Jacek Mączyński

AUTOMATYCZNE PROGRAMOWANIE W ZAKRESIE RACHUNKU MACIERZOWEGO

Andrzej Pilaszek

MATRALG — O
DOKUMENTACJA SYSTEMU

WARSZAWA 1980

Praca wpłynęła do Redakcji dnia 28 czerwca 1980 r.

Zarejestrowana pod nr 34/1980

N a p r a w a c h r ę k o p i s u

Instytut Podstawowych Problemów Techniki PAN

Nakład 170 egz. Ark.wyd. 3,8. Ark.druk. 6 .
Oddano do drukarni w sierpniu 1980 r.
Nr zamówienia 589/0/80

Warszawska Drukarnia Naukowa, Warszawa,
ul.Śniadeckich 8

Jacek Mączyński
Pracownia Obliczeń Numerycznych

AUTOMATYCZNE PROGRAMOWANIE W ZAKRESIE RACHUNKU MACIERZOWEGO

1. Wstęp

Wobec stale utrzymującego się wysokiego kosztu wytwarzania oprogramowania wzrasta ostatnio zainteresowanie programowaniem automatycznym. Pojawiają się obecnie pakiety programowe dążące do zmniejszenia nakładu pracy człowieka i polegające na wykorzystaniu komputera do deszyfracji języków rzędu wyższego, niż powszechne języki programowania: FORTRAN czy też ALGOL, a nawet PASCAL. Wachlarz zagadnień obejmuje te problemy, które wymagają dużego nakładu pracy przy implementacji komputerowej, a równocześnie są sformalizowane w tak wysokim stopniu, że możliwe jest automatyczne zastąpienie napisów w zwięzłym języku ogólnomatematycznych wzorów przez bardziej szczegółowy ciąg instrukcji komputera wyrażonych już np. w języku algorytmicznym, albo dość często, choć nie w omawianym tu przypadku, w języku wewnętrznym.

Ogólnie automatyzacja programowania dotyczy:

- organizacji współpracy centralnego procesora z pamięcią zewnętrzną /dyski, taśmy/ [2],
- tłumaczenia wzorów np. rachunku macierzowego lub tensorowego [1],
- przekształcenia wyrażeń, np. automatycznego różniczkowania lub półautomatematycznego całkowania [1] , [5],
- tworzenie języków programowania b. wysokiego poziomu jak np. PROLOG [7].

Korzyści z zastosowania automatycznego programowania są wielorakie:

- skrócenie czasu przygotowania programu od momentu kiedy zaistniała jego potrzeba, do zakończenia implementacji,
- obniżenie kosztów wytwarzania programów, poprzez znaczne wyeliminowanie wielokrotnych testowań próbnych, a także zmniejszenie liczby potrzebnych znaków pisarskich w tekście pisanym przez człowieka, tudzież zwiększenie przejrzystości tego tekstu,
- nieraz zmniejszenie kosztów obliczeń wynikowych, dzięki automatycznej optymalizacji programu.

Na cenę, którą się płaci za wzmiankowane korzyści składają się następujące elementy /niektóre alternatywnie/:

- koszt ewentualnego zakupu /łącznie z kosztem starań, rozeznania możliwości nabycia itp. /oraz utrzymania nabytego pakietu programowego,
- koszt wytworzenia translatora, płace, koszt czasu komputera, zarządzanie implementacją, utrzymanie programu,
- koszt przyuczenia użytkownika do posługiwania się translatorem,
- koszt translacji.

Najpowszechniejszą pozycją jest koszt nabycia lub wytworzenia translatora. Koszt ten zależy od przyjętych właściwości potrzebnego translatora, postaci programu wynikowego, postaci programu źródłowego, a przy nabyciu, głównie od sytuacji rynkowej. Występuje tu typowy przetarg /trade-off/, w którym trzeba uwzględnić korzyści z posiadania własnej implementacji, co umożliwia rozbudowę pakietu programowego i może stanowić punkt wyjścia do dalszych prac rozwojowych. Trzeba też uwzględnić pozafinansowe trudności rynkowe i wysoki koszt gotowych opracowań. Poza tym własna implementacja może być wariantowana stosownie do potrzeb i przenoszona na inne komputery, co z kolei może być źródłem obniżki kosztów własnych organizacji wytwarzającej.

Stosowanie rozbudowanej diagnostyki błędów podwyższa koszt translatora, ale wydatnie przyspiesza proces uruchamiania, generowanego w wyniku jego działania, programu użytkowego.

Wielu nienumeryków mogłoby się dziwić, że tak dużą wagę przypisuje się eliminacji błędów. Zapewne najlepiej byłoby po prostu błędów uniknąć, ale statystyka ośrodków obliczeniowych i wytwórców programów stwierdza, że koszt testowania pakietu programowego wynosi zwykle około połowy ich ogólnego kosztu finansowego, przyczym koszt liczony w godzinach pracy ludzkiej i pracy komputera wymaga przyjęcia dodatkowych przeliczników i wtedy staje się jeszcze poważniejszy, choćby wobec opóźnień terminów itp. Przyczyną są tu nie tylko błędy pojawiające się wskutek roztargnienia, ale przede wszystkim najrozmaitsze, nie wyjaśnione do końca, kolizje oznaczeń, czy odmienne interpretacje przez poszczególne grupy współpracujące ogólnych założeń projektowych programu.

Zastąpienie człowieka przez komputer w części pracy implementacyjnej zmniejsza prawdopodobieństwo błędu, choćby dlatego, że napisany przez człowieka, bardziej "matematyczny" program źródłowy jest mu bliższy, zawiera znacznie mniej symboli i elementów powtarzających się, niż program pisany od razu w języku algorytmicznym. Jednym z głównych wymogów jest, aby program źródłowy był przejrzysty. Nie są przejrzyste liczne dawniejsze próby języków, w których trzeba stosować złożone symbole, ulubione przez miłośników FORTRANu godzących się z ograniczeniami karty Hollerithа.

Postulatem zasadniczym jest jak najdalej posunięte podobieństwo tekstu źródłowego do stosowanych powszechnie wzorów matematycznych. Wynika to m.in. stąd, że sprawność ręcznej manipulacji wzorami bywa nieraz ogromna, co świadczy dobrze o celowości tradycyjnie stosowanych zapisów matematycznych. Z tego względu jako pierwsze zagadnienie podejmowane w naszej Pracowni nadawało się automatyczne przekształcenie wyrażeń /translacja/ algebry macierzowej. Świadomie w etapie pierwszym trzeba było przyjąć ograniczenie do macierzy o niewielkich wymiarach, całkowicie mieszczących się w pamięci operacyjnej /16 k słów, por. poniżej/ komputera ODRA - 1204. Macierze takie występują w wielu obliczeniach związanych z podzespołami /lub też elementami/ urządzeń, czy też konstrukcji.

Synteza macierzy "wielkich" jest zbyt odpowiedzialną czynnością, aby można było ją całkowicie ogólnie programować automatycznie. Znane sposoby inicjalizacji macierzy wielkich /por. [3] / są problemowo - zorientowane i dotyczą wąskiej klasy zagadnień jak np. obliczenia metody elementów skończonych. Istotnym warunkiem powodzenia przy automatycznej translacji jest uzyskiwanie nie tylko gotowych programów, ale również modułów programowych, podprogramów /procedur/ dających się wcielać do większych pakietów.

Za najlepszą postać programu wynikowego trzeba uznać więc program w języku algorytmicznym /tu ALGOLu - 1204/, a więc taki, który użytkownik może czytać i interpretować znając język, a także przetwarzać, korzystając ze standardowych translatorów języka algorytmicznego na kod wewnętrzny. Czytelność programu wynikowego pozwala przechowywać go w postaci tabulogramu, a więc nawet w razie uszkodzenia nośnika /taśmy/ łatwo go odtwarzać. Pozwala on też porównywać skuteczność wyboru określonych sposobów zapisu wyrażeń macierzowych, wpływu kolejności działań itp., a więc daje szansę nabycia pewnej biegłości w użytkowaniu.

2. Rola macierzy w mechanice

Motto:

"The origin of species"
Ch. Darwin

Macierze pojawiają się w mechanice z różnych przyczyn

-- z układów równań liniowych,

-- z odwzorowań układów współrzędnych,

-- z procesu składania większych systemów z ich elementów składowych,

-- obecnie najczęściej z zastosowania reprezentacji do operatorów funkcjonalnych.

Z punktu widzenia numerycznego trzeba dokonać rozróżnienia 3 rodzajów występujących w rachunkach macierzy:

- macierze operatory funkcjonalne, tzn. takie, które zawierają operatory /np. różniczkowe/ jako składniki ich elementów,
- macierze zmienne, do typowych należą tu przykładowo często obecnie spotykane Jakobiany odwzorowań i inne tego typu,
- macierze stałe /typ "najniższy" nadający się do bezpośrednich obliczeń/.

Oczywiście wystarczy, aby jeden element macierzy był typu "wyższego", aby cała macierz była tego typu.

W pierwszym przypadku, tzn. jeżeli mamy do czynienia z macierzami operatorami np. liniowymi różniczkowymi, konieczne jest wstępne zanalizowanie możliwości zastosowania algebraizacji poprzez wprowadzenie odpowiedniej bazy w przestrzeni i sprowadzenie zagadnienia do macierzy typu "niższego". Obecnie można w większości przypadków dopatrzeć się pochodzenia macierzy w zagadnieniach teorii konstrukcji w występowaniu na wcześniejszym etapie rozważań odpowiedniego operatora funkcjonalnego.

Nie wchodząc w szczegóły można stwierdzić, że potrzebne dla algebraizacji bazy funkcjonalne otrzymuje się poprzez stosowanie:

- reprezentacji w rozwiązaniach szczególnych,
- reprezentacji na nośnikach lokalnych /MES, Metoda panelowa itp./,
- reprezentacji aproksymacyjnych.

Rozważmy przypadek operatora liniowego różniczkowego L i zagadnienie

$$Lu = q, \qquad u \in \mathcal{L}(\Omega)$$

z danymi warunkami tzn.

$$u|_{\Gamma} = p$$

Algebraizację otrzymuje się wprowadzając bazę funkcyjną ϕ w $\mathcal{L}$ taką, że

$$u = c^T\phi$$

Wtedy

$$Lu = Lc^T\phi = c^TL\phi.$$

a $L\phi$ jest już macierzą zmienną.

Stosowane teraz powszechnie sformułowania wariacyjne posługują się z reguły bazami o nośniku lokalnym i najwięcej rozwiązań uzyskano dla poszukiwania rozwiązań słabych zagadnienia wariacyjnego: $\forall v \in B$, znaleść $u \in B$ takie, że zachodzi

$$a(u,v) = \varphi(v)$$

z koercytywną formą dwuliniową $a(u,v)$: $B \times B \to R$ /B - przestrzeń Banacha/.

W obliczeniach pojawiają się wtedy natychmiast tzw. macierze sztywności, które wyrażają się na ogół całkami typu

$$K = \iiint \bar{B}^T D \bar{B} d\omega,$$

gdzie D jest zwykle macierzą stałą /choć rozważa się zagadnienia przestrzennie niejednorodne, w których może być macierzą zmienną/, a $\bar{B}$ jest macierzą zmienną daną zależnością operatorową

$$\bar{B} = \partial N,$$

gdzie ∂ jest liniowym operatorem różniczkowym, a N jest bazą funkcyjną. Bazę tę formułuje się najczęściej w postaci macierzy bowiem np. w metodzie MES stosuje się reprezentację wielopunktową. Mówimy wtedy, że wprowadzamy wielomiany interpolacyjne oparte na węzłach siatki. Wielomianom tym nadaje się w pracach specjalistycznych nazwę funkcji kształtu, co odzwierciedla fakt, że zależą one od sposobu wyboru węzłów siatki. Rozważaną wielkość fizyczną f przedstawia się wtedy w postaci reprezentacji:

$$f = N\delta$$

δ to wektor /lub multiwektor/ o wartościach liczbowych, N jest macierzą o zmiennych elementach, co warunkuje wykonalność operacji ∂ i operacji całkowania.

Zapis macierzy sztywności związanej z lokalnym nośnikiem /elementem skończonym/ zawiera operacje na macierzach danych w punkcie przestrzeni, toteż stają się one w odpowiedniej głębokiej pętli obliczeniowej macierzami stałymi. Dlatego też możliwe jest tu zautomatyzowanie procesu mnożenia macierzy przez zapis tej operacji w języku wyższego rzędu.

Zauważmy, że w szczególności przy elementach krzywoliniowych trzeba wykonywać całkowanie dla każdego elementu z osobna, bowiem zmienia się tu Jakobian transformacji i wyrażenie macierzowe przybiera postać usprawiedliwiającą stosowanie, przy konstrukcji programu obliczeniowego, języka wyższego rzędu, jako zwiększającą przejrzystość i zwięzłość programu.

W przypadku dużych deformacji, kiedy trzeba stosować transformacje pomiędzy różnymi konfiguracjami pośrednimi zastosowanie skróconego zapisu wydaje się również pożądane, jeżeli programy obliczeniowe mają być łatwo modyfikowane.

Trudnoby tu pokazać wszystkie możliwości zapisu w języku wyższego rzędu bez wykonania programów, można jednak spodziewać się, że korzyści takiego zapisu mogą być poważne poprzez daleko idące zbliżenie tekstu programu do tekstu rozważań teoretycznych.

W braku takiej szczegółowej i wyczerpującej egzemplifikacji ograniczmy się do prostszego przykładu, w którym wyrażenia macierzowe są stosunkowo złożone, lecz mimo to udaje się przy zastosowaniu języka MATRALG uzyskać w sposób wysoce zautomatyzowany potrzebny program obliczeniowy.

3. Translacja wyrażeń - uwagi ogólne

Motto:

"Litterae cum sint paucae
varie tamen collocate
innumerabilia verba efficiunt"
Lactantius, Institutiones 3 /III.w.n.e/

Metodyka translacji języków algorytmicznych ma już swoją długą historię. Wyróżnić można tu kilka zasadniczych metod postępowania:

- metody automatów Mealy'ego, Moore'a itp.
- metody przetwarzania wyrażeń na ONP*) zwaną tak, wobec wprowadzonej przez polskiego logika Łukasiewicza notacji logicznej beznawiasowej, rozpowszechnionej obecnie nawet w niektórych typach minikalkulatorów,
- metoda procedur rekurencyjnych.

Na ogół, z przyczyn wyłuszczonych poniżej nie stosuje się jednej tylko z podanych metod, lecz pewną ich kombinację.

Rozróżnić należy poszczególne poziomy analizy tekstu programu. Najniższy poziom to poziom znaku /litery, cyfry, znaku specjalnego/, tutaj proces rozpoznawania jest prowadzony sprzętowo i jest zasadniczo bezkontekstowy. Oznacza to, że stwierdzenie obecności znaku przez czytnik wejścia komputera nie jest w niczym, lub bardzo mało, uwarunkowane obecnością innych znaków w tekście. Wyższy poziom to poziom symbolu pierwotnego. Ten etap rozpoznania angażuje najczęściej jakąś realizację automatu rozpoznającego, bowiem "Litterae sunt paucae" /Znaki są nieliczne/ i automat działa wtedy najsprawniej. Rozpoznane symbole pierwotne traktowane są następnie w ramach gramatyki formalnej języka akceptowanego przez translator. Symbole poprawne są przyjmowane, niepoprawne wyzwalają sygnalizację błędów.

Na wyższym poziomie badaniu podlegają symbole złożone języka. Znów błędne konstrukty są odrzucane i /najlepiej/ sygnalizowane, a poprawne są źródłem tzw. "akcji" tzn. budowy przekładu. Gotowy przekład udostępniany jest użytkownikowi.

Etapem pośrednim przekładu jest wspomniana translacja na ONP, która następnie pozwala już na wygenerowanie ciągów pojedyńczych operacji działających dwu - lub jedno - argumentowo przyczym zawsze występuje jeszcze jeden argument pomocniczy, który można nazwać akumulatorem operacji.

*) ONP - odwrotna notacja polska

Dopiero ciąg pojedyńczych operacji jest przedmiotem obróbki mającej na celu nadanie mu postaci programu. Ponieważ jednak liczba różnych operacji jest naogół ściśle ograniczona /znów "litterae paucae sunt"/, więc każdą z operacji stosunkowo łatwo jest opracować niezależnie tworząc atomy programu wynikowego.

Dla unaocznienia sensu procesu translacji rozważmy przykładowo, instrukcję zawierającą wyrażenie rzeczywiste

a := b X (c + d); tzn.: b c d + X → a

rozpada się na "atomy":

pom := c + d;	czyli	c d + → pom
a := b X pom;	czyli	b pom X → a

w których pojawia się nowa /tu jedna tylko/ dodatkowa zmienna pom.

Zmienna pom w przypadku wyrażeń liczbowych jest liczbą, zajmuje więc określone słowo maszynowe np. z komórki pamięci m.m. ODRA - 1204. W przypadku macierzy sytuacja się zmienia. Macierz pomocnicza zajmuje pole pamięci dostosowane do wymiarów macierzy, od których pochodzi, co znane jest dopiero w toku obliczenia, trzeba więc ją przechowywać w sposób elastyczny i powodować, aby znikała gdy przestaje być potrzebna /tak jak zmienna pom jest niepotrzebna po wykonaniu drugiego z opisanych przykładowo atomów/. Konieczna jest więc obsługa gospodarki pamięcią niezależna od woli i świadomości użytkownika, tkwiąca w sposób "genetycznie uwarunkowany" w przełożonym programie.

Gospodarkę pamięci można prowadzić na wiele sposobów. Najprostszy z możliwych polega na usuwaniu zbytecznych macierzy pomocniczych, wtedy dopiero, gdy jest to konieczne tzn. wtedy, gdy dostępny bufor pamięci jest już wypełniony. Wchodzą oczywiście w rachubę inne strategie. Usuwanie macierzy pomocniczej za każdym razem tzn. wtedy, gdy staje się ona zbyteczna prowadzi do zbyt częstych reorganizacji. Aktualnie wybrany sposób wymaga markowania nieaktualności macierzy pomocniczej w odpowiednim spisie macierzy pomocniczych. Ze spisu tego korzysta program wynikowy przy reorganizacji pól pamięci zajętych przez macierze pomocnicze.

Byłoby nadmiernym uproszczeniem gdybyśmy sądzili, że na tym kończy się problem translacji. Uzyskany "roztwór" musi zawierać dodatkowo pomocnicze zmienne, operacje obliczeniowe, gospodarkę pamięcią itp. Dołączenie potrzebnych segmentów programu wykonuje się automatycznie i pojawiają się one na gotowym tabulogramie programu wynikowego /por. poniżej/.

