pytanie, czy można robić sprzęt o tak dużych możliwościach adaptacyjnych jak oprogramowanie, było powstanie mikroprocesora 8088, stanowiącego w zasadzie 8086 z 8-bitową szyną danych, wyprowadzoną na zewnątrz. Wraz z rodziną 8086 Intel wprowadził nową ideę — koprocesor. Najbardziej znaczący jest koprocesor arytmetyczny 8087 z kompletną listą rozkazów arytmetyki zmiennoprzecinkowej.

ROZWÓJ MIKROPROCESORÓW 16-BITOWYCH

Od 1977 r. projektanci Motoroli pracowali nad 16-bitowym procesorem z usuniętymi rozkazami specjalnymi, z możliwością pracy we wszystkich trybach adresowania, na wszystkich rejestrach, na wszystkich rodzajach danych i z każdym rodzajem operacji. Taka właściwość nazywa się ortogonalnością. Programiści lubią ortogonalność, ponieważ nie muszą pamiętać setek wyjątków listy rozkazów. W 1979 r. wysiłki Motoroli zostały uwieńczone powodzeniem i pojawił się mikroprocesor MC 68000. W porównaniu do 8086/8088 wymaga on dużo większego oprogramowania. Wprowadzono również procesor 68008, odmianę 68000 z 8-bitową magistralą (koncepcja podobna do 8086). Jednakże ze względu na brak rzeczywistej kolejki rozkazów w 68000, procesor 68008 był od niego o połowę wolniejszy.

Mikroprocesor 68000 jest układem mikroprogramowanym, co znaczy, że wszystkie jego wewnętrzne elementy są funkcjonalnie uniwersalne. Odpowiedź procesora na każdy rozkaz jest formowana w pamięci ROM zawierającej mikroprogram. Jeśli jakiś rozkaz nie jest poprawny (użyteczny), to można umieścić jego kod w ROM, a nawet zmienić listę rozkazów mikroprocesora (oczywiście w ramach pewnych granic).

Wszystkie produkowane do tej pory mikroprocesory były typu random logic. Obydwa rozwiązania mają swoje plusy i minusy: mikroprogramowanie za cenę szybkości daje elastyczność systemu, random logic odwrotnie. W konstrukcjach random logic trudniej jest ustalić błąd, szczególnie wtedy, gdy projektant układu opuszcza firmę.

Przykładowo, Masatoshi Shima rozpoczął pracę nad 16-bitowym procesorem Z8000 stosując technikę random logic, która sprawdziła się przy projektowaniu 8080 i Z80. Po wykonaniu pierwszych egzemplarzy Z8000 w krzemie, ale przed ustaleniem wszystkich wad, Shima wrócił do Intela. Z tego względu firmie Zilog nigdy nie udało się wykryć wszystkich wad Z8000. Ten fakt, niepodobieństwo Z8000 do Z80 i jego natura random logic, są przyczynami niepowodzenia całego przedsięwzięcia.

W 1981 r. firma National Semiconductor uczyniła następną próbę wejścia na rynek z mikroprocesorem 16032. Miał to być 32-bitowy (szerokość wewnętrznej magistrali danych) mikroprocesor z 16-bitową magistralą zewnętrzną. Ponieważ Motorola nie mogła sobie poradzić z koprocesorem arytmetycznym, a koprocesor 8087 Intela nie mógł przekroczyć bariery 5 MHz, wszystkim zaimponował koprocesor pracujący dwa razy szybciej niż 8087, tj. z częstotliwością 10 MHz.

Niestety, National był pierwszą firmą mikroprocesorową sprzedającą wszystkie układy zewnętrzne, zanim osiągalny był sam mikroprocesor. W związku z tym 16032 ma kilka wad, ale programiści lubią jego listę rozkazów, przypominającą im VAX (serie superminikomputerów firmy DEC). Ponieważ VAX okazał się bardzo odpowiedni do pracy pod nadzorem UNIXA, 16032 może być naturalnym sukcesorem tego rodzaju komputerów.

W 1982 r. Intel wprowadził procesor iAPX 286. Ten nowy produkt jest rozwinięciem mikroprocesora 8086, zawierającym układ zarządzania wbudowaną pamięcią wirtualną i wiele innych cech przewidzianych do pracy w środowisku wielu użytkowników i wielu zadań. Posiada tryb pracy, w którym realizowane są wszystkie rozkazy 8086, co znacznie zwiększa jego przepustowość. Oczywiście wbudowane zarządzanie pamięcią pozwala na znacznie szybszą pracę niż w przypadku mikroprocesora z zarządzaniem zewnętrznym. Zastosowanie mikroprocesorów 80286 w komputerach osobistych firmy IBM zapewnia powodzenie tej linii mikroprocesorów na następnych kilka lat.

UKŁADY 32-BITOWE

Pomimo kłopotów z procesorem 16032 National Semiconductor była pierwszą firmą, która zbudowała pełny (ang. full-blown) 32-bitowy mikroprocesor, opatrzony symbolem 32032. Jest on zgodny programowo z 32016, ale nie można jeszcze przepowiedzieć jego przyszłości, choć prawdopodobnie popularność UNIXA będzie gwarantem jego sukcesu.

Motorola próbuje wprowadzić mikroprocesor 68020, będący rozwinięciem 68000. Ma on jedną cechę, która na pewno będzie charakteryzowała wszystkie mikroprocesory przyszłości. Pomni sukcesu procesora 8088, projektanci Motoroli zaprojektowali 68020 z możliwością dynamicznego wyboru szerokości magistrali (8, 16 lub 32 bity). Może on wykonywać całe oprogramowanie dla 68000 i robi to dużo szybciej. Możliwość osiągnięcia tej szybkości daje kryjówka (ang. cache) — logiczne rozwinięcie kolejki użytej w procesorach 8086/8088/80286.

Kryjówka procesora 68020 ma głębokość 256 bajtów i działa trochę inaczej niż kolejka. Jeśli następuje skok do punktu w kolejce, to kolejka jest oczyszczana i ładowana ponownie. Działanie kryjówki wygląda podobnie do pamięci. Skok do punktu w kryjówce nie powoduje jej oczyszczenia ani ponownego załadowania. Gdy pętle są wystarczająco małe, mogą być wykonywane prosto z kryjówki. Zaletą takiego rozwiązania jest szybszy dostęp procesora do kryjówki niż do pamięci zewnętrznej, dzięki czemu programy przechowywane w kryjówce są szybciej realizowane. Firma National zapowiedziała procesor 32132, który ma mieć wielokrotne kryjówki i pewien szczególny rodzaj bufora.

Firma Zilog zapowiedziała kolejną nowość Z800 — 16-bitową, ulepszoną wersję Z80, a poza tym próbuje przeprojektować Z80 na pracę z szybkością 10 MHz. W planach firmy jest jeszcze mikroprocesor 780000 — 32-bitowa wersja Z800 — złośliwie nazywany „procesorem zlewozmywakowym" — ponieważ będzie miał wszystko, włączając kompatybilność z Z8000, co jest dobrym pomysłem, ale ze złym procesorem.

W firmie Intel mówi się o procesorze 80386 — 32-bitowej odmianie mikroprocesora 80286, ale nie opublikowano jeszcze dokładniejszych danych na jego temat, poza tym, że ma być kompatybilny z 80286, co oznacza możliwość wykorzystywania go w komputerach osobistych IBM.

* * *

Ostatnio jesteśmy świadkami dyskusji, czy lista rozkazów mikroprocesora powinna ewoluować w stronę bardziej skomplikowanych instrukcji wysokiego poziomu, czy powinny one być prostsze i szybsze. Jak dotąd nie można powiedzieć, który z tych dwóch kierunków zwycięży. Jednym z bardziej szalonych pomysłów było stworzenie Pascal Microengine firmy Western Digital, wykonującej bezpośrednio pseudokod Pascala. Niestety wyparły ją całkowicie kompilatory Pascala, generujące kod wrodzony mikroprocesora 8080.

Podobny w zamyśle jest procesor Fortha firmy Novix, rozwijany pod kierunkiem projektanta języka Charlesa Moore'a. Zamiast wykonywać kod skompilowany, procesor akceptuje instrukcje Fortha, ponieważ lista rozkazów tego układu odpowiada słowom języka. Kolejna maszyna Forth jest odmianą układu NCR-32, pracująca w mikrokodzie na dwóch układach, z trzecim zawierającym sam mikrokod. Zasadniczo można do niej włączyć własną listę rozkazów, co otwiera całkiem nowe możliwości, takie jak emulacja różnych komputerów, przez załadowanie różnych list rozkazów.

Procesor iAPX 432 Intela był prawdopodobnie pierwszym dostępnym mikroprocesorem 32-bitowym. Został on oparty na liście rozkazów bardzo wysokiego poziomu, w większości zgodnej ze zbiorem instrukcji języka Ada.