Unaocznijmy to na przykładzie. Rozważmy w tym celu algorytm mnożenia n macierzy zespolonych przez siebie. Problem taki występuje w akustyce przy modelowaniu rozchodzenia się sygnału akustycznego przez kanały, lub też przy obliczaniu przepustowości łańcucha filtrów elektrycznych. Najprostszy algorytm w języku macierzowym źródłowym jest przytoczony poniżej /pkt.4/.

Nieprzewidywalne w czasie translacji rozmiary macierzy, dynamiczna rezerwacja w programie powodują, że nie można z góry gwarantować, iż /podobnie jak każdy program z rezerwacją dynamiczną/ program angażujący złożone wyrażenia macierzowe zadziała do końca, bowiem m.in. trudno jest przewidzieć dynamiczne rozmiary obszaru pamięci zajętego przez macierze pomocnicze. Analiza i interpretacja programu wynikowego daje użytkownikom szansę dostrzeżenia przyczyny ew. przekroczenia rozmiarów pola pamięci i w wyniku, dokonania z programem źródłowym manipulacji, zmieniających porządek działań lub dokonujących podziału programu, co prowadzi do zmiany liczby lub wymiarów dodatkowych pomocniczych macierzy. Takiej możliwości analizy nie daje translacja bezpośrednio na język wewnętrzny i dlatego nie była rozważana.

Ideę tworzenia programu źródłowego można w sposób mnemotechnicznie uproszczony zapisać jak następuje. W "roztworze" instrukcji języka algorytmowego /tu np. algolu/ umieszcza się "kryształy" zawierające informacje o potrzebnych operacjach macierzowych w postaci wyrażeń zawierających nazwy macierzy, "operatory działań", nawiasy. Z kolei translator "rozpuszcza kryształy" i zamienia je na "atomy", będące napisami już w języku algorytmicznym.

4. Przykład programu źródłowego i jego postaci wynikowej

W programie przykładowym dostrzegamy opisy algolowskie zmiennych, czytanie liczby macierzy n, opis dla potrzeb języka macierzowego, czytanie macierzy pierwszej w postaci jej części rzeczywistej i zespolonej /AR, AI/ przesyłanie do /BR, BI/ oraz, w cyklu ponownie, czytanie następnej macierzy /od 2-ej począwszy/, obliczenie iloczynu znanymi wzorami, identycznymi /choć z nieprzemiennym mnożeniem/ jak dla liczb rzeczywistych, wreszcie po cyklu, wydruk wyniku. Zauważmy, że przesyłanie informacji z macierzy do macierzy /np. AR := BR itp./ należy czytać w prawo "przepisujemy zawartość AR do BR", co trzeba zapamiętać, nie jest to bowiem podstawienie algolowskie.

Gdy zapamięta się postać ograniczników i operatorów, programy tego typu pisze się szybko, a jak można ocenić z przykładu, są one bardzo przejrzyste. Możnaby zaproponować czytelnikowi, jako ćwiczenie, przepisanie programu przykładowego z MATRALGu do postaci algolowskiej, czy to metodą cykli, czy też procedur. Widać byłoby w jakim stopniu zatraca się wtedy przejrzystość programu. Trzeba bowiem wtedy dokonać ręcznie potrzebnej operacji translacji.

Poniżej podany jest również program wynikowy wygenerowany automatycznie. Mimo pewnego zatarcia przejrzystości znać w nim strukturę rozpuszczonych "kryształów", podział na "atomy" w poszczególnych wierszach wykazu.

Przykład I: program w KATRALG'u . Organizacja współdziałania kryształów z roztworem.

```
begin
integer i,n;
array AR,AI,BR, BI,POM[1:2,1:2];
   read(n);
   format('-1.234567₁₀-123');
   ₙ[dim AR(2,2);BR(2,2);AI(2,2);BI(2,2);POM(2,2)ₙ];
  read(AR,AI);
   ₙ[ AR=BR ₙ] ;
   ₙ[ AI=BI ₙ]  ;
  for i:=2 step 1 until n do
   begin
      read(AR,AI);
      ₙ[ AR× BR + AI×BI = POM ₙ];
      ₙ[ AR×BI  + AI×BR = BI ₙ];
     ₙ[ POM = BR ₙ];
   end for i=1(1)n;
   print(BR,BI);
end mnożenie  n  macierzy zespolonych
; -

ₙ?
```

Przykład II: Złożone wyrażenie macierzowe w MATRALG'u

```
begin
integer array M30,M31,M32[1:4,1:4],SC,SS,SIGK,PSI[1:4],
SFT[1:4,1:8],D,IND[1:8,1:8],FIL[1:8];
real thL,sint,cost;
sint:=sin(thL);
cost:=cos(thL);

m[
dim M30(4,4);M31(4,4);M32(4,4);SC(4);SS(4);SFT(4,8);
D(8,8);IND(8,8);SIGK(4);FIL(8);PSI(4)
m]

m[
(M30×SC+M31×SS-M32×SS)×cost+
(M30×SS-M31×SC+M32×SC)×sint+
(M30×SFT+M31×SFT×D+M32×SFT×IND)×FIL+
M30×SIGK-M32×SFT×IND×FIL+
M32×SIGK×thL +M32×SS
=PSI
m]

;
end
m?
```

Przykład III: Program testowy w MATRALG'u dla testowania procedury OPER

```
begin    comment: TEST OPER;
integer
a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12,a13,a14,
a15,a16,a17,a18,a19,i,k,j,n,p,q                    ;

∫
setinput(0); setoutput(0); print('∫kod');
q:=ininteger; setinput(1);
read(a3,a4,a6,a7,a9,a10);
begin
array  A[1:a3,1:a4],B[1:a6,1:a7],C[1:a9,1:a10];
"[dim  A(a3,a4);B(a6,a7);C(a9,a10)"];
a2:=ref(A[1,1]);
a5:=ref(B[1,1]);
a8:=ref(C[1,1]);
read(A,B);
"[ A + B = C "];

format('-1.4567890"-123,');
setinput(0); setoutput(ininteger - lastintegerdiv4*4);

print('∫C∫');
line(1);
for i:=1 step 1 until a9 do
begin
for k:=1 step 1 until a10 do
print(C[i,k]);
line(1);
end for i;
end tablice;
end OPER test;

"?
```

comment w programie wynikowym pojawi się wartość kodu '8' w wykazie dla kryształu. Wartość ta pozwala na ręczne doperforowanie i zamianę na literę q. Zmienna o tej nazwie jest inicjalizowana osobno przy wywolaniu programu testowego jako całości, toteż nie trzeba wprowadzać zbyt wielu kryształów i organizować ich wybierania.;

Przykład IIIa: Tłumaczenie z MARALG'u na ALGOL-1204 programu z przykładu III.

```
begin
integer a1,a2,a3,a4,a5,a6,a,a8,a9,a10,a11,a12,a13,a14,a15,a16,a17,a18;
procedure xmat20000set;
begin  comment inicjalizacja zmiennych opisujacych pole pamieci roboczej
usak(+18);   snz(ET);   usak(14810);
ET:  pska(a14);   pska(a15);   usak(+17); ssak(1100);
pska(a13);  ssak(2);   rska(a15);
dokj(a13);   odkj(a14);   a16:=a14;
odkj(a16);   a17:=a15;   a18:=a13;   zerk(+a14);
end   xmat20000set;

a12:=0;

begin    comment: TEST OPER;
integer
a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12,a13,a14,
a15,a16,a17,a18,a19,i,k,j,n,p,q                ;

∫

setinput(0); setoutput(0); print('∫kod');
q:=ininteger; setinput(1);
read(a3,a4,a6,a7,a9,a10);
begin
array  A[1:a3,1:a4],B[1:a6,1:a7],C[1:a9,1:a10];
;
a2:=ref(A[1,1]);
a5:=ref(B[1,1]);
a8:=ref(C[1,1]);
read(A,B);

begin integer xmat200001,xmat200002,xmat200003;
xmat20000set;   xmat200001:=3797;   xmat200002:=0;
for xmat200003:=
   q,ref(   A[1,1]),a3,a4,ref(   B[1,1]),a6,a7,   -1,0,0 ,OPER,
2048,   -1,0,0,ref(   C[1,1]),a9,a10,   -2,0,0 ,OPER,
```

```
0 do
begin dokj(xmat200002); usak(xmat200003); mok1(xmat200001); pska(+xmat200002);
if xmat200002=1' then xmat200002:=0   end
end   ;

format('-1.4567890ᵇ-12³,');
setinput(0); setoutput(ininteger - lastintegerdiv4*4);

print('ʃCʃ');
line(1);
for i:=1 step 1 until a9 do
begin
for k:=1 step 1 until a10 do
print(C[1,k]);
line(1);
end for i;
end tablice;
end OPER test;


end
ʃ
```

5. Struktura "kryształu" i "atomu"

Występują tu powtarzające się sekwencje organizacyjne. Wywołane jest to dążeniem do efektywności i bezkolizyjności pracy programu wynikowego i potrzebami jego automatycznego generowania, kosztem zapewne jego długości.

Dla zrozumienia poszczególnych segmentów programu trzeba naświetlić metodę kontaktu z pamięcią operacyjną ALGOLu - 1204 /por. [4] /. Pamięć operacyjną jako Pa0 wyobrażamy sobie jako wielką tablicę jednowskaźnikową. Tradycyjnie wskaźnik takiej tablicy nazywamy "adresem" komórki pamięci.

Istnieją dwa sposoby dostępu do komórek pamięci:
- przez nazwę (ALGOL - 60 i 1204),
- przez adres (ALGOL - 1204).

W m.m. Odra - 1204 rozróżniamy trzy główne obszary pamięci operacyjnej, (rozłączne m.in. dzięki tzw. V-modyfikacji wykonywanej sprzętowo):

- obszar Pa01 [1:1536] tzw. systemu /administratora/ chroniony przed niepowołanym dostępem,
- obszar programów użytkowych Pa02 [1 : 14815] , trzeci, 32 - komórkowy, obszar Pa03 zabezpieczony przed zapisem zawiera tzw. "program stały".

W Pa02 pierwsze 3797 komórek zajmują podprogramy biblioteczne algolu /sin, cos itp./, toteż dla programu użytkowego i dynamicznie przechowywanych tablic pozostaje 11018 komórek pamięci. Można w tym polu pamięci przechować np. program obejmujący ok. 1000 rozkazów maszynowych i zbiór około 5000 dwukomórkowych liczb rzeczywistych. Funkcjonujące w ramach ALGOLu - 1204 procedury specjalne tłumaczy się na pojedyncze rozkazy języka wewnętrznego, ale tylko te które funkcjonują w Pa02 por. [4] . Program wynikowy zawiera instrukcje pochodzące z "roztworu" oraz "atomy" roztworzonych "kryształów" nie w postaci niezależnych instrukcji, lecz w postaci wykazu parametrów.

Informacje określające kod operacji "atomu", adresy i rozmiary tablic są elementami wykazu i służą do lokowania w bezwzględnie adresowanym zbiorze w pamięci operacyjnej, który umownie możemy nazwać "COMMON". Z obszarem "COMMON" kontaktuje się bezparametrowa procedura OPER, która dzięki dopuszczalnemu przez algol typowi całkowitemu /integer/ występuje jako element wykazu. Działanie procedury OPER polega na: wykonywaniu działań macierzowych poprzedzonym gospodarowaniem obszarem pamięci PAM zwanym "pole akumulatorów macierzowych".

W ten sposób następuje wstępne ustawienie parametrów dostępnego pola pamięci na początku każdego kryształu. Poza wywołaniem procedury „xmat20000 set" występuje w krysztale cykl po elementach wykazu inicjalizujący zmienne a1,..., a12 jako parametry poszczególnych atomów kryształu. Ciąg instrukcji ma tu postać:

```
for x3 := <wykaz parametrów> , 0 do
begin
x2 := x2 + 1; comment x2 przebiega 1,2,...,11;
PaO2 [x2 + x1] := 0;
if x2 = 11 then x2 := 0;
end for x3;
```

Celem zwiększenia przejrzystości w miejsce długich nazw „xmat20000" występują tu nazwy krótkie z literą x. Przejrzenie powyższego tekstu i porównanie z tekstem przetłumaczonego kryształu powinno pozwolić na pełne zrozumienie programu wynikowego przez użytkownika. Omawiana powyżej symbolicznie treść rozpuszczonego kryształu nie jest tekstem w ALGOLU bowiem PaO2 nie jest tablicą w tym języku. Wymienną częścią kryształu jest <wykaz> obejmujący parametry lokowane w COMMON i wywołanie procedury całkowitej OPER, również jako elementu wykazu. Parametry, to kolejno: kod operacji, następnie 3 trójki dla 3 macierzy: adres pierwszego leksykograficznie elementu i 2 zakresy, jedenastym elementem jest wartość liczbowa OPER. Procedura OPER korzysta z COMMON i wybiera właściwe postępowanie na podstawie wartości kod, sprawdza zgodność zakresów tablic, wykonuje działanie macierzowe i przekazuje wynik do macierzy pomocniczej, figurującej w wykazie nie jako adres, lecz ujemny numer. Jedenastka elementów wykazu może występować jako pojedyńczy atom lub też jako część wykazu atomów, tu w kolejnych wierszach. Zakończenie wykazu następuje przez wyzerowanie a1 w COMMON, gdzie rezyduje aktualny kod operacji.

W razie wystąpienia przerwania wskutek błędnego wykonania któregoś działania macierzowego można zbadać jego przyczynę, korzystając z opcji algolu 1204 czytania zawartości komórek podanej od ósemkowego numeru 07326 /= 3798 dziesiętnie/ do 07350 /= 3810/. Ułatwia to lokalizację przyczyny przyczyny błędu numerycznego, który może się zawsze przytrafić, nawet bardzo wprawnym użytkownikom programu.

Zmienne a14, a15, a13 przyjmują odpowiednio wartości górnego kresu obszaru wolnego, jego długości, wreszcie dolnego kresu tego obszaru. Korzysta się tu z ustawionych przez administratora "MASON" komórek 17 i 18 /oktalnie 021 i 022/, por. [4] .

Przytoczony w pkt. 4 program wynikowy może zawierać makietę procedury OPER, zapewniającą jego poprawność jako programu ALGOL-u i dodatkowo testującą właściwe przenoszenie zawartości obszaru COMMON /ustawianego 11-u elementami wykazu/ do wnętrza procedury.

Procedura OPER jest wymiennym segmentem programu wynikowego. Dlatego też można ją wprowadzić do tego programu w takiej postaci, która jest aktualnie potrzebna. Można w tym celu bezpośrednio wykorzystać bloki procedur np. z [6], lub skorzystać z opisanej poniżej procedury OPER. Te względy powodują, że warto znać treść programu wynikowego, choćby dlatego, aby móc usunąć części w danym przypadku zbyteczne, a wywołane uniwersalnością. Z drugiej zaś strony manipulacje trzeba podejmować bardzo ostrożnie, bowiem, na wejściu do tekstu przetłumaczonego kryształu, występuje sekwencja, która operuje bezwzględnymi adresami obszaru "COMMON" tzn. zapewnia dostęp do całkowitych zmiennych prostych a1, a2,..., a15 traktowanych jako elementy pseudotablicy:

PaO2 [3798] , ... , PaO2 [3814]

Zmienne o "barbarzyńskich" nazwach xmat200001,..., x mat200003 są deklarowane w obrębie bloku związanego z kryształem i nie powinny kolidować z żadną nazwą wprowadzoną przez użytkownika w programie źródłowym, stąd ich długość i brzmienie zmniejszające prawdopodobieństwo przypadkowej kolizji przez przeoczenie.

Prześledzając tekst tłumaczenia kryształu dostrzegamy kolejno występujące instrukcje procedur specjalnych, które tłumaczą się na pseudo-algol w poniższy sposób.

Procedura "xmat20000set" pracuje w obrębie programu wynikowego, toteż dla uniknięcia ewentualnej kolizji nazw musiała być zakodowana w adresach bezwzględnych, bez odwoływania się do zmiennych a1,...,a18 przez nazwy. Jej treść oddaje ciąg instrukcji w pseudo-algolu:

```
a14  := a15  := if PaO2 [18] = 0 then 14810 else PaO2 [18] ;
a13  := PaO2  17  + 1;
a15  := a15 - PaO2 [17]- 2;
a14  := a14 - 1;
a16  := a14 - 1;
a17  := a15;
a18  := a13;
```

W treści przetłumaczonego kryształu nie ma odwołań przez nazwę do zmiennych a1,..., a15, bowiem zmienne o takich nazwach mogłyby wystąpić w programie źródłowym użytkownika, prowadząc do kolizji.

6. Idea gospodarki pamięcią

Informacja o aktualnym stanie bufora macierzy pomocniczych znajduje się w polu PaOz określonym zawartością komórek a13 i a14. Kolejne zapisy tworzą pary /por. rys. 7.1/:

PaOz [a14] , PaOz [a14-1] , PaOz [a14-2] , PaOz [a14-3] ,...

W parze komórek zapamiętuje się

-- w pierwszej: liczbę kolumn i liczbę wierszy macierzy n x p w postaci liczby 4096*n + p,
-- w drugiej adres pierwszego elementu macierzy pomniejszony o jeden.

Dodatkowo przez implikację uzyskuje się:

-- bezwzględną wartość numeru macierzy pomocniczej,
-- liczbę komórek zajętych przez macierz pomocniczą, która może, ale nie musi być równa 2*n*p.

Pomniejszenie o jeden adresu jest konieczne wobec sposobu podawania wartości adresu elementu macierzy jako adresu starszej komórki spośród pary tworzącej liczbę rzeczywistą.

Długość zajętego pola przez macierz dobiera się w procedurze OPER w ten sposób, że dla operacji odwracania macierzy rezerwuje się nietylko podwójną ilość miejsca potrzebnego dla jednej macierzy, ale dodatkowo jeszcze n komórek dla rejestracji permutacji kolumn macierzy.

Stos par komórek zawierających opisy macierzy pomocniczych biegnie wstecz, co jest zgodne z numerami ujemnymi przypisywanymi tym macierzom. Numery te pozwalają bezpośrednio odnajdywać zapisy dzięki procedurze specjalnej „mokz" zapewniającej modyfikację adresu o podwójną liczbę komórek. Z zapisów stosu korzysta się w dwu sytuacjach:

--- przy wejściu do atomu, gdy macierz pomocnicza jest wynikiem, wtedy zakresy macierzy pomocniczej trzeba wprowadzić do stosu pod numerem pary odpowiadającym numerowi macierzy pomocniczej,
--- przy wejściu do atomu z macierzą pomocniczą jako daną. Wtedy wyzerowuje się pierwszą komórkę pary, sygnalizując w ten sposób, że dana macierz pomocnicza jest nieaktywna i może być usunięta jako zbyteczna.

Jeżeli w sytuacji pierwszej okaże się, że pole pamięci jest niewystarczające dla pomieszczenia macierzy pomocniczej, dokonuje się sprawdzenia, czy istnieją macierze pomocnicze nieaktywne i jeżeli takie istnieją, przesuwa się pola macierzy zajmując zwolnione miejsca. Jeżeli takie przesunięcie nie jest możliwe następuje sygnalizacja błędu przekroczenia zakresu pamięci.

Rekapitulując powyższe uwagi, należy stwierdzić, że

---- trudności z przechowywaniem macierzy pomocniczych z ich "nieprzewidywalnością" w trakcie translacji dają się pokonać w ramach przyjętej struktury translacji, a nawet bez zmiany wytwarzanych przez translator "akcji",

---- doświadczenia uzyskane z metodą gromadzenia macierzy pomocniczych mogą umożliwić dalsze postępy w rozwoju idei automatycznie programowanych obliczeń macierzowych.

7. Organizacja procedury OPER

Procedura Oper zawiera segment organizacyjny i segmenty programowe oznaczone etykietkami. Zmienne takie jak ref A11, ref B11, ref C11, n, p, q można uważać jako parametry tych segmentów traktowane podobnie jak parametry procedur. Parametry te, w zależności od wartości zmiennej „kod", są ustawiane na wejściu do segmentu, na podstawie parametrów adresowanych bezwzględnie przepisanych z "COMMON" do a1,...,a10.
Ponieważ dla części segmentów wartości parametrów są niezmienne, więc są one inicjalizowane wcześniej tzn. przed etykietą START.
Segmenty można podzielić na trzy zasadnicze grupy:

I. jednoargumentowe typu „A do C" z odpowiednim przetwarzaniem macierzy A;

NEG, REAL MATRIX MULT, MATRIXCOPY, MATRIX TRANSPOSE

II. dwuargumentowe typu „op (A, B) do C":

MATRIXADD, MATRIXMULT, MATRIXSUBSTR

III. segmenty "gaussowskie" wymagające dodatkowych pól pamięci jak MATRIXINV lub niszczące swoje argumenty jak MATREQSOLVE, korzystające ze wspólnego segmentu rozpoczynającego się do EQUIL:

MATRIXINV, MATREQSOLVE

Inicjalizacja przed etykietą START dotyczy grupy I i II.
W grupie III stosuje się inicjalizację na wejściu przez etykietę.
W przypadku dzielenia macierzy MATREQSOLVE występuje:
sten wejściowy, w którym parametry w "COMMON" oznaczają:

a2 - adres macierzy prawych stron,
a5 - adres macierzy układu równań,
a8 - adres wyniku.

Na wejściu do obliczenia przepisujemy macierz prawych stron z położenia a2 do położenia a8 i nadajemy wartości parametrom

refA11 := a5, refC11 := a8, peO := a8'(od peO – PERMUTE).

Wtedy macierz prawych stron nie ulega zmianie w swoim uprzednim położeniu, zniszczeniu ulega wyłącznie macierz układu równań figurująca pod a5. Począwszy od etykietki EQUIL

refA11, refB11, refC11 oznaczają kolejno:

-- refA11 adres macierzy A układu równań,
--- refB11 macierz pustą,
-- refC11 najpierw macierz C prawych stron, potem rozwiązanie, podobnie
-- n oznacza rząd macierzy układu równań,
-- m=q liczbę prawych stron.

W przypadku dzielenia macierzy
tzn. zapisu

$$\bar{C}/\bar{A} \text{ do } \bar{C} \text{ (tzn. } \bar{A}X = \bar{C},\ X \text{ do } \bar{C})$$

występują tu dwie macierze $\bar{C}$ i $\bar{A}$ przyczym A ulega w wyniku obliczeń zmianie:

--- macierz $\bar{A}$ jest zniszczona i zawiera wyniki przetwarzania macierzy $\bar{A}$
--- macierz $\bar{C}$ zawiera na wyjściu macierz rozwiązania
W przypadku odwracania macierzy

-- macierz $\bar{A}$ ulega zniszczeniu jak poprzednio
--- macierz $\bar{C}$ zawiera na wejściu macierz jednostkową,
na wyjściu odwrotność macierzy $\bar{A}$.

Aby jednak uzyskać współpracę z translatorem MATRALGu trzeba odpowiednio dobrać adresy refA11, i refC11.
Dlatego też w przypadku dzielenia macierzy należy umieścić prawe strony, które są dane w pierwszej macierzy pod refC11, a macierz układu równań daną w drugiej macierzy pod refA11, postępowanie to wymaga jednokrotnego przepisania macierzy prawych stron do pola identyfikowanego przez refC11.

Natomiast w przypadku odwracania macierzy („MATRIXINV") występuje stan wejściowy, w którym:

a4 - adres macierzy układu równań
a5 - adres macierzy pustej
a8 - adres macierzy odwróconej

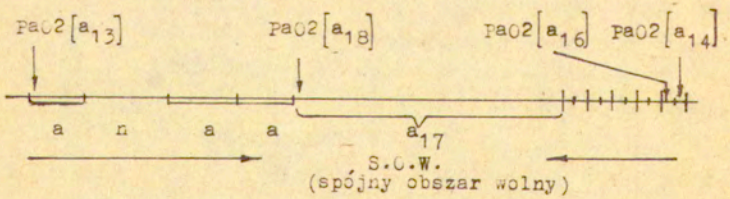
Aby wykonać obliczenie uzupełnia się pod adresem a8 długość macierzy wynikowej o podwójną długość i o długość wektora PERMUTE otrzymując a8, a8', a8''
Aby zminimalizować liczbę przesyłań przepisujemy jeden raz macierz układu równań pod adres a8', macierz jednostkową umieszczamy pod a8, i dokonujemy obliczenia począwszy od etykietki EQUIL tak, że

refA11 = a8 , refC11 = a8', peO: = a8'' ,

wtedy na wyjściu z obliczenia w a8 znajduje się odrazu macierz odwrócona.

PAM

(Pole „akumulatorów macierzowych")



aktywne (a)
i nieaktywne (n)
macierze pomocnicze

zakresy
i
adresy
macierzy
pomocniczych

Rys. 7.1

Literatura cytowana

1. HEARN A.C., Reduce 2, User's manual; Univ. of Utah, March 1973
2. PRYWES N.S., Automatic Generation of Computer Programs; w zbiorze Advances in Computers t. 16; pod red. M. Rubinoffa i M.C. Yovitsa. str. 57 - 125. Acad. Press 1977.
3. SZMELTER J.,i in., Programy metody elementów skończonych; Arkady, W-wa 1973.
4. JERZYKIEWICZ K., SZCZEPKOWICZ J., Algol - 1204. System programowania maszyny cyfrowej ODRA-1204.
5. STOUTEMEYER D.R., Symbolic computation comes of age; SIAM News 2 Dec. 1979.
6. DAN-VAN-VAN, MACZYŃSKI J., Zestaw procedur szybkich operacji macierzowych; w zbiorze: Prace Pracowni Obliczeń Numerycznych w zakresie oprogramowania m.m. ODRA-1204; Prace IPPT z. 68/1972
7. Dunin-Kęplicz B., Szpakowicz S., Język programowania PROLOG; Prace IPI PAN z. 379 (1979).

Tresc procedury O P E R - wersja pierwsza

```
integerprocedureOPER;
begin
integer
a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,
a11,a12,a13,a14,a15,a16,a17,a18
;kod,n,p,q,q2,m,k,j,dB,dC;
realefekt
;
commentautor:prof.drJ.Maczynski;
m:=ref(a1);
fork:=1step1until118do
begin
j:=k-1;
mok1(j);usak(+3798);
mok1(j);pska(+m);
end;
kod:=a1;n:=a3;p:=a4;
begininteger1,izk,ilak,pom,da,pa,pb;
booleanzlyw;
zlyw:=true;OPER:=a1;i:=3809;dokj(+1);
usak(+18);pska(j);usak(+17);pska(k);
setoutput(3);print('k,j',k,j);
ifkgta13thengotoBLAD;

ifa2lt0then
begin
usak(4095);pska(a3);mok2(a2);usak(+a16);
mlka(a3);pnk(+12);pska(a4);
enda2lt0;
ifa5lt0then
begin
usak(4095);pska(a6);mok2(a5);usak(+a16);
mlka(a6);pnk(+12);pska(a7)
enda5lt0;
ifa1=1ora1=4ora1=256ora1=512then
begina9:=a3;a10:=a4;
ENDelse
ifa1=256then
begina9:=a4;a10:=a3;
endelse
ifa1=8ora1=128then
begin
ifa3=a6thena9:=a3elsegotoBLAD;
ifa4=a7thena10:=a4elsegotoBLAD;
endelse
ifa1=16then
begin;
ifa4=a6then
begina9:=a3;a10:=a7;
endelsegotoBLAD;
enda1=16else
ifa1=32then
begin
ifa3=a4thena9:=a10:=a3elsegotoBLAD;
enda1=32else
ifa1=64then
 b
```

```
begin
ifa3=a6anda6=a7then
begina9:=a3;a10:=a4;
endelsegotoBLAD
enda1=64;
usak(+a14);pska(1lak);1zk:=a9*a10*2+2;
ifa1=32then1zk:=1zk+1zk+a3else
ifa1=64then1zk:=1zk+a6;
If1zkle15then
begin
If1zk>a17then
begin
pom:=a13;
fori:=-1step-1until1lakdo
begin
mok2(1);usak(+a16);skz(edp);
mok2(1);usak(+a14);pska(pa);
odkj(1);mok2(1);usak(+a14);
osak(pa);pska(da);dokj(1);odkj(da);
forpb:=0step1until1dado
begin
mok1(pb);usak(+pa);mok1(pb);pska(+pom);
end;
usak(pom);mok2(1);pska(+a14);pom:=pom+da;
edp:;
a18:=pom;
endfori=-1(-1)1lak;
a17:=a15;
end;
odkj(1lak);
usak(a18);mok2(1lak);pska(+a14);
usak(a10);1nk(+12);dlak(a9);
mok2(1lak);pska(+a16);odkj(+a14);
a15:=a15-1zk;a17:=a17-1zk;
1zk:=1zk-2;odkj(1lak);a18:=a18+1zk;
mok2(1lak);pska(+a14);dokj(1lak);
end1zkle15else
begin2lyw:=false;gotoBLAD;
end;
Ifa2lt0then
begin
mok2(a2);zerk(+a16);mok2(a2);usak(+a14);pska(pom);
odkj(a2);mok2(a2);usak(+a14);
osak(pom);sska(a15);a2:=pom;dokj(a2);
enda2lt0;
Ifa5lt0then
begin
mok2(a5);zerk(+a16);mok2(a5);usak(+a14);pska(pom);
odkj(a5);mok2(a5);usak(+a14);
osak(pom);sska(a15);a5:=pom;dokj(a5);
enda5lt0;
mok2(a8);usak(+a14);pska(a8);dokj(a8);
gotoEND;
BLAD:
setoutput(3);print('ff******');
```

```
ifzlywthenprint('zlewymiary')elseprint('brakpamieci');
format('Jwywolanienr3333kod=8888J');
i:=3809;usak(+1);pska(1);
print(1,a1);format('2222255556666J');
print('arg1',a2,a3,a4,'arg2',a5,a6,a7);
stop;
end;
END:;


begin
integerdcech,dn,g1,g2,h0,h1,h2,1,10,j,k,k0,k1,k2,l,m,
m2,mk,n2,nk,pe,pe0,r,r2,rf,refA11,refB11,refC11,s,ster
;dm;
realDETFAC,MAX,REAL,TWOFAC,FACTOR,efekt
;
refA1':=a2;refB11:=a5;refC1':=a8;n:=a3;p:=a4;
n2:=n+n;r:=p;r2:=r+r;
rf:=refA11;OPER:=kod;
format('-1.2345678"-123');
usak(kod);mlak(256);skz(START);
usak(kod);mlak(2048);snz(START);
kod:=kod+2048;
START:;
pe0:=refC11+n*n*2;odkj(pe0);
BPI:;
begin
integeri,j1,j2;
ster:=kod;zerk(j1);zerk(j2);
fori:=1step1until24do
begin
usak(ster);snu(shift);dokj(j1);
shift:lnk(+1);pska(ster);
usak(j2);pad(+1);pzka(j2);
endfori=1(1)24;
ster:=j2;
end;
begininteger i;
i:=25;
E:odkj(1);snd(E2);usak(ster);sku(E1);
lnk(+1);pska(ster);skb(E);
E2:skb(OPEND);commentgdykod=0;
E1:;ster:=24-i;
end;
ifsterlt0thengotoOPEND;
mok1(ster);skb(rozdzielnik);
rozdzielnik:skb(NEG);
skb(MAXEL);
skb(REALMATRIXMULT);
skb(MATRIXADD);
skb(MATRIXMULT);
skb(MATRIXINV);
skb(MATREQSOLVE);
skb(MATRIXSUBSTR);
skb(MATRIXTRANSPOSE);
skb(ZEROMATRIX);
skb(WYDRA);
```

```
skb(MATRIXCOPY);
skb(EQUIL);
skb(TRIANGULA);
skb(SOLVING);
skb(ROWEX);
skb(WYDRA);
ifkod=4thenbeginuzak(+a5);pzka(efekt);end;
NEG:k:=-2;skb(E124);
MAXEL:k:=-3;MAX:=-"100;skb(E125);
REALMATRIXMULT:uzak(+refB1');pzka(REAL);k:=-5;
ifREAL=0thenster:=9else
begin
ifREAL=1.0then
begin
ster:=11;g1:=refC11;
end;
end;
gotoE124;
MATRIXADD:dn:=0;k:=-9;skb(E123);
MATRIXSUBSTR:dn:=-1;k:=-129;skb(E123);
ZEROMATRIX:k:=-513;skb(E124);
WYDRA:line(1);k:=-1025;dn:=7;skb(E125);
MATRIXCOPY:g2:=refB11;k:=-2049;skb(E125);
E123:g1:=refB11;
E124:g2:=refC11;
E125:usak(kod);mlak(k);pska(kod);
h1:=refA11;h2:=h1-2+(n+n)*p;
j:=p;
fori:=h1step2until h2do
begin
u ak(+1);mok1(ster);skb(SOP0);
SOP0:skb(OP0);skb(OP1);skb(OP2);
skb(OP5);skb(ESCAPE);skb(ESCAPE);
skb(ESCAPE);skb(OP5);skb(ESCAPE);
skb(OP3);skb(OP4);skb(OP6);
skb(ESCAPE);
OP0:mina;pzka(+g2);skb(NONE1);
OP1:ozak(MAX);snd(NONE2);
uzak(+1);pzka(MAX);skb(NONE2);
OP2:mzak(REAL);pzka(+g2);skb(NONE1);
OP3:zeaw;pzka(+g2);skb(NONE1);
OP4:pzka(REAL);
ifREAL<0.0thenoutchar(32)elseoutchar(0);
REAL:=abs(REAL);
ifREALgt1.0000005"-9andREALlt9.9999995"9then
format('1.3456789"-1,')elseformat('1.345678"-234,');
print(REAL);odkj(j);odkj(dn);
skz(dnET);pgw(j);skz(jET);skb(NONE2);
jET:j:=r;line(1);
dnET:dn:=7;line(1);skb(NONE2);
OP5:pzka(REAL);uzak(+g1);pgw(dn);skz(ADD);

mina;ADD:szak(REAL);pzka(+g2);skb(NONE);
OP6:pzka(+g2);skb(NONE1);
NONE:g1:=g1+2;
NONE1:g2:=g2+2;
NONE2:;
endfori=h1(2)h2;
```

```
ifster=1then
begin
u ak(MAX);pzka(+refB11);
endifster=1;
Ifkod=0thengotoOPENDelsegotoSTART;
MATRIXMULT:
print(kod,'multƒ');q:=a7;
g1:=refC11;g2:=g1+n2*q-2;
h1:=refA11;h2:=h1+p+p-2;
k2:=refB11;j:=q;q2:=q+q;
fori:=g1step2until g2do
begin
MAX:=0.0;
fork:=h1step2until h2do
begin
u ak(+k2);skz(zwiekszk2);uzak(+k);skz(zwiekszk2);
mzak(+k2);szak(MAX);pzka(MAX);
zwiekszk2:k2:=k2+q2;
endfork=k1(2)k2;
uzak(MAX);pzka(+1);k2:=k2-p*q2+2;odkj(j);snz(iet);
j:=q;k2:=k2-q2;h1:=h1+p+p;h2:=h2+p+p;
k2:=refB11;
iet:;
endfori=g1(2)g2;
i:=-17;usak(kod);mlak(1);pska(kod);
gotoSTART;
MATRIXTRANSPOSE:;
refB11:=a8;ster:=1';
usak(kod);mlak(2048);snz(MATRIXCOPY);
begin
integerrefa,iless,mnless1,done,m,jj,modlessn;
realt;
refa:=refC1'-2;m:=n;n:=p;
mnless1:=m*n-1;modlessn:=mnless1-n;
done:=mnless1-1;k:=0;iless:=1;
fori:=2step1untildonedo
beginj:=k:=ifklemodlessn
thenk+nelsek-modlessn;
test:ifjltiless then
beginjj:=j*(n);
j:=jj-jjdivmnless1*mnless1;gototest;
endifjltiless;
Ifjneilessthen
begin
dokj(j);mok2(1);uzak(+refa);pzka(t);
mok2(j);uzak(+refa);mok2(1);pzka(+refa);
uzak(t);mok2(j);pzka(+refa);
end;
iless:=1;
endfori;
p:=m;
refB1':=a5;
i:=-257;usak(kod);mlak(1);pska(kod);
skz(OPEND);gotoSTART;
end
;
MAT
```

```
MATRIXINV:;
ifnnepthengotoESCAPE;
i:=-65536;usak(kod);mlak(1);skz(inv1);
usak(kod);mlak(65536);skd(inv2);
usak(kod);mlak(131072);skd(inv3);
usak(kod);mlak(262144);skd(inv4);
usak(kod);mlak(524288);skd(inv5);
inv1:;n:=a3;m:=n:=p:=n;refA11:=a2;refC11:=a8+2*n*n;
kod:=kod+67584;
refB11:=a8+2*n*n;ster:=11;;ster:=11;
gotoMATRIXCOPY;
inv2:;
i:=-65537;usak(kod);mlak(1);pska(kod);
kod:=kod+131072;
refB11:=a5;ster:=9;
refC11:=a8;
gotoZEROMATRIX;
inv3:;
refC11:=a8;
i:=-131073;usak(kod);mlak(1);pska(kod);
kod:=kod+262144;k:=a8;
fori:=1step1until1ndo
begin
usak(1.0);pzka(+k);k:=k+2+n+n;
endfori:=1(1)n;
gotoMATRIXINV;
inv4:;
i:=-262145;usak(kod);mlak(1);pska(kod);
refA11:=a8+n*n2;
kod:=kod+524288;pe0:=a8+4*n*n-1;q:=a4;
gotoEQUIL;
inv5:;
i:=-524288-32-1;usak(kod);mlak(1);pska(kod);
gotoOPEND;
MATREQSOLVE:;
i:=65536;usak(kod);mlak(1);skz(eq1);skd(eq2);
EQUIL:;DETFAC:=1;m:=q;m2:=m+m;
g1:=refC11;g2:=g1+m2-2;h1:=refA11;h2:=h1+n2-2;
fori:=1step1until1ndo
begin
dcech:=0;

forj:=h1step2until1h2do
begin
usak(+j);mlak(1023);
osak(dcech);snd(nextj);
ssak(dcech);pska(dcech);
nextj:;
end;
dcech:=dcech-512;
ifdcechne0then
begin
forj:=h1step2until1h2,
g1step2until1g2do
begin
usak(+j);skz(LEQ);osak(dcech);pska(+j);
LEQ:end;
usak(dcech);oska(DETFAC);
end1fdcechne0;
h1:=h2+2;g1:=g2+2;h2:=h2+n2;g2:=g2+m2;
```

```
end;
I:=4097;usak(kod);mlak(1);pska(kod);
pe:=pe0;
TRIANGULA:;
m:=q;
fori:=1step1until1ndo
begin
SEPIVEL:;
h1:=refA11+(1-1)*n2+1+1-2;h2:=h1+2*(n-1);
pe:=pe+1;
MAX:=-"100;dcech:=0;
forj:=1step1until1ndo
begin
k1:=I-1;
fork:=h1step2until1h2do
begin
dokJ(k1);usak(+k);mlak(1023);commentwakum.c+512;
osak(dcech);sku(kcyclend);
ssak(dcech);pska(dcech);
uzak(+k);absa;ozak(MAX);snd(kcyclend);
uzak(+k);absa;pzka(MAX);h0:=k;i0:=j;k0:=k1;
kcyclend:;
endfork=h1(2)h2;
h1:=h1+n2;h2:=h2+n2;
endforj=1(1)n;
IfMAX1e"-80thengotoESCAPE;
DETFAC:=DETFAC*MAX;
if10ne1then
begin
dm:=(10-1)*m2;dn:=(10-1)*n2;

uzak(DETFAC);m1na;pzka(DETFAC);
h1:=refA11+(1-1)*n2+1+1-2;h2:=h1+n2-1-1;
g1:=refC11+m2*(1-1);g2:=g1+m2-2;
fork:=h1step2until1h2,g1step2until1g2do
begin
k2:=k+(ifkge g1andkleg2thendmelsedn);
uzak(+k);pzka(TWOFAC);uzak(+k2);pzka(+k);
uzak(TWOFAC);pzka(+k2);
end;
endROWSWAP1f10ne1;
usak(1);pska(+pe);
ifk0ne1then
begin
uzak(DETFAC);m1na;pzka(DETFAC);
h1:=refA11+1+1-2;h2:=h1+(n-1)*n2;
dn:=(k0-1)*2;usak(k0);pska(+pe);
fork:=h1stepn2until1h2do
begin
k2:=k+dn;uzak(+k);pzka(TWOFAC);
uzak(+k2);pzka(+k);uzak(TWOFAC);pzka(+k2);
endfork=h1(n2)h2;
endCOLUMNSWAP1fk0ne1;
;MAX:=1.0/MAX;
g1:=refC11+(1-1)*m2;g2:=g1+m2-2;mk:=m2;
h1:=refA11+(1-1)*n2;h2:=h1+n2-2;nk:=n2;
```

```
h1:=h1+2*(i-1)+2;
forj:=1+1step1until1ndo
begin
k2:=h1+nk-2;
uzak(+k2);mzak(MAX);pzka(FACTOR);
mina;pzka(+k2);
fork:=h1step2until1h2,
g1step2until1g2do
begin
k2:=k+(ifkgeg1andkleg2thenmkelsenk);
uzak(+k);mzak(FACTOR);rzak(+k2);pzka(+k2);
end;
mk:=mk+m2;nk:=nk+n2;
end;
endfori=1(1)N,TRIANGULA;
Ifkod=8192thengotoOPEND;
s:=refC11-2+m2*(n-1);
SOLVING:;
forl:=1step1until1mdo
begins:=s+2;h1:=n2*n+refA11-2;
h1:=h1+n2;h2:=h1;h1:=h1+2;k2:=s+m2;
forj:=1step1until1ndo
begin
usak(n2);rska(h1);rska(h2);usak(m2);rska(k2);
uzak(+k2);pzka(FACTOR);
fork:=h1step2until1h2do
begin
k2:=k2+m2;
uzak(+k2);mzak(+k);rzak(FACTOR);pzka(FACTOR);
endfork=h1(2)h2;
h1:=h1-2;k2:=k2-m2*(j-1);
uzak(FACTOR);dzak(+h1);pzka(+k2);
end;
endforl=1(1)n,SOLVING;
usak(m);skz(DET);
ROWEX:;
g1:=refC1'-m2;pe:=pe0;
pe:=pe+1;
fori:=1step1until1ndo
begin
usak(+pe);pska(s);
k2:=m2*(s-1)+refC1';
usak(m2);sska(g1);osak(2);ssak(g1);pska(g2);
ifinesthen
fork:=g1step2until1g2do
begin
uzak(+k);pzka(FACTOR);uzak(+k2);pzka(+k);
uzak(FACTOR);pzka(+k2);usak(2);sska(k2);endfork=g1(2)g2;
dokj(pe);
endfori=2(1)n;
gotoDET;
eq1:;n:=a3;q:=p:=a4;
kod:=kod+65536;
refA1':=a2;refB1':=a8;ster:=11;skb(MATRIXCOPY);
eq2:;refA11:=a5;pe0:=a8+2*n*n-1;
refC1':=a8;refB11:=a5;
kod:=kod-65536;skb(EQUIL);
```

```
DET:efekt:=DETFAC;
1:=-97;usak(kod);mlak(1);pska(kod);skz(OPEND);
gotoSTART;
ESCAPE:kod:=1024;gotoSTART;
OPEND:;
end;
endOPER;/
```