W opozycji do zwolenników list rozkazów wysokiego poziomu są projektanci preferujący listy zawierające proste, lecz szybkie rozkazy, określane mianem RISC (ang. Reduced Instruction Set Computer). Nad zwiększeniem szybkości układu RISC o 32-bitowej architekturze pracuje się m. in. w Berkeley.

Ostatnim godnym uwagi urządzeniem jest Transputer firmy INMOS, zaprojektowany do przetwarzania równoległego. Trzeba jednak włożyć wiele pracy w rozwój oprogramowania, zanim uda się w pełni wykorzystać możliwości Transputera.

Opracował: **MARIUSZ KUC**

na podstawie BYTE, Vol. 10 Nr. 9, September 1986

Sir Clive Sinclair o trzeciej rewolucji przemysłowej

Jest to tekst przemówienia wygłoszonego w marcu 1984 r. w jednej z agend Kongresu USA (US Congressional Clearinghouse ou the Future). Artykuł ten nawiązuje do poruszanej na naszych łamach (przed dwoma laty) tematyki Raportu dla Klubu Rzymskiego pt. „Mikroelektronika a społeczeństwo", opublikowanego właśnie przez Książkę i Wiedzę.

Mówi się, że wkroczyliśmy w okres drugiej rewolucji przemysłowej. Myśl ta wydawała się rewolucyjna wczoraj, dziś już jest banałem. Uważam, że trzeba rozpatrywać proces rozwoju technologicznego jako trzecią, a nie drugą rewolucję przemysłową.

Moim zdaniem, pierwsza rozpoczęła się wtedy, gdy ludzie nauczyli się siać i zbierać owoce swej pracy. Tym samym skończył się okres życia koczowniczego, kiedy to człowiek poświęcał większość czasu na zdobywanie żywności. Farmer (uczestnik pierwszej z tych rewolucji), który posiadał własne gospodarstwo, mógł wyżywić siebie i kilka innych osób nie będących rolnikami, uwalniając ich od potrzeby zdobywania pożywienia. Tym samym, mogli oni zająć się inną działalnością — wytwarzaniem sprzętów potrzebnych w gospodarstwie (łopat, wiader, misek), środków transportu (pojazdów, statków), na których inni mogli poznawać świat. Ludzie zaczęli pisać i rejestrować wydarzenia, tworzyć prawa i chronić terytoria przed napaścią wrogów. Wielu z nich tęskniło za łatwym i niewinnym życiem, czego odzwierciedleniem może być historia o Raju, lecz ludność zwiększała się liczbowo i rozprzestrzeniała się, a zmian nie dało się zatrzymać. Nie było odwrotu.

Druga wielka przemiana nastąpiła pod koniec XVIII w., kiedy to ludzkość nauczyła się wytwarzać przedmioty za pomocą maszyn, a nie jak dotychczas — narzędzi ręcznych. Co prawda zmiany te następowały stopniowo, gdyż już w czasach rzymskich można znaleźć wiele przykładów przemysłu na dużą skalę. Momentem zwrotnym było wynalezienie napędu parowego, dzięki czemu powstały maszyny dla przemysłu i napęd dla środków transportu. Energie zawarte w węglu zastąpiły energię wiatru. Człowiek zaczął podróżować jeszcze dalej, bronił jeszcze większych terytoriów, znów ludzie zaczęli tęsknić za pełną prostoty przeszłością, która istniała bardziej w ich wyobraźni niż naprawdę. Dzięki tym, którzy więcej wytwarzali, coraz więcej innych mogło zająć się badaniami prowadzącymi do postępu nauki.

W ten sposób dotarliśmy do trzeciej wielkiej przemiany, zwanej drugą rewolucją przemysłową. Jest ona częściowo związana z zastępowaniem ludzi przez roboty i komputery. Częściowo odpowiada skokowemu postępowi w przetwarzaniu i przesyłaniu informacji. W ten czy inny sposób jest ona więc związana z komputerami. Dzięki nim — w przyszłości — znowu miliony ludzi będzie mogło przystosować się do innych zawodów. Z jednego punktu widzenia będą oni uwolnieni od mozolnej pracy w fabryce, w sensie negatywnym zostaną pozbawieni pracy. Jest to smutna konsekwencja postępu i nie jesteśmy jeszcze w stanie temu zapobiec. Wierzę, że są to skutki tymczasowe, spowodowane przez niewiarygodnie szybki spadek liczby zatrudnionych w przemyśle. O ile w latach czterdziestych 50% ogółu zatrudnionych pracowało w fabrykach, to, jak oblicza się obecnie, mniej niż 10% pozostanie w nich pół wieku później.

Takie spojrzenie na świat, w którym żyjemy, staje się coraz bardziej rozpowszechnione. Jeśli jednak skoncentrujemy się na analogii z rewolucją przemysłową, to stracimy z oczu inną, o wiele bardziej doniosłą analogię. Zamiast cofać się w porównaniach w wieki i tysiąclecia, cofnę się w przeszłość milion razy dalej, dalej niż do początków cywilizacji.

Cztery miliardy lat temu, gdy Wszechświat był o połowę mniejszy niż teraz, a nasz System Słoneczny miał tylko 5 mln lat, powstała osobliwa rzecz — życie. Dzięki pewnym nieuniknionym procesom zachodzącym w pierwotnej magmie, mieszanej przez gwałtowne promieniowanie kosmiczne, formowały się i przetwarzały związki węgla o niespotykanej złożoności, aż osiągnęły zdolność przetwarzania światła słonecznego i reprodukcji. Przez miliard lat, te pierwsze bakterie, tak tajemniczo wyczarowane, zbite razem w żyjące rafy, były jedynym życiem. Trzy miliardy lat później rozwinęły się one w rodzaj ludzki.

Powiedziałem, że wydarzenie, które rozpoczęło ten proces, było wyjątkowe, i z tego, co wiemy, takie było rzeczywiście. Ale długo takim nie pozostanie. Całe życie jest oparte na węglu i węgiel jest wyjątkowy pod względem różnorodności związków, do których prowadzi, dostarczając organizmowi szerokiego wyboru materiałów budulcowych. Jeśli kiedykolwiek odkryjemy życie na innych planetach, to nie będziemy zaskoczeni, że jest ono także oparte na węglu. Może się jednak zdarzyć, że tak wcale nie będzie. Kiedy byłem chłopcem, czytałem opowiadania fantastyczne, których częstym tematem było odkrywanie form życia znacznie różniących się od naszego. Popularną ideą było w nich życie oparte nie na związkach węgla lecz — krzemu, jako że krzem może również tworzyć bogactwo produktów o dużej użyteczności. Wydaje się, że wkrótce te opowiadania staną się prawdziwe, ponieważ powstanie życie oparte na krzemie. Nie wyłoni się ono metodą trwających miliony lat prób w energetycznej protoplazmie, lecz — wskutek ludzkich zabiegów w ciągu zaledwie wieku. Droga, na której znajduje się przemysł elektroniczny oparty na krzemie — prowadzi do życia.

Ludzki mózg zawiera — jak mi powiedziano — 10 mld komórek, a każda z nich może mieć tysiące połączeń. Taka olbrzymia liczba przerażała nas i spowodowała zaniechanie budowy maszyny o zdolnościach podobnych do ludzkich. Wkrótce jednak, może za 10 czy 20 lat, będziemy w stanie zbudować maszynę tak skomplikowaną jak ludzki mózg, a jeśli będziemy mogli, to zbudujemy. Wiele czasu może zająć uczynienie jej inteligentną przez odpowiednie oprogramowanie lub przez zmianę architektury, ale to także nastąpi.

Uważam za pewne, że w ciągu nie wieków, lecz dziesięcioleci — powstaną krzemowe maszyny zdolne rywalizować z człowiekiem, a następnie prześcignąć swych ludzkich przodków. W chwili, kiedy nas prześcignę, staną się zdolne do projektowania siebie samych, do reprodukcji. Krzem zakończy długi okres monopolu węgla. Przypuszczam, że zakończy się również era naszego monopolu, jako że nie będziemy mogli dłużej uważać się za najbardziej inteligentną istotę we Wszechświecie. W zasadzie proces ten można by zatrzymać, znajdą się też tacy, którzy będą próbowali to zrobić, ale bezskutecznie. Puszka Pandory zaczyna się otwierać.

Spójrzmy teraz trochę bliżej na teraźniejszość. Przed końcem tego dziesięciolecia w Wielkiej Brytanii przypuszczalnie zakończy się kryzys w produkcji, przy poziomie zatrudnienia w przemyśle wytwórczym mniejszym niż 10%. Import i przemiany technologiczne będą jednak dalej obniżać zatrudnienie prawie do zera, jak to się dzieje teraz w rolnictwie.