Andrzej Pilaszek
Pracownia Obliczeń Numerycznych

MATRALG - 0

Dokumentacja Systemu

I. SYSTEM MATRALG

OGÓLNA CHARAKTERYSTYKA I POJĘCIE PODSTAWOWE

System Matralg służy automatyzacji programowania obliczeń opartych na rachunku macierzowym. Przy jego pomocy można generować programy w wybranych językach programowania wysokiego poziomu.
Aktualnie zaimplementowana jest wersja systemu generująca programy w Algolu 1204.

Zakładamy, że Czytelnicy tej dokumentacji posiadają przynajmniej podstawową wiedzę o programowaniu maszyn cyfrowych. Natomiast praktyczne korzystanie z usług systemu wymaga umiejętności programowania w języku Algol 60.

Schemat działania systemu jest następujący:

- odpowiednio przygotowane wyrażenia rachunku macierzowego umieszczone są w tekście programu, w tych miejscach, w których trzeba obliczać ich wartość.
- wyrażenia te są automatycznie zastępowane przez ciągi instrukcji algolowskich obliczające wartość wyrażenia.
- instrukcje te umieszczone są w programie w miejsce wyrażeń macierzowych.

W ten sposób otrzymujemy program algolowy dokonujący obliczeń rachunku macierzowego.

Do zapisywania wyrażeń macierzowych służy specjalny język formalny o nazwie Matralg. Wstawki w tym języku będziemy nazywać kryształami uważając, że są one zanurzone w "roztworze" tekstu programu w języku docelowym. Dokładny opis języka zostanie podany w następnych ustępach, tutaj wyjaśniamy tylko, że jest on bardzo prosty i jego opanowanie nie powinno stwarzać żadnych trudności oraz, że zastosowanie systemu Matralg pozwala na znaczne zmniejszenie nakładu pracy programisty.
Równocześnie, ponieważ program wynikowy jest generowany automatycznie, więc można w nim zastosować techniki optymalizacji, które przy programowaniu "ręcznym" nie są stosowane ze względu na dużą pracochłonność.

W ten sposób nie tylko ułatwia się proces programowania ale jeszcze można otrzymać programy efektywniejsze od programów tworzonych w całości przez człowieka.

II. Opis języka Matralg.

II.1. Wstęp.

Matralg jest językiem wyrażeń arytmetycznych rachunku macierzowego. W pewnym sensie jest to język programowania. Jednakże, ze względu na specyfikę zastosowania, ma on kilka cech szczególnych. Jego semantyka jest bardzo rozbudowana w stosunku do nieskomplikowanej składni. Równocześnie nie da się określić pełnej semantyki zbioru wyrażeń Matralgu bez odwołania się do składni i semantyki języka programowania, na który wyrażenia te mają być przekładane.

Również wybór zbioru symboli terminalnych zależny jest od języka wynikowego.

W dalszych rozdziałach podany jest opis języka Matralg, używanego w systemie automatycznej generacji programów w języku Algol - 1204.
Przykłady wyrażeń Matralgu znajdują się w cz.III.1.

II.2. Składnia języka Matralg.

Do formalnego opisu składni języka Matralg, użyta została notacja BNF [1] . Pominięty został formalny opis liter i cyfr jako zupełnie naturalny /opis taki znaleźć można np. w [1] /. Przypomnijmy tylko, że w Algolu - 60 i Algolu - 1204 rozróżniane są małe i duże litery.

II.2.1. Zbiór symboli podstawowych.

Teksty wyrażeń Matralgu mogą składać się z następujących symboli:

- małe litery;
- duże litery;
- cyfry;
- wymienione poniżej znaki i symbole pomocnicze /oddzielone od siebie przecinkami/:

$$\underline{m}[,\underline{m}],\underline{m}?,\underline{+},\underline{=},\underline{\times},\underline{/},\underline{=},,;,(,),,,,^{\times},$$

$$\underline{dim},\underline{t},\underline{neg},\underline{inv},\underline{0},\underline{1}$$

Podobnie jak w Algolu, znaki składające się na symbol złożony nie mają odrębnego znaczenia.

II.2.2 Zbiór produkcji gramatyki Matralgu.

operatory jednoargumentowe:

<unioperator>::=$\underline{t}$|$\underline{inv}$|$\underline{neg}$

<+>::=$\underline{+}$|$\underline{=}$

<×>::=$\underline{\times}$|$\underline{/}$

<nazwal> ::= <litera> | <nazwal> <litera> | <nazwal> <cyfra>

<nazwam> ::= <duza litera> | <nazwam> <cyfra> |
<nazwam> <duza litera>

<macierz> ::= <nazwam> | 1 | 0 | (<wyrazenie>)

<czynnik> ::= <macierz> | <czynnik> × <nazwal>

<skladnik> ::= <czynnik> | <skladnik> <> <czynnik>

<wyrazenie> ::= <skladnik> | <unioperator> <skladnik>
<wyrazenie> <+> <skladnik>

<podstawienie> ::= <wyrazenie> = <nazwam>

<liczba> ::= <cyfra> | <liczba> <cyfra>

<zakres> ::= <liczba> | <nazwal>

<deklaracja> ::= <nazwam> (<zakres>) | <nazwam> (<zakres>,
<zakres>)

<lista deklaracji> ::= <deklaracja> | <deklaracja>;
<lista deklaracji>

<krysztal> ::= [<podstawienie>] | [dim <lista deklaracji>]

II.3. Semantyka Matralgu

Rozdział ten podaje podstawowe zasady interpretacji wyrażeń Matralgu. Jako szczegółowe rozwinięcie tego rozdziału można potraktować rozdział III. /podręcznik użytkownika/, /głównie część III.1, III.3/, gdyż zawiera on dokładny opis zależności między postacią kryształów, a otrzymywanymi na ich podstawie programami.

Opiszemy teraz kolejno dwie możliwe postacie kryształów: dyrektywy i wyrażenia.

II.3.1. Deklaracje

Kryształ dyrektyw zawiera opisy macierzy, które będą występowały w wyrażeniach arytmetycznych.

Zakłada się, że wszystkie macierze są typu real, tak więc dyrektywy służą do opisu wymiarów tablic. Dopuszczalne są tablice jednowymiarowe - wektory macierzowe i dwuwymiarowe macierze dwuwskaźnikowe.

Dolny zakres wskaźników jest zawsze równy 1, więc podaje się tylko górny zakres. Tak więc macierz opisaną w Algolu jako array A[1:n,1:3); w Matralgu opiszemy dodatkowo przez ⌊ dim A(n,3) ⌋

Wartość zakresu musi być większa od zera. Kolejność opisów w dyrektywie dim nie ma żadnego znaczenia. Każda tablica musi być opisana przed wystąpieniem w wyrażeniu. Nazwy tablic nie mogą się powtarzać, a więc jedną nazwą nie można opisać wielu tablic.

Jeden kryształ dyrektywy może zawierać wiele opisów, oddzielonych od siebie średnikami np. ⌊ dim A(3); B3(n,5); M1A(zakres,a) ⌋

II.3.2. Wyrażenia.

Wyrażenia arytmetyczne mogą być budowane z identyfikatorów macierzy i zmiennych rzeczywistych programu algolowego, operatorów i nawiasów okrągłych oraz 0 i 1, czyli symboli macierzy zerowej i jednostkowej. Ich wartość jest wyliczana zgodnie z powszechnymi regułami tzn. od lewej do prawej, zgodnie z priorytetami operatorów, z uwzględnieniem kolejności wyznaczonej przez rozmieszczenie nawiasów.

Priorytety operatorów są następujące /od najwyższego, do najniższego, operatory na jednym poziomie mają ten sam priorytet/:

inv t neg

×

± /

± =

=

Znaczenie operatorów jest następujące:

inv - odwracanie macierzy:

t -- transpozycja macierzy;

neg - negacja macierży /wszystkich jej elementów/;

× - mnożenie macierzy przez zmienną rzeczywistą.
Może wystąpić tylko w kolejności Macierz × zmienna;

× - mnożenie macierzy przez macierz;
B A oznacza rozwiązanie układu równań AX = B kolejno dla wektorów prawych stron zapisanych jako kolumny B;

± - dodawanie macierzy;

- - odejmowanie macierzy;

= - podstawianie macierzy /lub wyliczonej wartości wyrażenia macierzowego/ z lewj strony do macierzy znajdującej się po prawej stronie /skopiowanie wszystkich elementów/.

Wymiary wszystkich argumentów muszą być zgodne z zasadami wykonywania operacji rachunku macierzowego. Wymiary macierzy 0 i 1 są ustalane automatycznie w zależności od wymiaru drugiego argumentu operacji. Jeśli na 0 lub 1 wykonywana jest operacja jednoargumentowa lub mnożenie przez zmienną, to musi być jedyna /oprócz kopiowania/ operacja w danym krysztale. Np. niepoprawne w sensie Matralgu wyrażenie [neg1±A=B] można zapisać poprawnie w postaci:

[neg 1 =POM]

[POM±A = B]

Operatory ± - i = nie mogą być używane jako operatory jednoargumentowe. Dwa operatory nie mogą występować pod rząd. Np. wyrażenie:

[A ± negB = C] jest niepoprawne.

Każde wyrażenie musi zawierać dokładnie jeden operator kopiowanie.
Jeden kryształ może zawierać tylko jedno wyrażenie"
Użycie macierzy jako argumentu nie zmienia jej wartości.

III. Podręcznik użytkownika systemu Matralg.

III.1 Postać danych wejściowych.

Dane do systemu Matralg stanowi tekst /ciąg znaków pisarskich/, zakończony symbolem złożonym ?/ /podkreślenie, "dziesiątka algolowska", znak zapytania lub całki/.

Tekst ten może być zupełnie dowolny, w szczególności może w ogóle nie zawierać kryształów. Oczywiście, aby otrzymać w wyniku program algolowski dane powinny stanowić program algolowski ew. z kryształami umieszczonymi we właściwych miejscach /patrz cz.III.2/.

Treść każdego kryształu stanowią znaki zawarte między symbolami początku i końca kryształu, odpowiednio $\underline{[}$,$\underline{]}$. /podkreślenie, dziesiątka algolowska, nawias kwadratowy otwierający lub zamykający/.

UWAGA 1: symbol $\underline{\ }$ na niektórych urządzeniach kodowany jest jako romb ◇.

UWAGA 2: wewnątrz symboli złożonych - /między znakami tworzącymi symbol/ nie może znaleźć się żaden znak za wyjątkiem znaków ML i DL /małych i dużych liter/.
Poza tym niedozwolone są wielokrotne podkreślenia, przekreślenia zamiast podkreśleń. Jeśli w symbolu występują litery to muszą to być małe litery.

UWAGA 3: maksymalna długość kryształu wynosi 2048 znaków.

Graficzny układ treści kryształu nie ma żadnego wpływu na interpretację tej treści, gdyż znaki nowej linii, odstępu, a także znaki niewidoczne w maszynopisie są ignorowane /poza symbolami patrz. UWAGA 2/.

Kryształy zawierające dyrektywy dim mogą być dowolnie przemieszane z kryształami zawierającymi wyrażenia.

Identyfikatory macierzy /nazwam/ mogą być zbudowane tylko z dużych liter i cyfr. Ich długość nie może przekraczać 4 znaków. Gdy identyfikator jest za długi, to zostaje obcięty do pierwszych czterech znaków od lewej.

Zakresy macierzy w deklaracjach mogą być podane liczbami całkowitymi dodatnimi, mniejszymi od 9999, lub zmiennymi. Zmienna określająca zakres macierzy musi być zadeklarowana w programie algolowskim zawierającym kryształy. Wartość zakresu będzie wtedy taka jak wartość zmiennej w czasie przebiegu gotowego programu /nie w czasie translacji dyrektywy dim/. W szczególności wartość ta może być różna dla różnych wyrażeń.

Opis macierzy dyrektywą dim nie zastępuje deklaracji w programie algolowym. Jest ona tylko informacją dla translatora Matralgu, że opisana macierz, o takich samych wartościach zakresów występuje w programie algolowym /"roztworze"/ i może także być używana w wyrażeniach Matralgu.

Tak więc użytkownik musi umieścić w programie deklaracje macierzy i ewentualnie zmiennych określających wartość zakresów tak, aby były dostępne /w sensie składni Algolu/, w tych miejscach, w których macierze są użyte w wyrażeniach arytmetycznych. Dodatkowo, zmienne określające wartość zakresów muszą w czasie liczenia z użyciem macierzy w wyrażeniu mieć taką wartość jak zakresy określone w algolowej deklaracji macierzy.
Np. program otrzymany po przetłumaczeniu danych:

```
begin  integer i;
array A[1:5];
  ¤[ dim A(i) ¤]
E:  i:=6;
¤[  A±A=A ¤]    end   ¤?
```

może być tłumaczony, ale wykona się błędnie. Aby otrzymać działający poprawnie program, instrukcję po etykiecie E należy zastąpić przez i: = 5.

Także zmienne będące argumentami operacji /mnożenie macierzy przez zmienną/ muszą być dostępne z właściwą wartością liczbową w miejscu, w którym wykonywana jest ta operacja.

Spełnienia powyższych warunków nie sprawdza translator Matralgu. Tak więc, gdy nie są one spełnione, to wyprodukowany program będzie błędny. W szczególnych przypadkach mogą pojawić się błędy, które nie zostaną zasygnalizowane, ani podczas translacji programu, ani podczas jego wykonania.

III.1.1. Umieszczanie kryształów w treści procedur tłumaczonych niezależnie.

Wszystkie opisane powyżej reguły przygotowania tekstów stosują się niezależnie od postaci programu algolowskiego, a więc także dla procedur. Jeśli jednak procedura ma być tłumaczona i ew. segmentowana niezależnie, to muszą być spełnione dwa dodatkowe warunki:

- nagłówek procedury musi zawierać parametr formalny OPER, wyspecyfikowany jako procedura. We wszystkich wywołaniach procedury parametrowi temu musi odpowiadać parametr aktualny OPER.

- program główny korzystający z tej procedury musi być przetworzony przez system Matralg, nawet jeśli w jego treści nie występuje żaden kryształ MATRALGU.

III.1.2 Ograniczenia programów algolowskich współpracujących z Matralgiem.

1. Zabronione jest używanie nazwy OPER oraz wszystkich nazw rozpoczynających się od xmat 20 000

2. Zabronione jest używanie /możliwe w algolu - 1204 przy pomocy tzw. procedur specjalnych, por. [4] / obszaru pamięci operacyjnej nie objętej dynamiczną rezerwacją.

III.1.3. Ograniczenia ilościowe tekstów Matralgu.

Długość kryształu, ilość zadeklarowanych tablic, ilość operatorów i nawiasów, liczba kryształów są ograniczone. Przekraczanie dowolnego z tych ograniczeń jest sygnalizowane przez translator jako błąd danych wejściowych.

Jeśli jest to możliwe translacja jest kontynuowana.

III.1.4. Zmiany wymagań pamięciowych programu wynikowego w zależności od postaci wyrażeń.

Obliczanie wartości złożonych wyrażeń arytmetycznych wymaga przechowywania wyników pośrednich. W naszym przypadku wyniki te są akumulatorami macierzowymi, w związku z tym duża ich ilość może spowodować przepełnienie pamięci operacyjnej. Ponieważ wynik użyty w obliczeniach zostaje natychmiast usunięty, więc w razie kłopotów z brakiem pamięci należy przypuszczać, że zbyt wiele akumulatorów macierzowych wystąpiło równocześnie.

Jednym z rozwiązań jest wtedy zastąpienie wyrażenia przez kilka kolejnych wyrażeń o mniej skomplikowanej budowie. Przykładowo macierzowy schemat Hörnera trzeba zastąpić wyrażeniem macierzowym w cyklu. Nieraz możliwe jest takie przeredagowanie wyrażenia, aby obliczenie wymagało mniej wyników pośrednich. Trudno tu podać szczegółowe reguły, więc spróbujemy odwołać się do matematycznej intuicji użytkownika przy pomocy przykładu:

Wyrażenie:

$$\underline{m}[\ (A \underline{\times} B \pm (B \underline{\times} C \pm (C \underline{\times} A \pm D)))\ \underline{m}]$$

wymaga równocześnie trzech wyników pośrednich, ponieważ wyrażenie jest obliczane od lewej do prawej, zgodnie z priorytetami i rozstawieniem nawiasów.

Obliczenie tego samego wyniku przy pomocy wyrażenia:

$$\underline{m}[\ (((C \pm D \underline{\times} A) \pm B \underline{\times} C) \pm A \underline{\times} B)\ \underline{m}]$$

wymaga tylko jednego wyniku pośredniego.

III.2 Postać wyników otrzymywanych z systemu Matralg.

Wyniki produkowane przez system Matralg można w naturalny sposób podzielić na trzy zasadnicze grupy. Pierwsza to produkowane przez pierwszy z programów systemu /mat 1/, opcjonalne wydruki programu źródłowego lub jego fragmentów, oraz informacje o przebiegu translacji /komunikaty o błędach , słownik rozpoznanych nazw/. Druga grupa, to produkowany przez drugą część systemu, program wynikowy. Trzecia grupa, to wydawane przez obie części informacje o pracy systemu, przeznaczone dla operatora emc. Ta grupa komunikatów opisana jest w części VI.

W dwu pierwszych grupach można wybierać wariant wyników /w niektórych przypadkach także urządzenie wyjściowe /przy pomocy kluczy pulpitu technicznego /czytanych automatycznie procedurą key/.

Sterowanie położeniem kluczy interpretowane jest stale następująco: gdy klucz jest wciśnięty /wartość true/to opisana czynność będzie wykonana. w przeciwnym przypadku /klucz zwolniony/ akcja będzie pominięta.

III.2.1 Wyniki otrzymywane podczas translacji.

W czasie translacji można otrzymać:

1/ tabulogram na drukarce wierszowej /DW/ całej taśmy wejściowej - klucz 1.

2/ tabulogram na DW treści kryształów - klucz 5.

Przy produkowaniu tabulogramów obowiązują następujące zasady: przed każdą dużą literą wyprowadzany jest znak "↑", a zamiast pomijanych przez drukarkę znaków "_ " i " | " /podkreślenie i przekreślenie/ wyprowadzane są odpowiednio znaki "!" i "%".

Znak "." /alternatywa/ zastępowany jest przez tekst "!O!R ".

Jeśli podczas translacji znalezione zostały błędy, to informacje o nich wyprowadzane są na DW niezależnie od położenia kluczy.

Program mat1 dzieli się na kilka przebiegów /patrz część IV/. Jeżeli błędy zostaną wykryte już w pierwszym przebiegu, to program kończy działanie po wydruku /opcjonalnym - klucz 1/ treści taśmy i komunikatów o błędach. Wszystkie pozostałe akcje są pominięte, niezależnie od położenia kluczy. Jeśli natomiast wykrycie błędów nastąpi w dalszych przebiegach, to informacje o tych błędach opisuje się i drukuje na DW oddzielnie dla każdego kryształu.

Opis błędów i odpowiadających im komunikatów znajduje się w części III.3.

Przed listą błędów znalezionych w danym krysztale drukowana jest informacja zawierająca numer kryształu i numer przebiegu, po którym przerwano translację kryształu /a więc również zostało przerwane wyszukiwanie błędów/.

Jeśli ilość błędów w krysztale przekracza dopuszczalną wartość /ustaloną przy instalacji systemu/, to drukowane są komunikaty tylko o części błędów i informacja o pominięciu następnych komunikatów.

Jeśli podczas translacji nie znaleziono błędów i klucz 7 jest wciśnięty, to po zakończeniu translacji zostanie wyprowadzony na DW wykaz rozpoznanych nazw macierzy /tablica symboli/. Zawiera on następujące informacje o każdej rozpoznanej macierzy: nazwa, ilość wymiarów, wymiary.

III.2.2 Postać programu wynikowego.

Program wynikowy może być wyprowadzony na perforator - /klucz 11/ i na drukarkę wierszową - /klucz 13/. Klucze 11 i 13 mogą być wciśnięte równocześnie.

UWAGA: jeśli oba klucze /11 i 13/ są zwolnione, to żadne wyniki nie zostaną wyprowadzone.

Przy drukowaniu programu /kl. 13/ nie są wyprowadzane żadne dodatkowe znaki, sygnalizujące obecność znaków pomijanych przez drukarkę.

Postać wyników różni się w zależności od położenia klucza 15. Gdy jest on zwolniony, to otrzymujemy program główny. Do tekstu programu przygotowanego przez użytkownika dołączone zostają deklaracje wszystkich obiektów używanych przez kod wynikowy wyrażeń arytmetycznych. Aby poprawnie umieścić deklaracje cały tekst programu zostaje ujęty w dodatkową parę nawiasów begin end. Po end umieszczony jest znak "?", oznaczający koniec taśmy programu.

Jeśli klucz 15 jest wciśnięty, to zakładamy, że produkowany tekst ma stanowić niezależną procedurę. W związku z tym deklaracje nie są dołączone do tekstu. Jedynym uzupełnieniem jest znak "?", wyprowadzany na zakończenie tekstu wraz z ciągiem 16-u spacji.

Symbol końca taśmy wejściowej m? nie jest dołączany do tekstu wynikowego w żadnym z położeń klucza 15.

Reguły tworzenia tekstu programu są take same w obu położeniach klucza 15. Fragmenty programu źródłowego znajdujące się pomiędzy kryształami kopiowane są bez żadnych zmian.

Kryształy zawierające dyrektywy dim zostają pominięte - nie pojawia się na ich miejscu żaden tekst.

Kryształy zawierające wyrażenia arytmetyczne zostają zastąpione przez instrukcję złożoną zaczynającą się od begin, kończącą się na end. Instrukcja ta pojawia się w miejscu, w którym znajdował się kryształ.
Jej treść stanowi ciąg instrukcji algolowskich obliczających wartość wyrażenia zawartego w krysztale. Podczas wykonania programu wynikowego sprawdzana jest zgodność wymiarów argumentów i wyniku z regułami rachunku macierzowego. Jeśli stwierdzony zostaje błąd, to drukowany jest odpowiedni komunikat i program przerywa działanie.

III.2.3. Wykaz kluczy używanych przez system Matralg.

Klucz	Interpretacja
1	Tabulogram taśmy wejściowej
5	Tabulogram treści kryształów
7	Wydruk tablicy symboli
11	Program wynikowy wydziurkowany na taśmie papierowej
13	Tabulogram programu wynikowego na drukarce
15	Gdy zwolniony, to tekst produkowanego programu uzupełniony jest przez system o niezbędne dodatkowe deklaracje /program główny/

III.3. Informacje o błędach wykrywanych przez translator Matralgu.

III.3.1 Organizacja obsługi błędów.

Podział procesu translacji na przebiegi /patrz cz.IV/ znajduje swe odbicie w organizacji obsługi błędów.

Reakcje translatora na błąd różnią się w zależności od przebiegu. Zawsze jednak stosowana jest zasada "najpełniejszej informacji", tzn., jeśli tylko jest to możliwe, translację kontynuujemy po znalezieniu błędu. W niektórych wypadkach, aby podjąć translację trzeba wykonać pewne dodatkowe akcje, np. pominięcie fragmentu tekstu źródłowego lub odrzucenie deklaracji macierzy. Tego typu akcje zaburzają jednak przebieg translacji, tzn. mogą spowodować niewykrycie niektórych błędów lub wykrywanie pozornych błędów wtórnych. Np. sygnalizowanie jako niezadeklarowanej nazwy macierzy, której opis wystąpił, ale był niepoprawny w programie źródłowym.

Komunikaty o błędach powyższego typu /uniemożliwiających dalszą, w pełni poprawną analizę tekstu/ oznaczone są literą B. Pozostałe błędy /nie mające nielokalnych następstw/ oznaczane są literą I.

III.3.2. Rodzaje błędów i reakcje translatora w poszczególnych przebiegach.

III.3.2.1 Przebieg pierwszy.

Sprawdzane jest tylko, czy symbole początku i końca kryształów są rozmieszczone we właściwej kolejności - czy tworzą poprawną, niezagnieżdżoną strukturę nawiasową.

Znalezienie błędu nie przerywa obróbki taśmy wejściowej, ale jeśli znalezione zostały błędy, to translacja ulega przerwaniu po pierwszym przebiegu.

III.3.2.2 Przebieg drugi

W przebiegu tym ciągi znaków interpretowane są jako symbole języka, przyczym metoda, a więc i wykrywane błędy różnią się w zależności od typu kryształu. Przebieg ten przeprowadza kompletną analizę dyrektyw dim i wykrywa wszystkie występujące w nich błędy.

Natomiast w wyrażeniach arytmetycznych sprawdzana jest tylko poprawność budowy symboli, z których zbudowane jest wyrażenie, oraz czy użyte nazwy macierzy zostały opisane w dyrektywach dim.

III.3.2.3. Przebieg trzeci.

W przebiegu tym sprawdzana jest poprawność rozmieszczenia nawiasów, oraz czy wyrażenie zakończone jest poprawnym podstawieniem.

III.3.2.4. Przebieg czwarty.

W przebiegu tym sprawdzane jest, czy ilość i rozmieszczenie argumentów i operatorów są właściwe. W kategoriach tekstu źródłowego oznacza to eliminację sytuacji, gdy kilka operatorów lub argumentów występuje bezpośrednio po sobie np:

$$A \leq \underline{\text{neg}}\ B\ ,\ A(B \pm C)$$

III.3.2.5 Organizacja obsługi błędów w przebiegach 2.3.4.

Ponieważ każdy kryształ jest tłumaczony niezależnie od pozostałych, więc i obsługa błędów jest oddzielna dla każdego kryształu. Jej organizacja jest następująca: w przebiegach 2 i 3, po znalezieniu błędu translacja jest kontynuowana do końca przebiegu. Następne przebiegi dla tego kryształu są pominięte.

Natomiast znalezienie błędu w przebiegu 4 natychmiast przerywa proces translacji kryształu.

We wszystkich trzech przebiegach informacje o błędach są gromadzone i umieszczone w pamięci zewnętrznej.
Po zakończeniu translacji wszystkich kryształów na podstawie tych informacji drukowane są komunikaty o znalezionych błędach. Postać wydruków jest następująca:
wszystkie komunikaty o błędach w jednym krysztale stanowią oddzielną grupę, poprzedzoną informacją o numerze kryształu i przebiegu, po którym zakończono translację. Grupy te są uszeregowane w kolejności występowania kryształów w tekście źródłowym.

Jeśli ilość błędów znalezionych w jednym krysztale przekracza maksymalną dopuszczalną przez system wartość, to drukowane są komunikaty o początkowej /w kolejności wykrywania/ części błędów oraz informacja o pominięciu reszty komunikatów.

III.3.3. Błąd 0.

W trakcie pracy systemu programowego Matralg przeprowadzane są wewnętrzne testy poprawności działania systemu. Jeśli wykryją one nieprawidłowości to sygnalizowany jest błąd 0.
Odnalezieniem i usunięciem przyczyny błędu musi się zająć konserwator systemu programowego.

III.3.4. Przepełnienie obszaru roboczego.

Przepełnienie którejkolwiek z tablic pomocniczych translatora jest na ogół sygnalizowane jako błąd tego przebiegu, w którym nastąpiło przepełnienie i nie przerywa procesu translacji. Jedynym wyjątkiem jest przepełnienie stosu podczas przebiegu 3 lub 4, które powoduje natychmiastowe zakończenie translacji kryształu.

III.3.5. Wykaz błędów wykrywanych przez translator Matralgu.

Podamy wykaz komunikatów o błędach, uporządkowany w/g rosnącej kolejności numerów błędów. Komunikaty te na ogół dostatecznie wyjaśniają istotę błędów. Jeśli jest to potrzebne, to zamieszczony jest dodatkowy komentarz.

B1 - za dużo kryształów <n>

<n> - liczba całkowita określająca dozwoloną ilość kryształów.

B2 ₁₀] nieoczekiwany <n>

<n> - numer kolejny ostatniego rozpoznanego kryształu.

B3 * za długi kryształ <n>

<n> jak w pkt. 2.

B4 koniec programu wewnątrz kryształu <n>

<n> jak w pkt. 2

B5 ₁₀[nieoczekiwany <n>

<n> jak w pkt. 2.

B6 zły ogranicznik <n>

<n> jak w punkcie 2.

Błąd ten jest sygnalizowany, gdy po rozpoznaniu początku symbolu początku lub końca kryształu, lub końca taśmy, czyli ₁₀, nie występuje poprawny znak kończący symbol, czyli " [" lub "]" lub "?".

B7 Niezadeklarowana nazwa <nazwa>.

B8 zły symbol złożony <sym> .

Wewnątrz kryształu znaleziono symbol złożony, z podkreślonych znaków, który nie jest poprawnym symbolem Matralgu. <sym> zawiera znaki składające się na ten symbol.

B9 nazwa oczekiwana <char>

W miejscu, w którym może wystąpić tylko nazwa macierzy, napotykano znak <char>, nie będący dużą literą.

B10 za dużo nazw <nazwa>

Macierz o identyfikatorze <nazwa> nie może zostać umieszczona w tablicy symboli, gdyż tablica ta jest już pełna. Deklaracja odrzucona.

I 11 podwójna deklaracja <nazwa>
Identyfikator <nazwa> użyty w dyrektywach dim więcej niż raz.
Wszystkie wystąpienia od drugiego począwszy są ignorowane.

B 12 niedozwolony znak <char> ,
Znak <char> nie może wystąpić w miejscu, w którym go napotkano.

B 13 zła wartość zakresu < nazwa>

I 14 za długa nazwa <nazwa>
Nazwa macierzy dłuższa niż 4 znaki.
Deklaracja przyjęta z nazwą obciętą do czterech pierwszych znaków.

B 20 brak (

B 21 brak)

B 22 złe podstawienie
Po prawej stronie operatora = musi wystąpić identyfikator macierzy.

B 23 brak podstawienia

B 31 - argument
Wyrażenie niepoprawnie zbudowane - złe rozmieszczenie operatorów.

B 31 + argument
Jak wyżej.

B 40 przepełnienie stosu.
Wyrażenie zawiera zbyt wiele zagnieżdżonych nawiasów.
Błąd ten przerywa natychmiast translację kryształu.

IV Algorytm translacji języka Matralg.

IV.1 Krótka charakterystyka języka.
Matralg i założeń procesu translacji.

Matralg jest językiem wyrażeń arytmetycznych rachunku macierzowego, przeznaczonych do umieszczania w treści programów algolowych. Dla wyróżnienia wyrażeń z treści programu wprowadzono symbole początku i końca wyrażenia. Wyrażenie wraz z symbolami początku i końca nazywane jest kryształem. Zadaniem procesu translacji jest zastąpienie wyrażenia macierzowego fragmentem programu algolowego. Pragment ten ma obliczać wartość wyrażenia.

Aby umożliwić przekazanie translatorowi potrzebnych informacji o macierzach biorących udział w obliczeniach, uzupełniono język Matralg o kryształy zawierające dodatkowe opisy macierzy /zwane też kryształami dyrektyw typu dim/.

Kryształy te dostarczają tylko informacji dla procesu translacji, nie pozostawiają natomiast żadnego przekładu wynikowego.