Rozmowy o technologii informacyjnej mogą być nieco mylące. Prawdą jest, że w najbliższych latach nastąpi gwałtowny, być może nawet stukrotny spadek kosztów publikowania, dzięki zastosowaniu nowych technik, jak np. dyski optyczne itp., które zastępują papier. Może być to nawet tak ważne jak wynalezienie słowa pisanego czy wprowadzenie ruchomego druku przez Caslona. Jednakże, nazwa „technologia informacyjna" prowadzi do pewnych nieporozumień, ponieważ termin ten dotyczy raczej przetwarzania informacji, a nie obsługiwania maszyn, a to jest element o znaczeniu fundamentalnym. Rewolucja, która właśnie się rozpoczyna, dotyczy rozwoju ludzkiej inteligencji. Elektronika zastępuje ludzki umysł, tak jak para zastąpiła ludzkie mięśnie. Japończycy, ze swoim programem

ICOT, zamierzają produkować komputery operujące pojęciami, a nie liczbami, tysiące razy sprawniejsze niż współczesne duże maszyny. Stanowi to duży bodziec do dokonania szybkiego postępu w USA. Amerykańskie firmy komputerowe uczestniczą w dużym wspólnym programie rozwojowym, co najmniej tak obszernym jak wojskowy program DARPA, zaś IBM, chociaż niczego na ten temat nie ujawnia, może mieć największy program ze wszystkich.

Celem tych projektów jest stworzenie tego, co ogólnie określa się jako komputery piątej generacji. Jest to całkowicie nowy rodzaj maszyn, które będą różniły się od dzisiejszych komputerów, tak jak współczesne różnią się od zwykłych sumatorów. Dzięki rozwojowi przemysłu półprzewodnikowego nie będą one przesadnie drogie. Gdy staną się dostępne, zaczną zastępować ludzką inteligencję na wyższym poziomie abstrakcji.

Współczesne roboty montażowe (ang. assembly line robots) są wyposażone w dostateczną inteligencję przez proste mikroprocesory. Roboty rozwiną jednak swoje mózgi, kiedy nauczą się widzieć i czuć. Wreszcie, w niezbyt odległej przyszłości, będą one podejmować decyzje na linii produkcyjnej, zastępując nadzór techniczny.

Poza fabryką ludzki umysł jest wykorzystywany w dwa zasadnicze sposoby: jako źródło wiedzy i źródło podejmowania decyzji. Pierwsza z tych właściwości przejmują obecnie komputery, na skutek rozwoju systemów ekspertowych, dzięki którym wiedza osiągnięta przez człowieka, np. eksperta w dziedzinie górnictwa, może być gromadzona w pamięci komputera. Przepływ danych od człowieka do komputera nie jest ani łatwy, ani szybki, lecz dane te raz uzyskane mogą być dowolnie powielane i rozglaszane. Zdolność do wyciągania logicznych wniosków, których spodziewamy się od lekarza czy prawnika, z wystarczających lub niepełnych danych, będzie jeszcze nadal monopolem człowieka, lecz nie na zawsze.

Przywilej ten będą miały komputery piątej generacji. Może już jutro będziemy wybierać się z naszymi dolegliwościami do komputera równie chętnie jak do lekarza. Taki komputer będziemy mieli na miejscu, w domu, co pozwoli nam uniknąć zbędnej podróży do lekarza, umożliwi też prowadzenie bardzo regularnej kontroli naszego stanu zdrowia.

Komputer w roli nauczyciela może przynieść jeszcze więcej korzyści. Tak długo, jak długo będziemy pod tym względem zależni od ludzi, jeden nauczyciel będzie uczył wielu uczniów. Korzyść z posiadania jednego wychowawcy dla każdego dziecka jest oczywista i jeśli ten wychowawca będzie nieskończenie cierpliwy i poinformowany lepiej od człowieka, to możemy spodziewać się cudownej poprawy w poziomie wykształcenia. Dokąd jednak to wszystko prowadzi, skoro w najbliższej przyszłości nie będzie miejsc pracy? Ciekawe, że pewną analogię można znaleźć w przeszłości. Ludzie wolni w Atenach za czasów Peryklesa wiedli życie nie tak bardzo odmienne od naszego, jakie możemy prowadzić, posiadając inteligentne maszyny. Mieli oni niewolników, którzy byli ich nauczycielami i służącymi. Prawdopodobnie dzięki wspaniałemu wychowaniu, ludzie wolni w Atenach nie mieli trudności ze spędzeniem swego wolnego czasu. Tak samo my musimy wychowywać nasze dzieci, by doceniały subtelne aspekty życia, umiały kochać sztukę, muzykę i naukę. My też możemy doświadczyć wieku tak złotego, jak starożytni Grecy.

Maszyny będą zdolne zastąpić człowieka wszędzie tam, gdzie jest wymagane wykonywanie złożonych funkcji. Myślę, że łatwiej jest skonstruować maszynę uczącą matematyki czy łaciny niż grającą w tenisa, jako że to drugie wymaga zdumiewającego refleksu i szybkiej decyzji w połączeniu z precyzyjnym wykonaniem. Jednakże, taka maszyna może być i będzie skonstruowana, choć nie po to, aby pozbawić nas przyjemności gry w tenisa, lecz aby uchronić nas przed monotonią i niebezpieczeństwami związanymi z prowadzeniem samochodu. Samochód daje nam poczucie wolności, umożliwiając przemieszczanie się w dowolne miejsca o każdej porze. Ceną, którą za to płacimy, jest zwiększenie śmiertelności wśród ludzi i zanieczyszczenie środowiska. Zdecydowaliśmy się jednak zmniejszyć o ponad połowę prędkość samochodów, aby zredukować te ujemne skutki. Przyszłość przyniesie lepsze rozwiązania. Osobiście przewiduję całkowitą automatyzację samochodów osobowych, które nadal będą zapewniały przemieszczanie się w dowolne miejsca o dowolnej porze, lecz sterowane będą przez inteligentne automaty. W miastach i na drogach podrzędnych będą one napędzane elektrycznością pobieraną z wewnętrznych baterii, zaś na autostradach — z centralnego zasilania, przypuszczalnie przez sprzężenie indukcyjne. Samochody przyszłości będą całkowicie niehałaśliwe, a co najważniejsze, wolne od ludzkiej zawodności. Tak więc, nie będą wymagały ograniczeń prędkości do 55—70 mil na godzinę na drogach głównych. Bezpieczna i ekonomicznie uzasadniona będzie prędkość powyżej 200 mil na godzinę. Przesuwanie magnetyczne zastąpi prawdopodobnie koła, dzięki czemu zwiększy się komfort jazdy, jak też długowieczność pojazdów, które nie będą potrzebowały regularnego serwisu, ponieważ nie będą miały części ruchomych.

Jednym z najnowszych systemów komunikacji umożliwiających ludziom porozumiewanie się, jest tzw. komórkowy system radiowy (ang. cellular radio system), rozwijany w pewnych miastach amerykańskich. Myślę, że jest to częściowe rozwiązanie ogólnego problemu, polegającego na umożliwieniu ludziom telefonowania do siebie, niezależnie od miejsca i czasu. Obecnie istnieją już tylko ekonomiczne, nie techniczne, ograniczenia nie pozwalające na wprowadzenie telefonów osobistych. Takie bezprzewodowe telefony, noszone przy sobie, pozwoliłyby na komunikowanie się niezależnie od aktualnego miejsca pobytu. Nie trzeba by wiedzieć, gdzie znajduje się osoba, do której chcemy zadzwonić, wystarczyłoby tylko znać jej osobisty numer. Wierzę, że jest to osiągalne przez rozszerzenie systemu komórkowego w przestrzeni i zwiększenie jego pojemności, co wymaga zwiększenia stopnia ziarnistości w systemie, a więc gęstszego rozmieszczenia nadajników-odbiorników (ang. transceivers).

Często wydaje się, że każdy krok naprzód w technice przynosi raczej rozczarowanie niż zadowolenie. Jest tak dlatego, że każdy postęp przynosi najpierw zmiany, a dopiero później korzyści, a każda zmiana, chociaż często pobudza do działania, zawsze wywołuje niepokój. Właśnie tak jest z rewolucją związaną z rozwojem sztucznej inteligencji, lecz w tym wypadku korzyści przewyższają niepowodzenia. Już wkrótce okazać się może, że nawet nasze najbardziej dokuczliwe problemy zostaną rozwiązane. Rozpatrzmy, na przykład, sprawę odosabniania przestępców. Jeśli nie robimy tego z chęci biblijnego odwetu, to postępowanie takie ma na celu zredukowanie przestępczości przez odstraszanie i powstrzymywanie. Jest to jednak bardzo kosztowne, a skutki wychowawcze w stosunku do tempa recydywizmu są niewielkie.