Bliższe informacje na temat języka Matralg oraz postaci tekstów wynikowych znajduje się w częściach:
II, III.1, III.2.

W treści opisu występują odwołania do schematów blokowych zawartych w Dodatku A. Są one postaci SB-X, gdzie X jest cyfrą rzymską oznaczającą numer schematu.

IV.2 Ogólna organizacja procesu translacji.

Translacja podzielona jest na dwa zasadnicze etapy.
Pierwszy z nich /SB-I/ składa się z następujących przebiegów:

P1: wczytanie i umieszczenie w pamięci masowej tekstu wejściowego. Zapisanie adresów początku i końca każdego kryształu /SB-II/.

P2: rozpoznanie treści każdego z kryształów i przekształcenie jej z postaci tekstowej na postać pośrednią, dogodną do przetwarzania przez komputer /SB-III/.

P3: tłumaczenie wyrażeń arytmetycznych na odwrotną notację polską /ONP/ /SB-IV/.

P4: tłumaczenie ONP wyrażeń na czwórki liczb, służące bezpośrednio do generacji kodu wynikowego /SB-VII/.

P5: przekazanie wyników do drugiego etapu, ewentualna obsługa znalezionych błędów.

W czasie pierwszego etapu wykonywane są wszystkie sprawdzenia poprawności tekstu wejściowego. Wyniki translacji umieszczane są w zewnętrznej pamięci masowej /PaZ/.

Jeśli dane wejściowe są poprawne, to po zakończeniu pierwszego etapu, w PaŻ znajdują się wszystkie informacje niezbędne do wygenerowania programu wynikowego.

Etap drugi /SB-IX/, to pobieranie z PaZ kolejnych czwórek liczb opisujących operacje i generowanie na ich podstawie /w oparciu o tablice pomocnicze, będące także wynikami pierwszego etapu/ programu wynikowego.

Oczywiście zarówno pierwszy, jak i drugi etap komunikują się z PaZ przy pomocy odpowiednich buforów.

Opiszemy teraz dokładnie działanie poszczególnych przebiegów etapu 1.

P1 /SB-II/.

Przebieg ten wczytuje z taśmy tekst programu źródłowego i umieszcza go w PaZ. Znaki kodowane są jako liczby całkowite i umieszczane po jednym w słowie maszynowym. Podczas wczytywania przeprowadzana jest wstępna analiza tekstu. Rozpoznawane są symbole początku i końca kryształów. Adresy /numery słów w PaZ/ początków i końców umieszczane są w tablicy LK. W ten sposób tablica LK tworzy mapę pozwalającą odszukać treść dowolnego kryształu. Równocześnie sprawdzana jest poprawność rozmieszczenia symboli początków i końców kryształów. Symbole te muszą tworzyć niezagnieżdżone struktury nawiasowe.

Wczytywanie kończy się po rozpoznaniu symbolu końca tekstu.

Jeśli podczas wczytywania nie znaleziono błędów, to przechodzimy do następnego przebiegu.

P2 /SB-III/.

Przebieg ten to tzw. analizator leksykalny /ang. scanner/. Jego zadaniem jest wstępna obróbka tekstu źródłowego, polegająca na rozpoznawaniu ciągów znaków tworzących poszczególne symbole języka i zakodowaniu otrzymanych symboli. W naszym przypadku kodami symboli są liczby całkowite.

Kolejność operacji jest następująca: sprowadzamy do pamięci operacyjnej /PaO/ tekst kryształu z PaZ.

Sprawdzamy, czy zawiera on dyrektywy czy wyrażenie arytmetyczne /te dwa typy wyrażeń są przetwarzane w różny sposób, niezależnie od siebie/.

Jeśli jest on wyrażeniem /SB-V/. arytmetycznym, to w zależności od pierwszego znaku w jeszcze nierozpoznanym fragmencie tekstu, próbujemy rozpoznać nazwę /macierzy lub zmiennej algolowej/ lub ogranicznik.

Wszystkie poprawne ograniczniki umieszczone są w słowniku. Są one zakodowane w postaci tzw. "masek różnicowych", co pozwala na bardzo szybkie przeglądanie słownika.

Ponieważ wynikowy ciąg liczb jest krótszy od tekstu źródłowego, więc wyniki umieszczamy w tym samym buforze co tekst źródłowy, na już przetworzonym fragmencie. Jeśli w trakcie przetwarzania znalezione zostały błędy, to przeglądamy dalszy ciąg tekstu, jednak bez zapisywania kodów wynikowych. Bufora w PaO używamy do zapisania informacji o błędach. Taka skomplikowana gospodarka buforem została wprowadzona, aby jak najoszczędniej wykorzystać niewielką pamięć operacyjną emc. Odra - 1204.

Po przejrzeniu całego tekstu zawartość bufora /stanowią ją kody symboli lub kody błędów/ jest wysyłana do PaZ, na to samo miejsce, w którym znajdował się tekst kryształu.

Drugą możliwą zawartość kryształu stanowi lista opisów tablic. Lista taka służy do budowy tak zwanej "tablicy symboli" /TS/. Znajduje się w niej /w TS/: nazwa macierzy /conajwyżej 4 znaki upakowane w jednej komórce pamięci/ ilość wymiarów; wartość zakresów.

Działanie modułu rozpoznającego deklarację /SB-IV/ jest następujące: rozpoznajemy nazwę i upakowujemy ją w komórce, sprawdzamy, czy dopuszczalna długość TS nie jest już osiągnięta, oraz czy taka sama nazwa nie jest już umieszczona w TS. Jeśli wszystko jest w porządku, to dodajemy nazwę do TS. Następnie sprawdzamy ilość wymiarów oraz wczytujemy wartości zakresów. Jeśli są one poprawne, to umieszczamy je w TS. Wartości zakresów są wczytywane jako ciąg znaków i przed zapisaniem poddawane konwersji na liczby całkowite.

Jeśli w którymkolwiek z momentów opisanego procesu wykryty zostaje błąd, to analiza tego opisu zostaje przerwana, a nazwa macierzy nie zostaje umieszczona w TS.

Po zakończeniu przetwarzania jednego opisu /poprawnego lub nie/ powtarzamy proces dla następnego, aż do wyczerpania listy opisów.

Kody znalezionych błędów są umieszczane w buforze, na miejscu przetworzonego już tekstu źródłowego.

Jeśli podczas przetwarzania znaleziono błędy, to ich kody, zawarte w buforze, wysyłane są do PaZ po zakończeniu analizy kryształu. W przeciwnym razie zawartość tablicy, używanej jako tablica pomocnicza w dalszym procesie translacji jest jedynym śladem przekładu kryształu typu dim.

P3-/SB-VI/, P4-/SB-VII/.

Dwa powyższe przebiegi warto omówić wspólnie, gdyż współpracują one ze sobą bardzo ściśle.

Zajmują się one translacją wyrażeń arytmetycznych do postaci umożliwiającej generację kodu realizującego poszczególne operacje.

Zastosowany tutaj algorytm translacji opisany jest np. w[2],[3] .

Ogólnie mówiąc, proces ten polega na przekształceniu wyrażenia arytmetycznego do tak zwanej odwrotnej notacji polskiej /ONP/. Jest to sposób zapisywania wyrażeń arytmetycznych, w którym najpierw podaje się argumenty, a po nich operator. W procesie translacji wykorzystana jest specjalna organizacja pamięci zwana stosem. Następnie wyrażenie zapisane w ONP poddawane jest jeszcze jednej transformacji, także przy pomocy stosu. Po tej transformacji otrzymujemy ciąg czwórek postaci /argument 1, argument 2, wynik, operator/.

Czwórki te zapisane są w takiej kolejności w jakiej należy wykonywać operacje w programie wynikowym. Tak więc czwórki te są wystarczającą podstawą do generacji programu wynikowego.

Organizacja przebiegów 3 i 4 jest następująca:

- przetwarzamy tylko poprawne wyrażenia arytmetyczne. Sprowadzamy z PaZ do PaO kod wyrażenia uzyskany z drugiego przebiegu. Przekształcamy zapis wyrażenia na zapis w ONP /przebieg P3/. Zapis ONP pozostaje w pamięci operacyjnej. Jeśli podczas translacji nie znaleziono błędów, to przechodzimy od razu do następnego przebiegu /P4/, który przekształca ONP na czwórki. Czwórki te za pośrednictwem dodatkowego bufora wysyłane są do PaZ, gdzie tworzą zbiór informacji przekazywany do etapu generującego program wynikowy. Napotkanie błędu w ONP wyrażenia powoduje przerwanie generacji czwórek. Jeśli błąd wykryto już wcześniej /podczas przekładu na ONP/, to generacja czwórek nie zostaje w ogóle rozpoczęta.

Wygenerowanie czwórek opisujących operacje kończy w zasadzie pierwszy etap translacji. Pozostaje jeszcze do zrobienia pewne "podsumowanie" wyników P5.

W tym przebiegu sprawdzamy czy podczas translacji zostały wykryte błędy w tekście źródłowym. Jeśli tak, to sprowadzamy z PaZ informacje o wykrytych błędach /ustawione tam przez poprzednie przebiegi/ i drukujemy na ich podstawie odpowiednie komunikaty.

Do drugiego przebiegu przekazywana jest tylko informacja, że żaden poprawny kod do dalszego przetwarzania nie został przygotowany.

Jeśli natomiast translacja przebiegała prawidłowo, to wysyłamy do PaZ zawartość tablic pomocniczych dla drugiego etapu translacji, oraz informacje o położeniu w PaZ wyników pierwszego przebiegu /tablic pomocniczych, fragmentów programu źródłowego przeznaczonych do skopiowania, zbioru czwórek/.

Działanie drugiego etapu.

Drugi etap translacji wykonywany jest tylko wtedy, gdy podczas pierwszego etapu nie wykryto błędów w tekście źródłowym.

Funkcją tego etapu jest generacja programu wynikowego. Wykonuje się ją następująco:

fragmenty programu źródłowego zawarte między kryształami dołączane są bez żadnych zmian, w kolejności w jakiej występują w programie źródłowym.

Teksty kryształów nie pojawiają się w ogóle w programie wynikowym. W przypadku kryształów typu dyrektyw dim cały kryształ jest po prostu pomijany. Natomiast w miejscu, w którym występowało wyrażenie arytmetyczne wstawiany jest fragment programu algolowego, wyliczający wartość tego wyrażenia. Fragment ten generowany jest następująco: kolejne czwórki, otrzymane w wyniku przekładu wyrażenia, są sprowadzane z PaZ. Na podstawie każdej z tych czwórek generujemy instrukcje realizujące operację opisaną tu czwórką. Generacja polega na uzupełnieniu ustalonego schematu wartościami aktualnymi pobieranymi z tablicy symboli i z czwórki opisującej operację.

W ten sposób otrzymujemy program wynikowy składający się na przemian z fragmentów tekstu źródłowego i z fragmentów wygenerowanych automatycznie. Program ten jest ostatecznym wynikiem procesu translacji.

V. Dokumentacja programu.

V.1. Wstęp.

Rozdział poniższy zawiera informacje o programach, z których składa się translator języka Matralg. Są to informacje przeznaczone dla programisty zajmującego się konserwacją translatora. Powinny one ułatwić wprowadzenie do programu poprawek lub modyfikacji. Podane informacje nie zawierają wielu drobnych szczegółów. Szczegóły te są łatwe do odczytania z wydruku treści programu, gdyż prawie cały program jest pisany w Algolu, bez stosowania "chwytów" związanych z mniej lub bardziej ukrytymi własnościami maszyny matematycznej. Poza tym treść programu zawiera liczne komentarze.

Zakładamy, że Czytelnik zna treść części I,II,IV oraz dodatku A. W wielu miejscach opisu korzystamy z informacji zawartych w wymienionych pozycjach, bez powoływania się na źródło.

System Matralg składa się z dwóch programów: mat1 i mat2. Realizują one odpowiednio pierwszy i drugi etap algorytmu.

V.2. Opis programu mat 1.

V.2.1 Tablice i zmienne programu.

Zmienne typu integer.

db - długość bufora PaZ.

afr - początek obszaru roboczego w PaZ.

nafr - adres, od którego można aktualnie umieszczać informację w obszarze roboczym PaZ.

nrk - liczba rozpoznanych kryształów.

maxk - maksymalna dopuszczalna ilość kryształów

lb - liczba znalezionych błędów.

maxb - maksymalna ilość błędów rejestrowanych przez translator. Musi być:

$maxb \leq db/4 - 4$

ils - liczba rozpoznanych i zarejestrowanych nazw macierzy.

maxs - maksymalna dopuszczalna ilość nazw macierzy.

ds - maksymalna długość stosu.

dbp - długość bufora PaŻ, używanego przy transmisji "czwórek".

UWAGA: musi być $dbp = 4 \cdot n$, $n \in N$

np - numer przebiegu.

wyp - wskaźnik wypełnienia bufora podczas wczytywania taśmy.

k - ostatnio wczytany znak. Po drugim przebiegu - robocza.

lo - wskaźnik wypełnienia bufora kodem wynikowym.

li - wskaźnik początku nieprzetworzonej zawartości bufora.

kk - koniec tekstu w buforze.

ak - numer obrabianego kryształu.

top - wskaźnik wypełnienia stosu.

ilop - ilość rozpoznanych w wyrażeniu operatorów.

maxa - ilość akumulatorów potrzebna do wyliczenia wartości wyrażenia.

lbp - wkaźnik wypełnienia bufora transmisji czwórek.

Zmienne typu Boolean.

p - true, gdy ostatnim znakiem było podkreślenie.

pd - true, gdy ostatnio rozpoznano symbol: podkreślenie, (_) .

pk - true, gdy wczytujemy treść kryształu.

fe - wskaźnik błędu, true, gdy znaleziono błąd.

feg - wskaźnik poprawności danych źródłowych, true oznacza dane niepoprawne.

nextd - steruje zakończeniem rozpoznawania listy deklaracji.

Tablice /wszystkie typu integer/

SOS [1:11] - zawiera informacje o lokacji w PaZ wyników pierwszego etapu translacji.

LK [0:maxk, 1:2] - pierwsza kolumna zawiera adresy /w PaZ/ początków kryształów, druga adresy końców.
LK [0,2] zawiera adres początku programu źródłowego.

NB [1 : maxb, 1 :2] - zawiera pary liczb opisujące błędy znalezione podczas translacji.

BUF [1:db] - bufor komunikacji z pamięcią bębnową.

TS [1:maxs, 1:4] - tablica symboli, jeden wiersz opisuje jedną tablicę. Kolumny od 1 do 4 zawierają odpowiednio upakowaną nazwę, ilość wymiarów, zakres pierwszy, zakres drugi /dla tablic jednowymiarowych zero/.

STOS [1:ds] - służy do organizacji stosu używanego podczas translacji.

LPC [1:nrk, 1:2] - pierwsza kolumna zawiera adresy początków przekładów poszczególnych kryształów, na czwórki. Druga długości przekładów. Wiersze odpowiadające kryształom deklaracji są niewykorzystane.

BP [1:dbp] - bufor używany przy wysyłaniu do PaZ ciągu czwórek.

V.2.2 Procedury

Użyte oznaczenia:

/i/ - procedura funkcyjna typu integer.

/b/ - procedura funkcyjna typu Boolean.

Procedury bez oznaczeń - niefunkcyjne.

output /l,pocz,kon/;

Wyprowadza tekst zapisany w pamięci bębnowej, od adresu pocz do kon, jeden znak w słowie, na urządzenie wyjściowe nr l.

ERROR /n,m/

Ustawia parę /n,m/ w pierwszym wolnym wierszu tablicy NB.

Ustawia wskaźniki błędu.

ZNAK; /i/.

Zwraca liczbę będącą kodem pierwszego znaku w nieprzetworzonej części tablicy BUF, przy czym znaki niewidoczne w maszynopisie są pomijane. Jeśli cały tekst jest przetworzony, to zwracane jest zero.
Procedura pracuje metodą tzw. masek różnicowych /patrz.V.2.3.1/, przy użyciu procedur specjalnych języka Algol - 1204.

LIT/ a/ ; /b/.

Zwraca true, gdy a jest kodem litery.

DLIT /a/; /b/

Zwraca true, gdy a jest kodem dużej litery.

CYF /a/; /b/.

Zwraca true, gdy a jest kodem cyfry.

SŁOWNIK /N/; /i/.

Sprawdza czy liczba N jest zarejestrowana w słowniku, jeśli tak, to zwraca liczbę odpowiadającą N, w przeciwnym przypadku zwraca 0;
Pracuje metodą masek różnicowych. /V.2.3.1/

UPAKZ /A,Y/;

Przesuwa Y o 6 bitów w lewo, na powstałe miejsce wstawia 6 dolnych bitów z A.

UNPACK /n,i/; /i/.

Zwraca kod znaku zapisanego na i - tej /licząc od lewej/ szóstce bitów w słowie n.

UKOD /n/;

Ustawia wartość n na pierwszym wolnym miejscu w buforze BUF.

KONWERT; /i/

Przetwarza znakowy zapis całkowitej /w sensie składni Algolu 60/ liczby dziesiętnej, na wartość dziesiętną tej liczby zapisaną jako zmienną typu integer. Zwraca wartość tej liczby.

PISZID; /i/

Przepisuje identyfikator algolowski znajdujący się na początku nieprzetworzonego tekstu w buforze, do obszaru roboczego w pamięci bębnowej.
Liczona w znakach długość identyfikatora umieszczona jest w słowie bezpośrednio poprzedzającym pierwszy znak identyfikatora. Adres słowa zawierającego długość, zwiększony o 1024, zwracany jest jako wartość procedury.

Jeśli identyfikator jest dłuższy niż 64 znaki, to pobrane z bufora i przepisane na bęben będą tylko 64 pierwsze znaki.

SKIP;

Pomija fragment tekstu w buforze do najbliższego średnika.

PUSH/ a/;

POP/ /i/.

Procedury te organizują w pamięci operacyjnej stos. Korzystają z tablicy STOS i zmiennej top wskazującej wierzchołek stosu. Wartości tych obiektów nie powinny być zmieniane poza procedurami POP i PUSH.

PUSH/ a/ umieszcza element na stosie

POP zdejmuje wierzchołek stosu i zwraca jego wartość.

OPERATOR / a/ ;

Procedura generuje czwórkę liczb /a, arg 1, arg 2, wynik/, opisującą ralizację operatora a. Argumenty arg 1, arg 2 są pobierane ze stosu, nowy akumulator dla przechowania wyniku jest umieszczony na stosie. Utworzona czwórka umieszczona jest w buforze BP, w celu późniejszego wysłania do PaZ.

V.2.3 Działanie programu

Program główny napisany jest w Algolu, a jego struktura jest zgodna ze strukturą algorytmu, nie będziemy więc powtarzać opisu struktury. Opisane zostaną tylko rozwiązania, które nie są zupełnie naturalne.