Przestępcy, którzy nie są fizycznie niebezpieczni, mogliby być wyposażeni w miniaturowe aparaty, tak aby ich aktualne miejsce pobytu mogło być precyzyjnie określone i stale kontrolowane. Gdyby wywoływało to obawę przed powstaniem społeczeństwa orwellowskiego, należałoby dać przestępcom możliwość wyboru kary innej niż uwięzienie. Jestem przekonany, że sposób ten zyskałby powszechną akceptację.

Inteligentne roboty będą też opiekować się ludźmi starszymi, dla których mogą nawet stanowić towarzystwo. Nigdy nie zasypiające automaty mogłyby zaspokajać potrzeby fizyczne, jak też zajmować się opieką medyczną.

Ponieważ inteligencja robotów zaczyna rywalizować z ludzką, a ich koszty zmniejszają się wraz ze zwiększeniem skali zastosowań, możemy użyć ich do rozszerzania granic naszej działalności, przede wszystkim na Ziemi, wykorzystując ich zdolności pokonywania trudnych warunków. Tak więc dzięki nim pustynie mogą zakwitnąć, a dna oceanów stać się kopalniami. Co więcej, dzięki wzrastającemu bogactwu i nowoczesnej technice, które przyniesie ten nowy wiek, będziemy mogli zacząć wykorzystywać przestrzeń kosmiczną. Budowa rozległego świata w przestrzeni kosmicznej, dom dla tysięcy czy milionów ludzi, będzie w zasięgu naszych możliwości i tę drogę powinniśmy wybrać. Wtedy możemy z gorliwością zabrać się do szukania światów poza naszym Systemem Słonecznym i do kolonizacji Galaktyki.

Opracowała:
ROMA BARLAK

Standaryzacja języka BASIC

W dniach 16—18 października 1985 roku w Londynie odbyło się kolejne spotkanie grupy roboczej ISO/TC97/ /SC22 WG8. Celem spotkania było przedyskutowanie ostatecznej postaci języka BASIC przeznaczonej do zatwierdzenia przez ECMA, a następnie przez ISO.

Historia języka BASIC rozpoczęła się w latach 1963—1964 w Dartmouth College w USA. Dwaj profesorowie, John Kemeny i Tom Kurtz zaprojektowali bardzo prosty język dla osób początkujących w programowaniu — stąd jego nazwa Beginners All-purpose Symbolic Instruction Code, tworząca akronim BASIC. Cechy jego pierwszej wersji były następujące:

- zmienne samodzielne określają swój typ
- wykorzystanie numeracji linii do edycji i etykietowania,
- pojedynczy typ danych numerycznych,
- prosty format wprowadzania danych i wyprowadzania wartości.

Język ten został zaimplementowany jako konwersacyjny w systemie Dartmouth Timesharing System dla komputera GE256.

W kolejnych latach pierwotna wersja była rozszerzana o nowe konstrukcje [3]. W 1966 roku wprowadzono instrukcje operowania tablicami i funkcje, w 1967 roku — instrukcje zmiany wartości początkowej generatora pseudolosowego i skoku wyliczanego, a następnie wieloliniowych definicji funkcji, instrukcje obsługi plików i łączenia programów. Dwa lata później opracowano wersję języka z instrukcjami strukturalnymi oraz funkcje tekstowe. W następnym roku wprowadzono instrukcje wprowadzania linii tekstu (także z plików) oraz definicje podprogramów. Rozszerzenia te w 1971 roku były modyfikowane i poprawione. Kolejne większe zmiany nastąpiły dopiero w 1979 roku, w ostatnim etapie tworzenia standardu. Ten skrócony i bardzo pobieżny opis rozwoju języka dotyczy tylko jego prawdziwej wersji — tak twierdzą autorzy BASICA, a nie setek jego mutacji powstałych szczególnie dla potrzeb systemów mikrokomputerowych. Jedną z ostatnich wersji języka BASIC opisano w [8].

*) Według polskich zasad pisowni nazw własnych będących akronimami (skrótowcem literowym) nazwę języka BASIC powinniśmy pisać dużymi literami. Możliwe jest też potraktowanie tej nazwy jako ogólnej nazwy języka (nie związanej z żadną firmą lub implementacją) pisanej w postaci nazwy własnej Basic.

Z początkiem lat siedemdziesiątych rozpoczęły się zdecydowane działania w kierunku standaryzacji BASICA. W styczniu 1974 roku w ramach ANSI utworzono podkomitet X3J2 ds. standaryzacji, a w tym samym roku podobny Komitet TC 21 pod egidą ECMA (European Computer Manufactures Association). Jednocześnie w ramach działalności innych instytucji rozpoczęto prace standaryzacyjne nad fragmentami języka dotyczącymi czasu rzeczywistego (IFIP TC5 WG2) oraz system CAMAC (ESONE, NIM) [6]. Od 1976 roku wszystkie te komitety i grupy robocze postanowiły ze sobą ściśle współpracować.

Pierwsza norma języka Minimal BASIC została opublikowana w 1978 roku równocześnie przez ANSI [1], ECMA [4] oraz ISO [7]. Dokumenty te opisują identyczną wersję języka, ale różnią się w formach opisu. W kolejnych latach pracowano nad normą rozszerzonego BASICA, Enhanced BASIC. Prace te zbliżają się obecnie do końca i ich wynikiem ma być zatwierdzenie dwóch wersji normy — jednej dla ANSI [2], a drugiej dla ECMA i ISO [5]. Różnica między tymi wersjami polega na sposobie podziału języka na poziomy lub moduły. Rysunek przedstawia schematycznie obie wersje. Zaznaczono na nim tylko podstawowe elementy języka. W obu wersjach w podstawowym module (przeznaczonym do zlokalizowania w pamięci operacyjnej) zawarte są pełne konstrukcje instrukcji strukturalnych oraz definicji podprogramów. W wersji BASIC ISO/ECMA wyróżniono dwa poziomy modułu podstawowego — jeden z ograniczoną obsługą sytuacji wyjątkowych oraz z zastrzeżeniem identyfikatorów słów kluczowych. Poza obecną wersją normy znajdują się propozycje do dalszych rozszerzeń języka, na przykład złożone typy danych lub obsługa bazy danych. W celu ułatwienia przenoszenia programów zapisanych w różnych wersjach norm — opis języka jest podany w tej samej formie i notacji z wyszczególnieniem pośrednich różnic. Pełne wersje języka, według obu norm, w założeniu mają być identyczne.

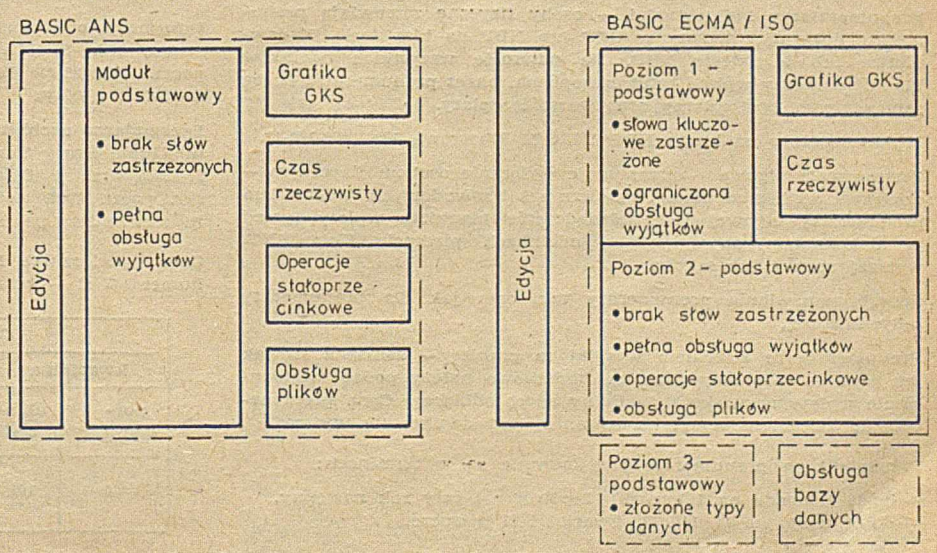
Forma opisu standardu ma klasyczną postać wykorzystującą do specyfikacji składni języka notację BNF. Semantyka języka jest opisana słownie, z wykorzystaniem przykładów. Uzupełnieniem opisu jest interpretacja sytuacji wyjątkowych, które mogą powstać w trakcie wykonywania programu.

Podstawowym ustaleniem w normie języka jest określenie reguł stwierdzania zgodności (ang. conformance) z normą. Wyróżniono dwa aspekty zgodności programu z definicją języka oraz implementacji przetwarzającej taki program. Zapis tekstu programu jest zgodny z normą języka, jeżeli tekst programu, każda jego instrukcja i jej elementy spełniają reguły składni wyspecyfikowane w normie. Jednocześnie cały program nie może naruszać żadnych ustaleń zawartych w normie dotyczących reguł składni.

Implementacja języka jest zgodna z normą tylko wtedy, gdy translator akceptuje i przetwarza wszystkie programy zgodne z normą oraz dodatkowo spełnione są następujące warunki:

- przyczyny niezgodności programu z normą są sygnalizowane,
- interpretacja błędów i sytuacji wyjątkowych jest zgodna ze specyfikacją w normie,
- interpretacja semantyczna każdej instrukcji zgodnej z normą jest również zgodna z normą,
- wprowadzenie, interpretacja i wyprowadzenie wartości liczbowych i tekstowych jest dokonywane zgodnie z normą,
- istnieje dokumentacja dostępna dla użytkownika, opisująca wszystkie działania określone w normie jako niezdefiniowane lub zdefiniowane implementacyjnie.

Przytoczona za raportem [2, 5] definicja zgodności z normą pozwala stwierdzić, że dopuszczalne są rozszerzenia języka względem opisanego w normie. W każdym jednak przypadku wykonanie tekstu programu zapisanego zgodnie z normą musi dać iden-



Struktura modularności obu wersji języka BASIC

tyczne wyniki dla każdej implementacji zgodnej z tą normą. Rozszerzenia względem normy muszą być ujęte w osobnym dokumencie dostępnym dla użytkownika, omawiającym również ich interpretację.

Opracowanie normy definiującej rozszerzoną postać języka BASIC zajmuje już ponad 7 lat. Wydaje się, że okres ten jest zbyt długi, z uwagi na postępy dokonane w tym czasie w programowaniu. Należy jednak zwrócić uwagę, że podczas standaryzacji nie można się opierać tylko na arbitralnych decyzjach jakiejś grupy osób. Powstanie normy musi zyskać pełną akceptację liczącego się świata informatyki. Szczególnie BASIC jest obiektem trudnym do standaryzacji. W swojej dwudziestoletniej historii obrósł on bowiem setkami rozszerzeń i modyfikacji. Akceptacja lub pominięcie wielu tych modyfikacji musi być wynikiem długotrwałych ustaleń między różnymi, ścierającymi się grupami. W trakcie tych dyskusji konieczne jest udzielenie odpowiedzi na podstawowe pytanie: jaki ma być kształt języka BASIC, aby nie zgubić jego podstawowej idei polegającej na prostocie. Z drugiej strony musi on stanowić narzędzie programowania o mocy i efektywności porównywalnej z innymi nowoczesnymi językami programowania.

Analizując obecną wersję propozycji normy języka można stwierdzić, że jego kształt jest już ustalony. Proponowany w normie BASIC łączy w sobie cechy jego pierwotnej wersji przeznaczonej dla początkujących programistów z wymaganiami poprawnego stylu programowania, charakteryzującego się proceduralnością, modularnością i strukturalnością programu. Jednocześnie język jest w miarę uniwersalnym narzędziem umożliwiającym wykorzystanie go do prostych i złożonych obliczeń numerycznych, przetwarzania danych administracyjnych, konstruowania obrazów graficznych zgodnie z zasadami GKS [10] oraz programowania systemów w czasie rzeczywistym. Nowe metody implementacji języka, oparte na interpretacji kodów pośrednich, otrzymanych drogą kompilacji przyrostowej oraz wykorzystujące sprzętowe jednostki zmiennoprzecinkowe, pozwalają uzyskać zadowalającą efektywność i szybkość translacji oraz wykonania obliczeń.

Obecny etap standaryzacji języka można więc uznać prawie za zamknięty. W warunkach krajowych pozostaje nam więc zaadaptowanie normy do wymagań polskich przez zdefiniowanie zawartych w niej ustaleń [2, 5] — takich jak uporządkowanie znaków alfabetu rodzimego w tekstach i specyfikacja plików rodzimych, zależnie od środowiska, dla którego będzie realizowana implementacja. Między innymi rodzi się tutaj pytanie, czy wzorem francuskim nie można dokonać pełnego tłumaczenia normy (łącznie ze słowami kluczowymi) na język polski i — implementować następnie obu tych wersji [9].

Proces standaryzacji języka nie zostanie definitywnie zakończony. Przewiduje się, że w ciągu następnych lat będzie przygotowywana jego następna, rozszerzona wersja. Główne kierunki tych rozszerzeń będą zmierzać do wprowadzenia agregatów struktur danych, dalszej modulacji programu z podziałem na moduły oraz — przystosowania języka dla potrzeb różnego rodzaju systemów wieloprocesorowych. Porównując tendencje rozwoju języków programowania można sądzić, że w przyszłości będziemy posługiwać się jednym językiem w zakresie języków typu proceduralnego. A to, czy będzie on nazywał się Fortran 88 czy Full Enhamed BASIC lub Ada daughter, nie będzie miało większego znaczenia.

WACŁAW ISZKOWSKI

LITERATURA

[1] ANSI: Programming Language Minimal BASIC Report ANSI X3.60, 1978

[2] ANSI: Dreft proposed American National Standard for BASIC Report ANSI X3J2/85-08, April 1985

[3] Bull G. M.: Basic. Programming Language Standarisation Hill I. D., Meek Bol., ed. Ellis Horwood, Chichester, 1982

[4] ECMA: Minimal Basic, Standard ECMA-55, 1978

[5] ECMA: Standard ECMA BASIC-1, Standard ECMA BASIC-2, ECMA/TC21, July 1985

[6] ESONE: Real-time Basic for CAMAC, Report ESONE/RTB/02, 1977

[7] ISO: (1978) Draft standard for Minimal Basic. Draft DP6573 ISO/TC97/SC5 N447. Part 1

[8] Iszkowski W., Maniecki M.: Standaryzujący język BASIC, INFORMATYKA nr 5, 6, 7—8, 1983

[9] Iszkowski W.: A jednak po polsku, INFORMATYKA, 1985

[10] Urban A.: Norma GKS, INFORMATYKA, nr 1, 1985.

WARUNKI PRENUMERATY NA 1986 R.

Prenumeratorzy zbiorowi — jednostki gospodarki uspołecznionej, instytucje i organizacje społeczne zamawiają prenumeratę dokonując wpłat na blankiecie „polecenie przelewu" rozszerzonym dla potrzeb Wydawnictwa o część dotyczącą zamówienia. Blankiety te będą dostarczane przez Zakład Kolportażu.

Prenumeratorzy indywidualni — osoby fizyczne zamawiają prenumeratę dokonując wpłaty w UPT lub NBP na blankiecie Wydawnictwa lub blankiecie NBP. Na odwrocie wszystkich odcinków blankietu należy wpisać tytuł czasopisma, okres prenumeraty, liczbę zamawianych egzemplarzy oraz wartość wpłaty.

Wpłacać należy na konto NBP III O/M Warszawa 1036-7490-139-11.

Prenumerata ulgowa — przysługuje wyłącznie osobom fizycznym — członkom SNT, studentom i uczniom szkół zawodowych. Warunkiem prenumeraty ulgowej jest poświadczenie blankietu wpłaty (przed jej dokonaniem) na wszystkich odcinkach pieczęcią Koła SNT, wyższej uczelni lub szkoły.

Sposób zamawiania prenumeraty taki sam jak dla prenumeraty indywidualnej.

Prenumerata ze zleceniem wysyłki za granicę — zamawia się tak jak prenumeratę indywidualną. Dodatkowo należy podać na blankiecie wpłaty nazwisko i dokładny adres odbiorcy. Cena prenumeraty ze zleceniem wysyłki za granicę jest dwukrotnie wyższa.

Przedpłaty na prenumeratę przyjmowane są w terminach:

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

U w a g a!

Wpłaty na dwumiesięczniki przyjmowane są na okresy półroczne lub roczne.

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

Egzemplarze archiwalne czasopism — można nabyć za gotówkę w Klubie Prasy Technicznej w Warszawie ul. Mazowiecka 12, tel. 27-43-65 oraz w Dziale Handlowym Wydawnictwa ul. Bartycka 20 skr. poczt. 1004, 00-950 Warszawa, na rachunek dla instytucji lub za zaliczeniem pocztowym dla osób fizycznych.

Cena miesięcznika INFORMATYKA została ustalona na 120 zł za numer (35 zł — cena ulgowa).