V.2.3.1 Metoda masek różnicowych.

Używana do szybkiego porównywania danej liczby z elementami skończonego zbioru liczb /masek/.

Aby porównać parę liczb odejmujemy je i sprawdzamy czy wynik jest zerem. W celu przyspieszenia porównywania zastępujemy maski ich różnicami. Pozwala to wykorzystać pozostającą w akumulatorze różnicę między badaną liczbą a poprzednio sprawdzaną maską i w ten sposób zmniejszyć ilość przesłań.

V.2.3.2 Rozpoznawanie symboli złożonych.

Po napotkaniu znaku "_" /podkreślenie/ program zaczyna wczytywać symbol złożony. Symbolowi przyporządkowana zostaje liczba otrzymana przez upakowanie w jednej komórce pamięci kolejnych znaków symbolu i interpretacją otrzymanego ciągu bitów jako liczby całkowitej. Upakowanie przebiega następująco: kolejne znaki umieszczane są na 6-ciu bitach, od prawej strony. Dodanie jednego znaku przesuwa poprzednią zawartość o 6 bitów w lewo. Jako pierwszy upakowany jest znak "_". Pozostałe znaki "_" są pomijane. Upakowane są tylko znaki podkreślone.

Po upakowaniu pierwszego podkreślonego znaku przerywamy wczytywanie i sprawdzamy w słowniku, czy już upakowana część nie stanowi poprawnego symbolu. Jeśli nie, to staramy się doczytać jeszcze dwa znaki podkreślone, aby otrzymać symbol trzyznakowy.

V.4.3.3 Upakowanie nazw macierzy.

Znaki stanowiące nazwę zapisywane są na 6-ciu bitach każdy i umieszczane w jednej komórce pamięci, od lewej strony. Ostatni znak zajmuje skrajną prawą szóstkę bitów. Jeśli z lewej strony pozostało wolne miejsce, to zawiera ono spacje. Dopełnianie spacjami nie jest wykonywane w sposób jawny, gdyż komórki są zerowane przed umieszczeniem w nich znaków. Na maszynie Odra 1204 jest to równoznaczne z wypełnieniem komórki spacjami.

V.3 Opis programu mat 4.

V.3.1 Tablice i zmienne programu.

Zmienne typu integer.

db, nrk, ils

znaczenie zmiennych takie jak w mat1. Ich wartości są ustalone w mat1 i przekazane przez tablicę SOS.

ak - numer kryształu, którego przekład jest wyprowadzany,

Zmienne typu Boolean.

main - gdy main = true, to produkowany będzie program główny /uzupełniany automatycznie deklaracjami/.

Tablice.

Wszystkie tablice są typu integer.

Opisy tablic są podane w części V.2, gdyż są to tablice wypełniane informacją przez program mat1. Do programu mat4 są one przekazywane za pośrednictwem Pa2. Jedynym wyjątkiem są bufory BUF i BP, które nie przenoszą żadnej informacji między etapami. Ich zastosowanie jest takie same jak tablic BUF i BP w mat1.

V.3.4 Procedury.

Procedury PPRINT, OOUTCHAR, OUTINT, output mają tę wspólną własność, że wyprowadzają wielkości opisane parametramii aktualnymi, na urządzenia wyjściowe zależne od położenia kluczy pulpitu technicznego w momencie wywoływania procedury. Procedury te korzystają ze standartowych procedur wyjściowych języka Algol - 1204.

Wielkości wyprowadzane przez poszczególne procedury są następujące:

PPRINT - łańcuch /string/ będący parametrem aktualnym.

OOUTCHAR - znak, którego kodem jest parametr aktualny.

OUTINT - liczba całkowita, której wartość jest parametrem aktualnym.

output - obszar bębnowy zawart między adresami, które są parametrami aktualnymi.

OUTAG/a/;

Procedura generuje trzy pozycje wykazu "dla" w programie wynikowym. Te trzy pozycje ustalone przez OUTARG na podstawie wartości a i tablic przekazanych z mat1 opisują argument lub wynik operacji.

V.3.3 Działanie programu.

Program mat2 generuje tekst algolowy, będący ostatecznym wynikiem translacji. Tekst ten składa się z fragmentów tekstu źródłowego zawartego między kryształami, oraz z przekładów wyrażeń zawartych w kryształach. Fragmenty tekstu źródłowego są odnajdowane na podstawie zawartości tablicy LK i kopiowane na wyjście przy pomocy procedury output. Tekst przekładu konstruowany jest na podstawie stałego wzorca, uzupełnionego informacjami o argumentach operacji przez procedurę OUTARG. Wykorzystywane są przy tym czwórki wygenerowane przez program mat1 oraz zawartość tablic wypełnianych przez ten program.
/patrz cz.V.2/.

VI. Podręcznik operatora systemu Matralg.

System składa się z dwóch programów mat1 i mat2. Przewiduje się umieszczenie ich w bibliotece bębnowej. Wtedy przygotowanie programu do wykonania wykona system operacyjny po zleceniu "load mat1" /odpowiednio "load mat2"/. Jeśli programy nie są zapisane w bibliotece, tylko na taśmie binarnej, czy też jako programy algolowe na taśmie, to przygotowanie programu do pracy polega odpowiednio na wczytaniu taśmy binarnej - zlecenie "rbt", lub tłumaczeniu programu algolowego - zlecenie "t".

UWAGA: przy tłumaczeniu programu algolowego klucze 3 i 4 muszą być zwolnione. W dalszym ciągu będziemy określać te czynności jako przygotowanie programu do pracy, a właściwe czynności należy wybrać w zależności od zaistniałej sytuacji.

Użytkownik może zażądać wciśnięcia podczas działania programu pewnych kluczy na pulpicie technicznym. Wszystkie pozostałe klucze muszą być zwolnione.

Czynności wykonuje się w następującej kolejności:
Przygotować do pracy program mat1.
Uruchomić go zleceniem "w13".
Program pisze na monitorze "wait klucze" i czeka na dowolny znak z monitora. Przed dalszym działaniem programu wszystkie żądane klucze muszą być wciśnięte. /W szczególnym przypadku użytkownik może nie żądać wciśnięcia żadnych kluczy/.

Po wczytaniu znaku z monitora program wznawia działanie i wczytuje taśmę z danymi.

Wyniki programu są umieszczane w pamięci bębnowej. Część z nich jest wyprowadzana na drukarkę wierszową.

Jeśli program mat1 znajdzie błędy w danych, to na monitorze pojawia się komunikat:
"bledy w danych SORRY" i program kończy działanie. W tej sytuacji dalsza praca systemu nie ma sensu, więc nie należy uruchamiać programu mat2. Gdyby jednak został on uruchomiony, to zatrzyma się z sygnalizacją:
"błędy w pierwszym przebiegu SORRY".

Jeśli dane wejściowe są poprawne to mat1 wypisuje komunikat:
"OK load mat2 rez od a1 do a2" i kończy działanie. a1 i a2 oznaczają odpowiednio numer pierwszego i ostatniego słowa obszaru pamięci bębnowej zajętego przez wyniki przekazywane do programu mat2. Zawartość tego obszaru bębnowego oraz położenie kluczy pulpitu technicznego stanowią jedyne dane dla programu mat2.

Po poprawnym zakończeniu programu mat1 przygotowujemy do pracy program mat2. Następnie uruchamiamy go zleceniem "w13". Program pisze na monitorze "wait keys" i czeka na dowolny znak z monitora. Najpóźniej w tym momencie należy wcisnąć żądane klucze pulpitu technicznego. Następnie piszemy dowolny znak na monitorze. Program zaczyna działać i produkować wyniki, które z reguły będą poprawnym programem algolowym. W szczególności, jeśli jako urządzenie wyjściowe użytkownik wybierze /wciskając klucz 11/ perforator, to otrzymamy w wyniku taśmę z programem. Program ten może być następnie przetłumaczony i wykonany.

UWAGA 1: uruchomienie programu mat2 nie musi następować bezpośrednio po zakończeniu mat1. W międzyczasie można dawać maszynie inne zlecenia, pod warunkiem, że ich wykonanie nie spowoduje zniszczenia zawartości obszaru w PaZ zawierającego /obszar między adresami a1 i a2/.

UWAGA 2: wykonanie programu mat2 nie powoduje zniszczenia zapisu na bębnie, więc po jednokrotnym wykonaniu mat1 można wielokrotnie wykonywać mat2, np: z różnymi zestawami wciśniętych kluczy, czy też, gdy produkowany program zostanie uszkodzony /awaria perforatora, koniec taśmy itp./ można powtórzyć generację programu od początku /ponownie zleceniem "w13"/, ze wszystkimi akcjami opisanymi powyżej.

UWAGA 3: program mat2 produkuje wyniki na perforator, gdy wciśnięty jest klusz 11. Długość wyników może kilkakrotnie przekraczać długość taśmy z danymi.

BIBLIOGRAFIA

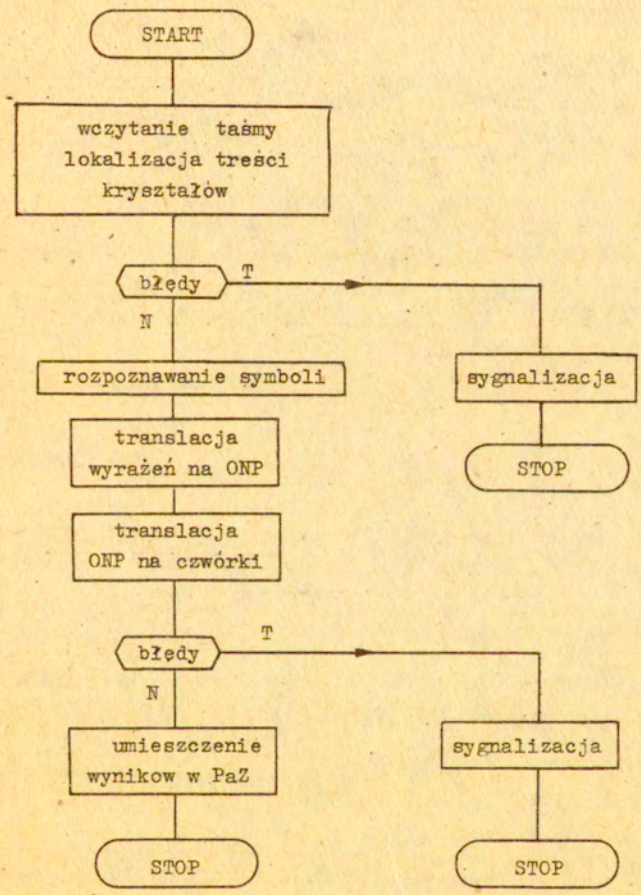
[1] Stefan Paszkowski, "Język Algol 60",
PWN, Warszawa 1974.

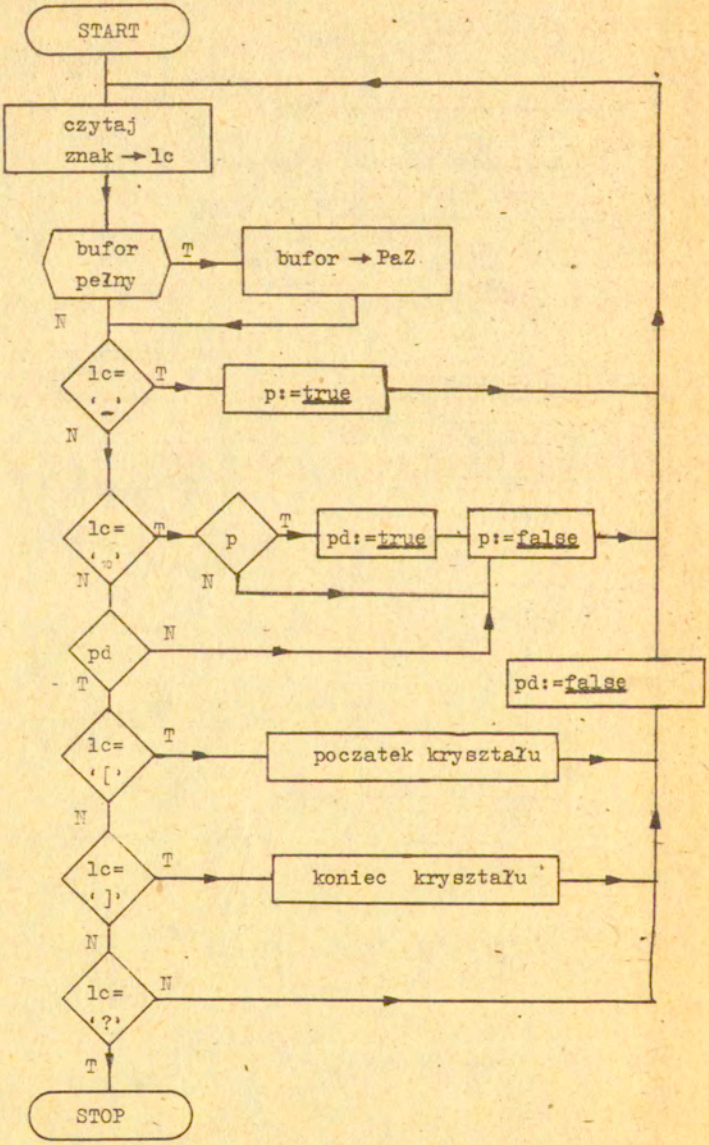
[2] W.M. Turski "Podstawy użytkowania maszyn cyfrowych
w ośrodkach naukowo technicznych".

[3] W.M. Turski : "Propedeutyka informatyki"
PWN, Warszawa 1977.

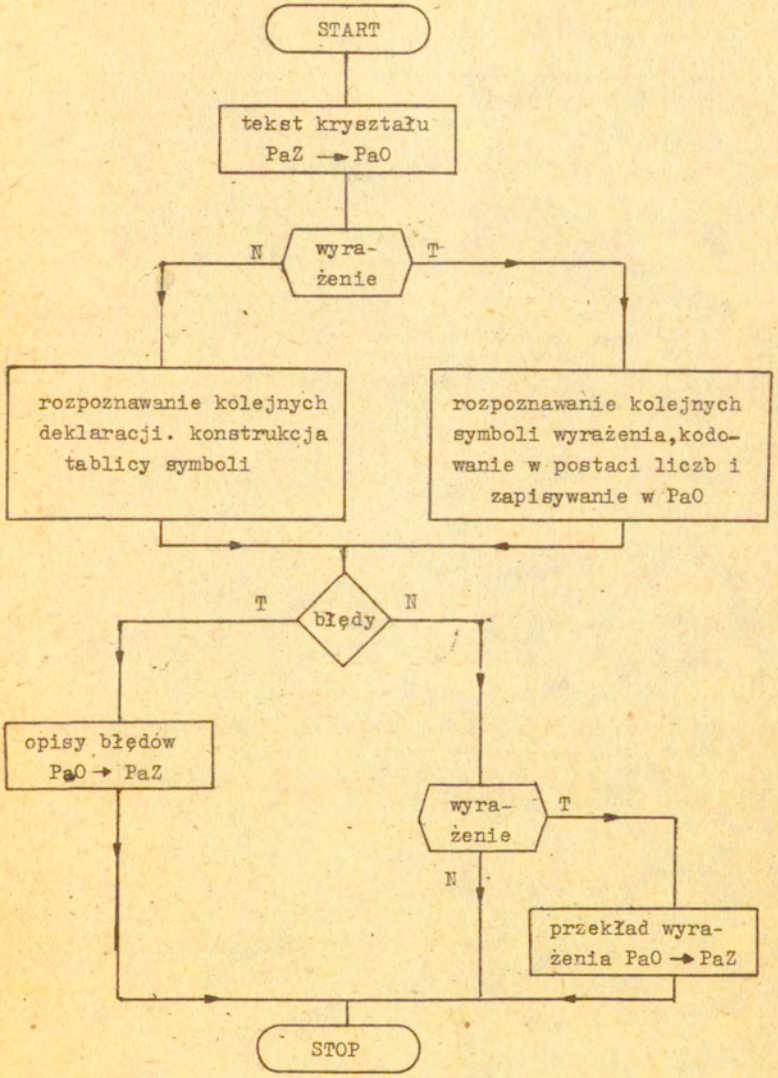
[4] K. Jerzykiewicz, J. Szczepkowicz, "Algol 1204.
System programowania m.c. 1204",
Warszawa 1973, PWN.