Cena prenumeraty wg cennika					
kwartalna		półroczna		roczna	
normalna	ulgowa	normalna	ulgowa	normalna	ulgowa
360	105	720	210	1440	420

Kto jest kim w IFIP

Pierre Bobillier

Po posiedzeniu Zgromadzenia Ogólnego, które odbyło się we wrześniu 1984 r., profesor Pierre Bobillier zakończył pracę w IFIP, której służył umiejętnościami jako delegat Szwajcarii od 1968 r. W latach 1968—1975 był sekretarzem; od 1975 do 1976 r. — członkiem zarządu, a następnie, jako jedyny w dotychczasowej historii IFIP, prezydentem tej organizacji przez dwie kadencje (1977—1983).

Profesor Bobillier studiował matematykę na Uniwersytecie w Lozannie i mechanikę techniczną w Szwajcarskim Federalnym Instytucie Technicznym. W 1953 r. rozpoczął pracę w Międzynarodowej Unii Telekomunikacyjnej w Genewie. Od 1957 r. pracował w szwajcarskiej filii IBM na różnych stanowiskach, zajmując się m. in. metodami komputerowymi w nauce, marketingiem dużych systemów i uniwersyteckimi sieciami komputerowymi. W 1961 r. został członkiem kierownictwa filii.

W latach 1966—1977 profesor Bobillier wykładał symulację i języki symulacyjne w IBM-owskim Instytucie Badań Systemowych w Genewie. Od 1966 r. prowadził kursy symulacji i metodyki badań w Szwajcarskim Federalnym Instytucie Technicznym w Lozannie, gdzie jest profesorem nadzwyczajnym. Jest autorem ponad 20 artykułów technicznych z dziedziny informatyki i badań operacyjnych oraz książki „Simulation with GPSS and GPSS V", wydanej przez Prentice-Hall w 1976 r.

W 1977 r. otrzymał srebrną odznakę IFIP, w 1980 r. — złoty medal Marina Drinova Bułgarskiej Akademii Nauk, a w 1981 r. — honorowe członkostwo Towarzystwa Informatycznego Japonii.

W sprawozdaniu końcowym dla Zgromadzenia Ogólnego Pierre Bobillier napisał: *Za największe osiągnięcia minionych lat uważam:*

- *Rozpoczęcie przez IFIP działalności w zupełnie nowej dziedzinie, jaką jest przetwarzanie danych rządowych i komunalnych i ich zabezpieczanie.*
- *Działalność konferencyjną, tj. zorganizowanie Światowej Koferencji Informatyki Medycznej (MEDINFO), Światowej Konferencji Zastosowań Komputerów w Nauczaniu (WCCE) i Konferencji nt. Zastosowań Komputerów w Produkcji i Zagadnieniach Konstrukcyjnych (CAPE).*
- *Nowe wysiłki, podjęte przez Komitet Informacyjny w celu lepszego informowania o pracy IFIP, wydawanie czasopisma »Computer Compacts« oraz wielu książek i innych czasopism.*

Ostatnie sześć lat, oceniając z różnych punktów widzenia, było okresem znacznego rozwoju IFIP:

- *Federacja powiększyła się o dziewięciu pełnoprawnych członków (w tym jeden członek regionalny reprezentujący sześć nowych państw) i trzech członków afiliowanych (obecny pełny skład: 44 członków pełnoprawnych i 6 członków afiliowanych); ten fakt jest szczególnie interesujący, ponieważ członkowie naszej Federacji to kraje różnych części świata, w tym kraje rozwijające się.*
- *Rozwinęły się stosunki IFIP z innymi organizacjami międzynarodowymi, co powinno pozostać tendencją stałą.*
- *Stworzono dziewięć nowych Grup Roboczych, a dwie rozwiązano (obecnie Federacja liczy 9 Komitetów Technicznych z 34 Grupami Roboczymi i jedną Grupą Specjalną).*

Profesor Pierre Bobillier opuścił IFIP po 16 latach pracy, aby — jak twierdzi — ustąpić miejsca młodym.

MK

Trzydziestolecie terminu bajt

Znakomity informatyk amerykański Werner Buchholz — znany jako autor i redaktor dzieł komputerowych, a zwłaszcza opisu PROJECT STRETCH, który zapoczątkował rodzinę IBM-360 — przypomniał przed blisko dekadą, że przeszukując notki archiwalne ze swej dawnej działalności znalazł (pod datą — czerwiec 1956) najstarszy ślad neologizmu *byte*. List W. Buchholza w tej sprawie zamieściło czasopismo BYTE (February, 1977), a następnie przedrukował kwartalnik „Annals of the History of Computing", Vol. 3, No. 1, January 1981.

Właśnie konstruktorzy komputera STRETCH, czyli pierwszej maszyny cyfrowej mającej w zamiarach swych twórców osiągnąć rewelacyjną — na owe czasy — szybkość miliona operacji na sekundę, potrzebowali osobnego terminu na oznaczenie manipulacyjnej porcji kilku bitów bezpośrednio nie związanych z kodem alfabetycznym, wówczas najczęściej 6-bitowym. Tak więc, ów „czerwcowy" bajt miał długość logiczną ustalaną w zależności od potrzeby — od 1 do 6 bitów. Ale już dwa miesiące później, w sierpniu 1956 r. tę nową jednostkę wydłużono do 8 bitów, i ta definicja przeszła do dokumentacji projektowej prototypu STRETCH i jego wersji produkcyjnej IBM-7030. Pod względem technicznym bajt miał o 1 bit więcej (parytet). Po kilku latach okazało się, że praktycznie nie występowały inne bajty logiczne niż dokładnie 8-bitowe, co obecnie można uważać za standard definicyjny, choć jeszcze nie wprowadzony do słowników informatycznych (por. np. słownik Sipple'ów, recenzowany swego czasu na łamach INFORMATYKI).

Zdawałoby się, że do wyjaśnionej w ten sposób prehistorii terminu *byte* — od samego początku używanego przez autorów z modyfikacją ortograficzną słowa *bite* (kęs), które było zbyt podobne do zadomowionego już wówczas terminu *bit* — nie można nic dodać. Można jednak dopowiedzieć pewne wątki ... arabskie.

Otóż arabiście stołecznemu, p. Januszowi Daneckiemu z Uniwersytetu Warszawskiego, zawdzięczam informację, że w czterowersowej formie rubajatu, ulubionej przez poetów muzułmańskiego Wschodu, wyróżnia się dwa paski składowe — zwane właśnie *bajtami*. W językach semickich rdzeń spółgłoskowy *bjt* oznacza namiot, namioty zaś najczęściej budowano jako dwuspadowe, stąd w przenośni dwuwers nazwano *bajt*. Wprawdzie niewiele z tego wynika dla samej informatyki, ale można wątkiem arabskim podbudować samą popularyzację nazwy fonetycznej *bajt*. Kto wie, może słowa 16-bitowe będziemy kiedyś nazywać *rubajatami*?

ABE

Cardware

Sezon na wyroby „czwartej generacji" w krajach przodujących informatycznie już zapewne się skończył, skoro w reklamach pojawił się nowy wyraz *cardware*. Brzmi on fonetycznie zupełnie podobnie do *hardware*, a oznacza wymienną płytkę układową, nazywaną potocznie kartą (ang. card), a raczej zbiór tych płytek. Jest to więc „opłytkowanie", „pakiet standardowy", „sprzęg araptacyjny", czy jeszcze coś innego w tym rodzaju, ale nie dającego się wyrazić prosto i zrozumiale w mowie nadwiślańskiej. Zapewne więc najtęższi eksperci, o ile tylko nowy termin zyska szersze uznanie, będą mówić *kardłer*.

Na razie termin *cardware* — pisany jako nazwa własna, a więc *Cardware* — wprowadziła firma Grandsystems Ltd. z Bristolu na oznaczenie adaptera swej produkcji do minikomputerów VAX i PDP-11 (Computing, 5 December 1985, p. 11). Adapter ten, przy pojemności pamięci do 768 KB, w zależności od liczby dołączonych końcówek, pozwala korzystać z całego oprogramowania IBM-owskiej rodziny mikrokomputerów PC.

Rzecz charakterystyczna, że o ile nazwy rodzin minikomputerów VAX, PDP-11 czy też skrót firmowy ich producenta — DEC — są z reguły używane z wyróżnieniem ich zastrzeżenia, jako zarejestrowanych prawnie nazw handlowych (np. VAX), to nazwa *Cardware*, choć pisana z dużej litery, nie ma już tego oznacznika. Wygląda to tak, jakby ogłoszeniodawcy zależało na wylansowaniu nowego terminu informatycznego.

ABE

Wylicznik

Przeglądając INFORMATYKĘ z lat ubiegłych zauważyłem w dziale terminologicznym numeru 4 z 1983 r. notatkę zatytułowaną „Samolicz".

Podzielam w pełni zdanie Autora notatki nt. zeszpecenia mowy polskiej słowem *komputer*. Widzę również zasadność konstrukcji proponowanego słowa *samolicz*. Nie wierzę jednak, że słowo to zostanie zaakceptowane przez szerokie rzesze informatyków. Dlatego proponuję słowo *wylicznik*. Dokładniej opisuje ono istotę działania maszyny cyfrowej, a jednocześnie jest bliższe dźwiękowo zawodowym informatykom, zwłaszcza konstruktorom urządzeń cyfrowych. Poza tym współbrzmi z innymi dość rozpowszechnionymi w tym środowisku słowami: *przelicznik* (procesor) oraz *sterownik* (jednostka sterująca).

CZYTELNIK

Wichmann B. A., Meijerink J. G. J.: Konwersja oprogramowania na język Ada

INFORMATYKA 1986, nr 4, s. 1

Przykład konwersji podprogramu generowania liczb losowych z języków Fortran i Pascal na język Ada. Omówiono metodę zwiększającą efektywność podprogramów w Adzie.

Faber R., Ostrowski M.: UPT — uniwersalny procesor do redagowania tekstów dla MERY 400

INFORMATYKA 1986, nr 4, s. 5

Charakterystyka pakietu programów do redagowania tekstów z uwzględnieniem znaków polskiego alfabetu na minikomputerze MERA 400. Omówiono konstrukcję, funkcje i tryby pracy oraz przykład zastosowania pakietu.

Janczura A. T.: Rozwiązywanie pasmowych układów równań liniowych na mikrokomputerach

INFORMATYKA 1986, nr 4, s. 9

Omówienie algorytmu oraz programu usprawniającego rozwiązywanie pasmowych układów równań liniowych na prostych mikrokomputerach.

Szkaradnik Z.: Rekurencja w języku Forth

INFORMATYKA 1986, nr 4, s. 10

Charakterystyka różnych metod stosowania procedur rekurencyjnych w języku Forth.

Kleiber M., Leśny M., Szuniewicz R.: Komputery osobiste w zastosowaniach profesjonalnych (2)

INFORMATYKA 1986, nr 4, s. 11

Dokończenie charakterystyki komputerów osobistych z punktu widzenia ich cech użytkowych. Omówiono programy graficznego przedstawiania wyników obliczeń, przechowywania i wyszukiwania informacji oraz pakiety zintegrowane, a także stosowane języki programowania.

Karczmarczuk J.: Jezyk programowania Icon (2)

INFORMATYKA 1986, nr 4, s. 13

Druga część charakterystyki języka Icon, obejmująca omówienie pierwotnych struktur danych oraz generatorów.

Kotulski L.: Monitor jako narzędzie strukturalizacji programów współbieżnych (2)

INFORMATYKA 1986, nr 4, s. 16

Druga część charakterystyki monitora programowego jako narzędzia strukturalizacji programów współbieżnych, zawierająca analizę jego funkcjonowania.

Вихман Б.А., Мейеринк Д.Г.Т.: Конверсия программного обеспечения на язык Ада

INFORMATYKA 1986, № 4, с. 1

Пример конверсии подпрограммы генерирования случайных чисел с языков Фортран и Паскаль на язык Ада. Описание метода повышения эффективности программ на языке Ада.

Фабер Р., Островский М.: УПТ — универсальный процессор для редактирования текстов для мини-ЭВМ МЕРА 400

INFORMATYKA 1986, № 4, с. 5

Характеристика пакета программ для редактирования текстов с учетом знаков польского алфавита на мини-ЭВМ МЕРА 400. Описаны конструкция, функции и режимы работы, а также пример применения пакета.

Янчура А.Т.: Решение полосовых систем линейных уравнений на микро-ЭВМ

INFORMATYKA 1986, № 4, с. 9

Описание алгоритма и программы, облегчающей решениефполосовых систем линкйных уравнений наспростых микро-ЭВМ.

Шкарадник З.: Рекуррентное соотношение на языке форт

INFORMATYKA 1986, № 4, с. 10

Характеристика разных методов применения процедур рекуррентных соотношений на языке Форт.

Клейбер М., Лесьны М., Шуневич Р.: Персональные ЭВМ в профессиональных областях применения (2)

INFORMATYKA 1986, № 4, с. 11

Завершение характеристики персональных ЭВМ с точки зрения их эксплуатационных свойств. Описание программы графического представления результатов расчетов, хранения и поиска информации, а также интегрированных пакетов и применяемых языков программирования.

Карчмарчук Е.: Язык прграммирования Икон (2)

INFORMATYKA 1986, № 4, с. 13

Вторая часть характеристики языка Икон, содержащая описание первичных структур данных и генераторов.

Котульский Л.: Монитор в качестве инструмента структуризьции параллельных программ (2)

INFORMATYKA 1986, № 4, с. 16

Вторая часть характеристики программного монитора в качестве инструмента структуризации параллельных программ, содержащая анализ его функционирования.

Wichmann B. A., Meijerink J. G. J.: Software conversion to Ada programs

INFORMATYKA 1986, No. 4, p. 1

An example of random number generating routine conversion from Fortran and Pascal to Ada programs. A method, which improves effectiveness of Ada routines, is discussed.

Faber R., Ostrowski M.: UPT — an universal text processor for MERA 400

INFORMATYKA 1986, No. 4, p. 5

Characteristics of the program package for text editing with polish letters on MERA 400 minicomputer. Structure, functions and operation modes as well as an example of the package application, are discussed.

Janczura A. T.: Linear equations solving on microcomputers

INFORMATYKA 1986, No. 4, p. 9

Algorithm and program for improving of linear equations solution on simple microcomputers are discussed.

Szkaradnik Z.: Recurrence in Forth

INFORMATYKA 1986, No. 4, p. 10

Characteristics of different methods for recurrent procedures application in Forth language.

Kleiber M., Leśny M., Szuniewicz R.: Personal computer in professional applications (2)

INFORMATYKA 1986, No. 4, p. 11

Termination of personal computer characteristics from application features point of view. Graphic presentation of calculations, information storage and retrieval, and integrated program packages, as well as applied programming languages are discussed.

Karczmarczuk J.: Icon programming language (2)

INFORMATYKA 1986, No. 4, p. 13

Second part of the Icon language characteristics, which includes discussion of original data structures and generators.

Kotulski L.: Monitor as a tool for concurrent programs structuralization (2)

INFORMATYKA 1986, No. 4, p. 16

Second part of characteristics of programming monitor as a tool for concurrent programs structuralization, which includes analysis of the monitor function.

Wichmmana B. A., Meijerink J. G. J.: Softwareumwandlung in Ada

INFORMATYKA 1986, Nr. 4, S. 1

Ein Beispiel von Umwandlung der Fortran und Pascal-Zufallzahlengeneratorroutinen in Ada-Routine. Es wurde eine Methode zur Leistungssteigerung der Ada-Routinen besprochen.

Faber R., Ostrowski M.: UPT — ein universeller Textprozessor für MERA 400

INFORMATYKA 1986, Nr. 4, S. 5

Eine Charakteristik des Programmpakets für Textredigieren mit Berücksichtigung des polnischen Alphabets auf MERA 400-Kleinrechner. Es wurden Konstruktion, Funktion und Arbeitsmodi, sowie ein Beispiel der Paketsanwendung besprochen.

Janczura A. T.: Lösung der Lineargleichungen auf Mikrorechnern

INFORMATYKA 1986, Nr. 4, S. 9

Eine Besprechung von Algorithmus und Programm, die die Lösung der Lineargleichungen auf einfachen Mikrorechnern rationalisiert.

Szkaradnik Z.: Recurenz in Forth

INFORMATYKA 1986, Nr. 4, S. 10

Eine Charakteristik von verschiedenen Methoden für Anwendung der rekurrenten Prozeduren in Forth-Sprache.

Kleiber M., Leśny M., Szuniewicz R.: Personal Computer in professionellen Anwendungen (2)

INFORMATYKA 1986, Nr. 4, S. 11

Beendigung einer Charakteristik von Personal Computern aus Gesichtspunkt ihrer Nutzmerkmale. Es wurden Programme für graphische Darstellug von Berechnungen, für Informationsspeicherung und -Recherche, sowie die integrierten Programmpakete und verwendeten Programmiersprachen vorgestellt.

Karczmarczuk J.: Icon-Programmiersprache (2)

INFORMATYKA 1986, Nr. 4, S. 13

Zweiter Teil einer Charakteristik von Icon-Programmiersprache, die eine Besprechung von ursprünglichen Datenstrukturen und Generatoren umfasst.

Kotulski L.: Monitor als Hilfsmittel zur Strukturalisierung der parallel ablaufenden Programme (2)

INFORMATYKA 1986, Nr. 4, S. 16

Zweiter Teil einer Charakteristik von Programmonitor als Hilfsmittel zur Strukturalisierung der parallel ablaufenden Programme, der eine Analyse der Monitorsfunktionierung umfasst.

Z pamiętnika programującego w Basicu

8 maja 1975

Ten Basic to całkiem niezły pomysł (powiadają, że jest przeznaczony przede wszystkim do początkowej nauki programowania...). Zwłaszcza te numerowane instrukcje — można je dowolnie wymieniać, usuwać, wstawiać. Cóż za wygoda!

A łańcuchy, czyli dane tekstowe — o ileż łatwiej operować nimi w Basicu, aniżeli w Algolu! No i dogodność polegająca na współpracy z interpreterem w fazie opracowywania programu i na możliwości skompilowania gotowego programu, aby mógł pracować szybciej.

I pomyśleć, że programuję w Basicu na maszynie Odra 1204 [1]).

15 października 1978

Okazuje się, że w Basicu można operować podobnymi strukturami sterowania, jak w Pascalu, nie tracąc przyjemności wynikających z pracy konwersacyjnej. W RC-Basicu, zwanym Comalem [2]), pracującym na minikomputerze RC 3600 dostępne są wszystkie instrukcje umożliwiające strukturalizację programu: pętla rodzaju „powtarzaj..., aż...", pętla typu „dopóki..., wykonuj...", instrukcja warunkowa „jeśli..., to..., w przeciwnym razie...", instrukcja wariantowa „wybierz przypadek ... spośród...", nie mówiąc o tradycyjnej pętli „dla sekwencji... wykonaj...". Są i procedury! I współpraca z dyskiem, i operacje na macierzach dwuwymiarowych (może wreszcie polubię algebrę?!), no i ta nieoceniona wygoda wprowadzania danych i wyprowadzania wyników. Posłyszałem opinię, że Comal wypełni lukę w dziedzinie języków edukacyjnych.

Myślę, że to idzie w dobrym kierunku.

13 listopada 1980

— Coś nie słychać, aby rozpowszechniano na maszynach Odra 1305 nowy system programowania — Basic wieloaktywny. A tyle weń włożono pracy!

28 listopada 1984

Przeczytałem bardzo interesującą i pouczającą dyskusję na temat przydatności języków programowania do nauczania początkowego informatyki [3]). W szranki stanęły: APL, Basic, Fortran IV i 77, Pascal i PL/I. Basic strukturalny obronił się w argumentach, ale w klasyfikacji zajął któreś ze środkowych miejsc (oceniono 10 języków, wygrał Pascal).

W dyskusji i jej ocenie wielokrotnie przewijał się pogląd, który od pewnego czasu udziela się i mnie — w nauczaniu początkowym znacznie istotniejsze od doboru pierwszego języka programowania są: fachowość i doświadczenie pierwszego wykładowcy, z którym zetknie się uczeń [4]). Poza tym pozostawiono miejsce dla hobbistów (od pewnego czasu zdradzam — niestety, głównie platonicznie, Basic z Icon [5]).

23 września 1985

Mam w domu centrum obliczeniowe, a pod oczami — sińce. To od zaliczania gier na Spectrum. Ostatnią noc poświęciłem firmowemu Basicowi. I do sińców dołączył ból głowy. Co się stało z tym tak dobrze ewoluującym językiem? Na mikrokomputerach zatrzymał się w rozwoju? Gdzież czasy Comala, strukturalnych instrukcji?! Od przeglądania rodzimych programów w Basicu ZX Spectrum nabawiłem się oczopląsu. Skąd wzięła się maniera tasiemcowego łączenia instrukcji pod jednym numerem? Gdzie możliwość przenumerowania instrukcji programu? I w ogóle, co za styl?!

17 października 1985

Postanowiłem zawalczyć o upadający Basic. Śmiem twierdzić, że to mikro-biznes i ta rynkowa „softtandeta" tak go załatwiły. Nie mogąc się przegryźć przez pewien program, dołączony jako ilustracja programowania w Basicu na Meritum (model roku szkolnego 1984/1985), postanowiłem napisać ten program od nowa: przejrzyście i zrozumiale. Postanowienia dotrzymałem. Ale czy to pomoże Basicowi w jego obecnych warunkach? Pomijam fakt, że ciałem jest rzeski — wystarczy włączyć komputer do kontaktu. Ale duch! Kto mu doda nowego, zdrowego ducha?

Ostatnio dużo doświadczam Logo.

2 listopada 1985

Mam nowego idola! Basic? Do licha z nim — zatrzymał się u nas w rozwoju. Gdzie mu do Logo! Gdzie procedury z parametrami, gdzie modularny charakter, gdzie wszechstronna możliwość wypowiadania się klarownymi nazwami, gdzie nowoczesne struktury danych (listowe, oczywista)? A środki wspomagania programowania? Edytor Logo — to niańka w porównaniu z tymi numerami, od których miga pod powieką! O grafice już nawet nie wspomnę.

3 lutego 1986

Basic na Spectrum jest jednak szybszy niż Sinclair Logo (i ma kompilatory!). Ale w Logo szybciej i składniej się pracuje. Systemy buduje się w nim z procedur jak z klocków.

8 lutego 1986

Jestem w rozterce. Logo naprawdę mnie przekonuje, ale... Wczoraj znowu wykrylem błąd w realizacji Logo Sinclaira. To bardzo utrudnia pracę: człowiek musi odbiec od własnego zadania szukając drogi obejścia przeszkody. O, niezawodności programowania! Gdzieżeś?!!

Lecz do Basica wracać nie będę. Extended Basic? Beta-Basic na Spectrum i jego instrukcje strukturalne? Cóż, nie mają tyle uroku i elegancji co Logo. Choć oznaczniki sposobu traktowania słów w Logo — owe dwukropki, średniki, backslashe (jak to się nazywa po polsku?) — lub ich brak, też mi się czasem mylą, nic to w porównaniu z gmatwaniną skoków do bezdusznych, nic nie mówiących numerów w Basicu. Chcę do Logo! To już postanowione (poza tym — panta rei).

11 lutego 1986, o jedenastej

Ostatnio dużo mówi się o zastąpieniu w Logo informatycznej łaciny, czyli języka angielskiego, polskim słownictwem. Piękny i pożyteczny to cel. A gdyby tak ktoś zastąpił cały interpreter Logo Sinclaira nowym i bezbłędnym? Tylko, czy pisanie rzetelnych translatorów nie najnowszych języków jest jeszcze w cenie?

11 lutego, po południu

Byłoby utratą szansy, gdyby Logo miało pozostać tylko językiem rysowania obrazków dla raczkujących. Będzie tak, jeśli nie doczeka się sprawnego i niezawodnego oprogramowania. Ale dość o tym. Niech przemówi Logo!

W tym miejscu urywa się zapis w pamiętniku starego basicowca. Jakie będą jego dalsze losy? Czy wróci do Basica, czy wytrwa przy Logo? Czas pokaże.

ZDZISŁAW PŁOSKI

---

[1]) S. Borak, T. Czaińska, S. Korczak: System programowania Basic 1204. Instytut Informatyki Uniwersytetu Wrocławskiego, 1976 (1204-M-5 II Wr — MERA ELWRO).

[2]) RC Basic — a structured language (Comal). Programming Guide. A/s Regnecentralen Development Division, September 1977 (RCSL 42-i 0671).

[3]) Abacus, Vol. 1, No. 4, Summer 1984.

[4]) Sąd tego rodzaju sformułował m. in. Anthony Ralston.

[5]) Por. artykuł J. Karczmarczuka w bieżącym numerze (przyp. red.).

Przedsiębiorstwo Zagraniczne AMEPOL

oferuje

PÓŁPRZEWODNIKOWE PAMIĘCI OPERACYJNE

do komputerów:

- **ODRA 1305**

pojemność od 64 k do 2 M

- **R – 32**

pojemność od 256 k do 16 MB

do komputerów:

- **MERA 400**

pojemność od 64 k do 1 M

- **MERA 9150 (SEECHECK)**
- **PDP 11 – SM 4**

Pamięci wykonane są na elementach VLSI produkcji zachodniej.
Na powyższe pamięci udzielamy dwuletniej gwarancji.
Terminy dostawy według życzeń klienta.

PZ AMEPOL oferuje ponadto:

- do komputera **ODRA 1305** skaner oraz multiplekser
- do minikomputera **MERA 400** procesory peryferyjne **PLIX** i **MULTIX** oraz moduł inicjatora

Szczegółowych informacji udziela:
Przedsiębiorstwo Zagraniczne
AMEPOL

Zakład Elektroniki i Aparatury Medycznej
Plac Żelaznej Bramy 1
00-136 Warszawa
tel.: 20-34-75, 20-45-05
teleks: 812539 apol pl

EO/461/K/86