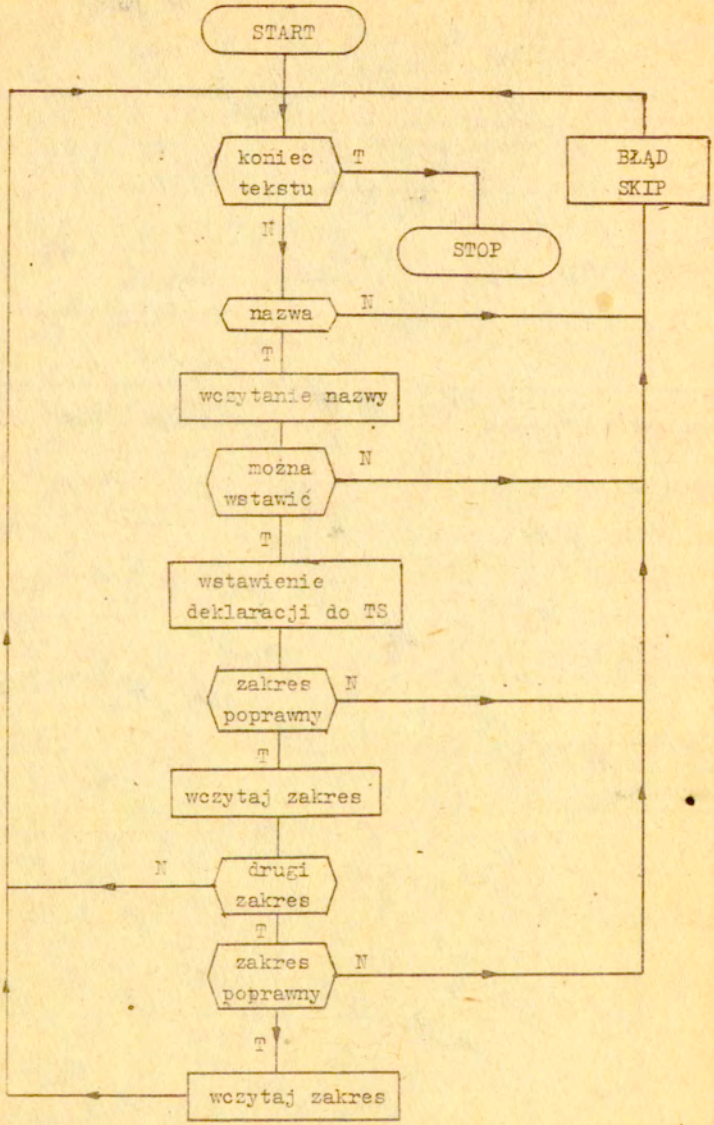
Program matl. SB-I



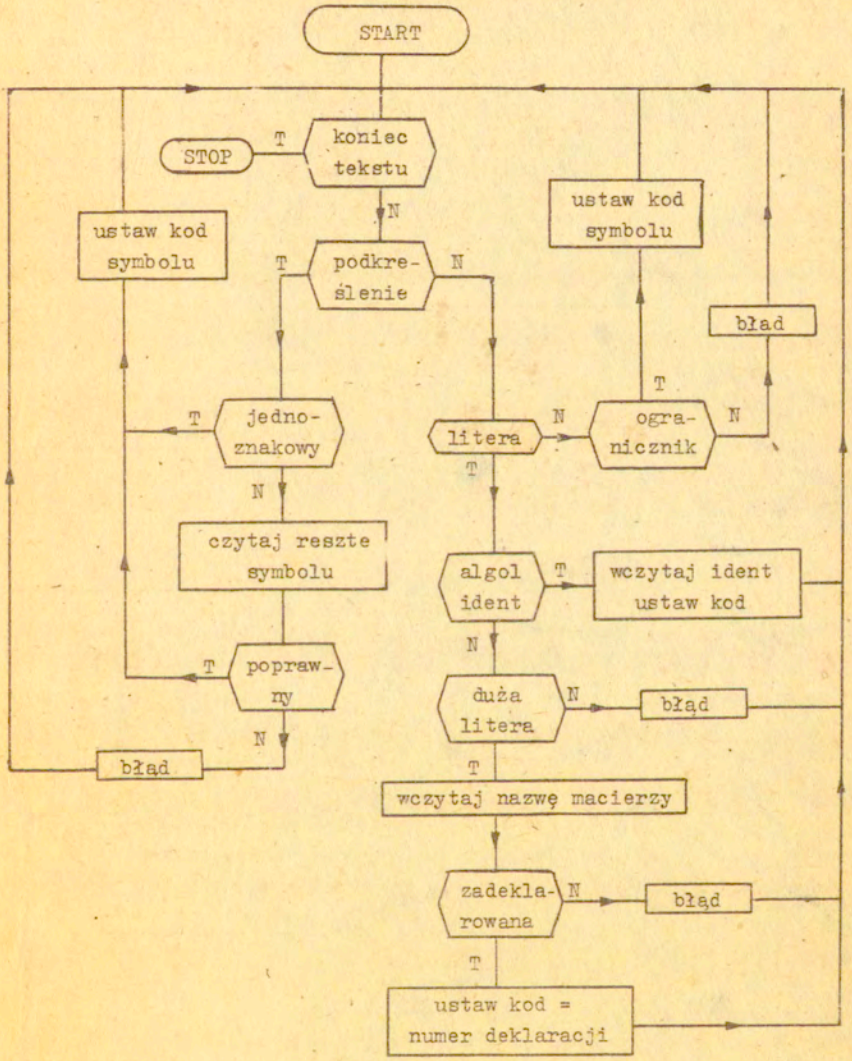


Rozpoznawanie symboli /scanner/ SB-III

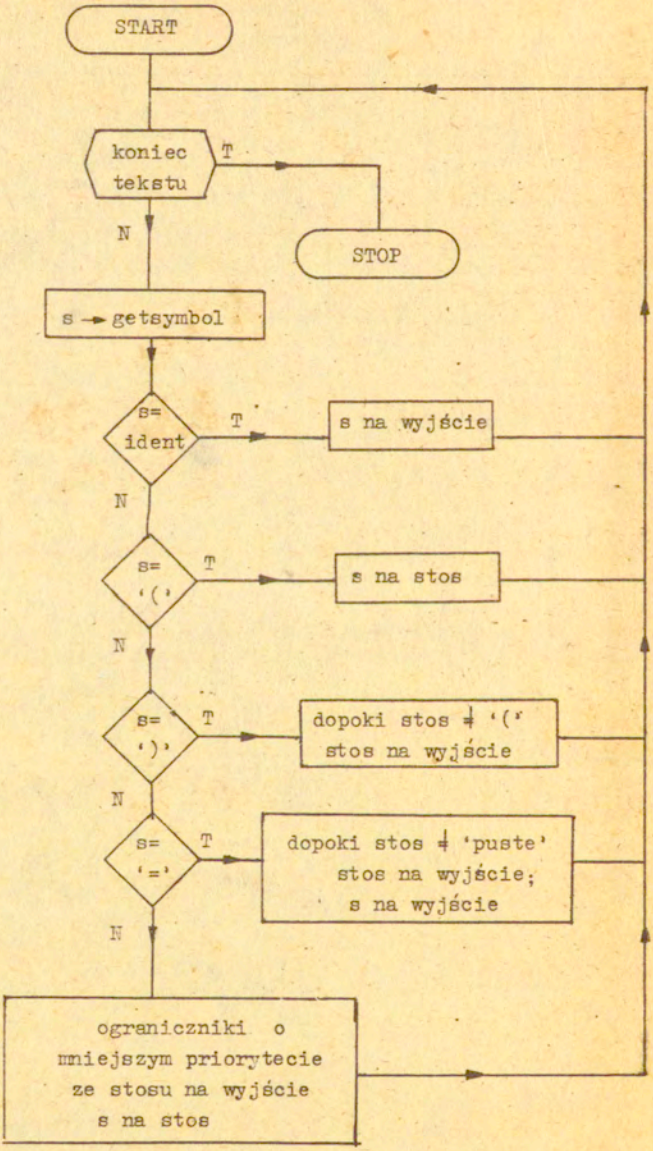




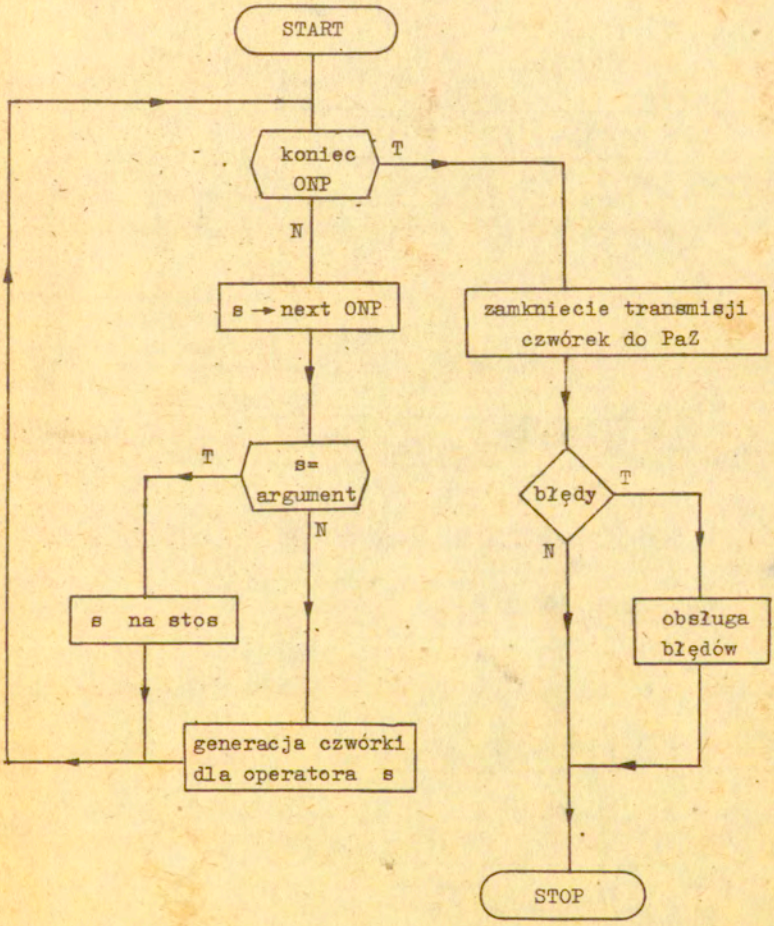
Rozpoznawanie symboli w wyrażeniu arytmetycznym SB-V



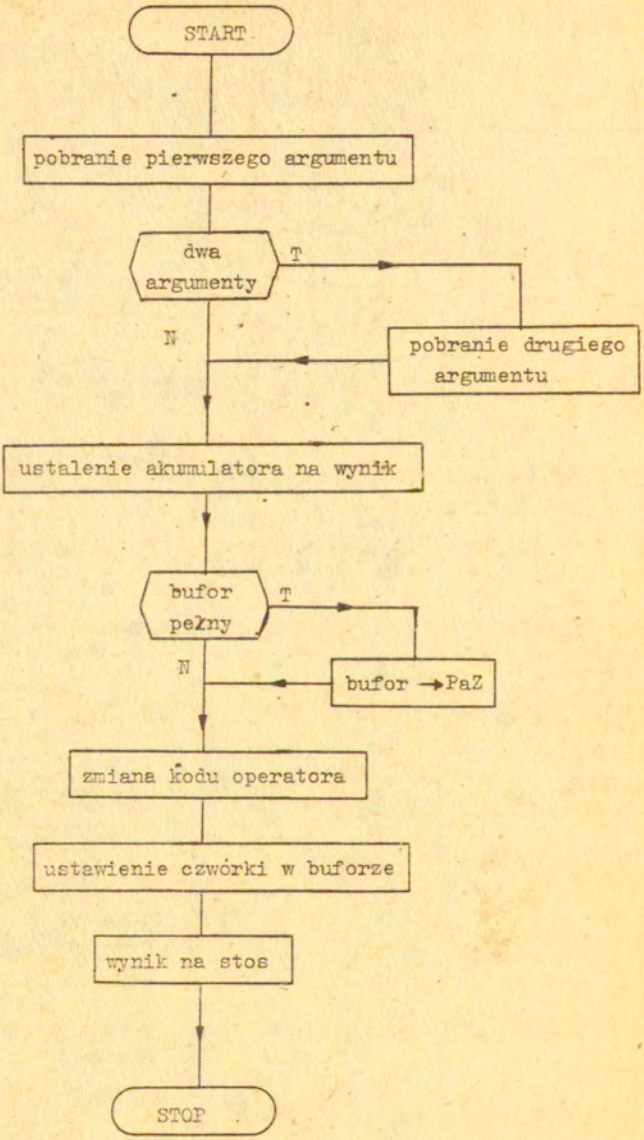
Translacja wyrażenia na ONP SB-VI



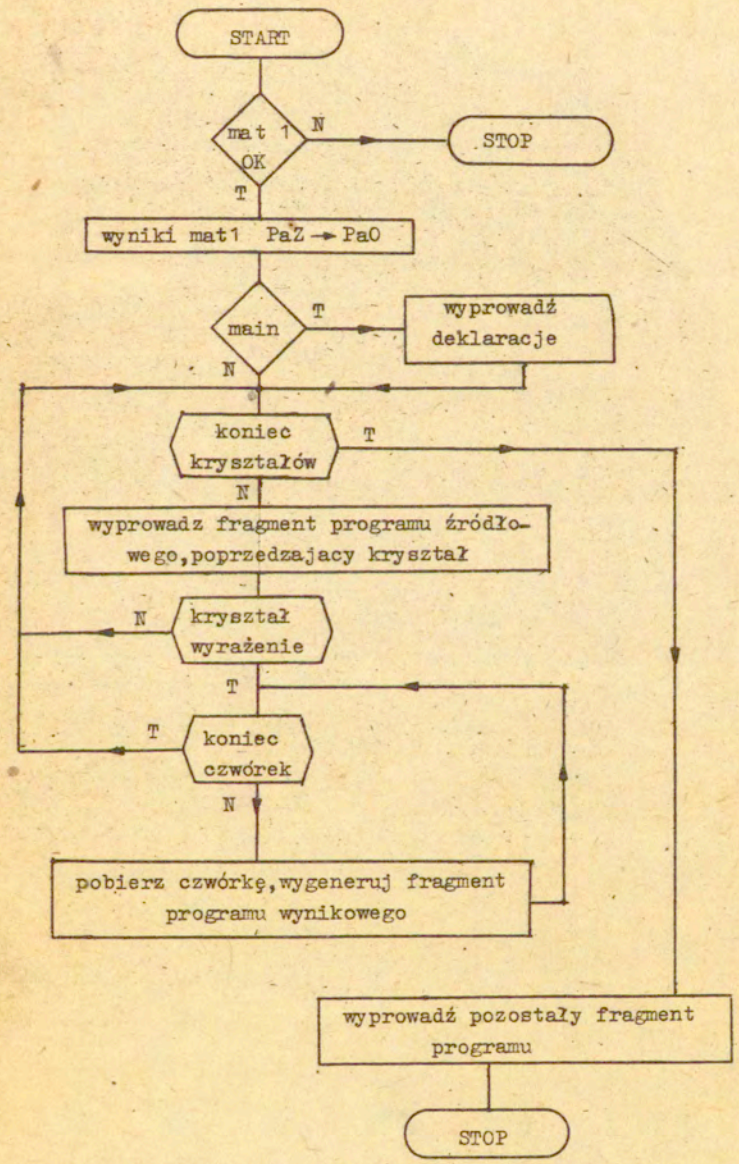
Translacja ONP na czwórki SB-VII



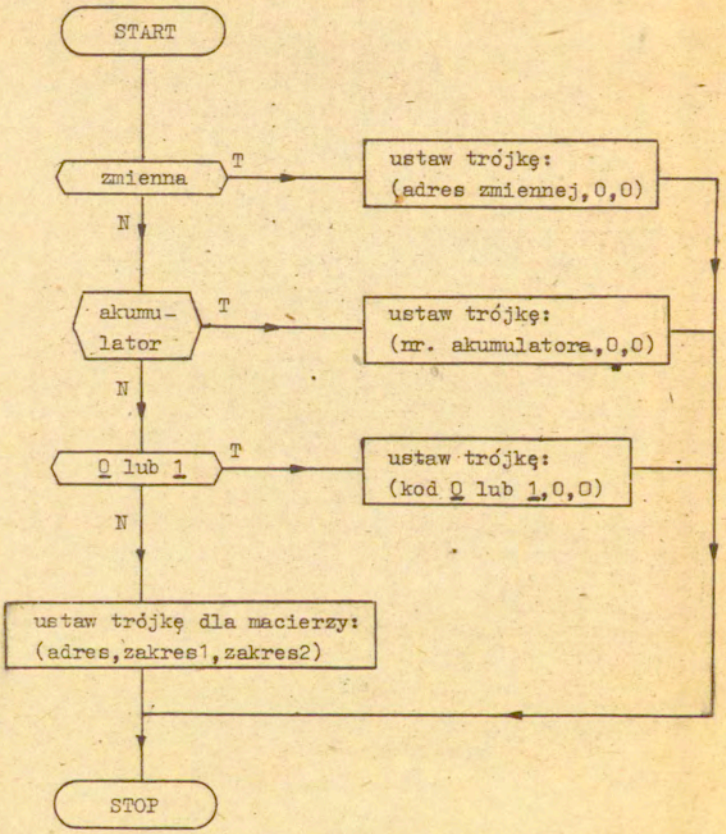
Procedura OPERATOR SB-VIII



Program mat2 SB-IX



Procedura OUTARG SB-X



TEKST PROGRAMU MAT1.

```
begin
comment program matralg tlumaczacy wyrazenia macierzowe;
integer db,afr,nafr,nrk,maxk,lb,maxb,ils,maxs,k,l,np,ds,dbp,lob;
boolean fe,feg;
integer array SOS[1:11];
wait(' klucze ');
db:=2048;
maxk:=128;
maxb:=20;
maxs:=64;
ds:=256;
dbp:=256;
feg:=false;
drumplace:=1;
SOS[1]:=-1;
todrum(1,SOS[1]);
SOS[1]:=db;
afr:=nafr:=drumplace:=12;
SOS[5]:=afr;
begin
integer array LK[0:maxk,1:2],NB[1:maxb,1:2],BUF[1:db],TS[1:maxs,1:4];
procedure output(l,pocz,kon);
value l,pocz,kon;
integer l,pocz,kon;
begin comment procedura wyprowadza obszar bebna zawarty miedzy
                pocz i kon na urzadzenie nr l;
integer i,ilb,r,j;
procedure out(a);
integer a;
if a>0 then
begin
    integer j;
fromdrum(a,BUF[1]);
for j:=1step 1 until ado
begin
i:=BUF[j];
if (i>112∧i<122)or(i>96∧i<106)or(i>81∧i<90) then outchar(109) else
if i=29 then outchar(96) else
if i=93 thenoutchar(81) else
if i=96 then begin print('VoVr'); i:=29 end;
outchar(i);
end
end procedury out;

setoutput(l);
ilb:=(kon-pocz)div db;
r:=kon-pocz-db×ilb;
drumplace:=pocz;
space(50);
outchar(58);
outchar(15);
for j:=1step 1 until ilb do
out(db);
out(r);
end procedury output;

comment wszystkie powyzsze deklaracje beda dostepne w calym programie;
```

```
fe:= false;
nrk:=0;
lb:=0;

comment  rozpoczynamy fragment czytajacy tasme na beben i robiacy
         mape lokacji na bebnie  oraz output zrodlowy;

begin
integer i,wyp,lc,refbuf;
boolean p,pd,pk;

procedure ERROR(n,m);
integer n,m;
begin
lb:=lb+1;
if lb≤maxb then
begin
NB[lb,1]:=n;
NB[lb,2]:=m
end;
if n<0 then fe:= true;
end  procedury ERROR;

refbuf:=ref(BUF[1])-1;
np:=1;
wyp:=0;
drumplace:=afr;
p:=pd:=pk:= false;
setinput(1);
IK[0,2]:=drumplace-1;

NEXTCHAR :

wyp:=wyp+1;
if wyp>db then
begin   comment bufor pelny,wysylamy na beben;
wyp:=1;
todrum(db,BUF[1]);
end  wysylania na beben;
comment tu na pewno jest miejsce na buforze;
lc:=inchar;
usak(lc);
mok1(wyp);
pska(+refbuf);

comment   zaczynamy badanie znaku;
if lc=29 then
begin    comment podkreslenie;
p:= true;;
go to NEXTCHAR
end   lc=29
else
if lc=65 then
begin  comment  napotkalismy % sprawdzamy,co bylo poprzednio;
if p then pd:= true;
p:= false;
go to NEXTCHAR
end lc=65
else
if pd then
```

```
     begin     comment poprz-ednio rozpoznane n,sprawdzamy czy teraz
koniec symbolu;
if lc=66 then
begin  comment   poczatek krysztalu;
if pk then ERROR(5,nrk)  else
begin
pk:= true;
nrk:=nrk+1;
if nrk>maxk then ERROR(1,nrk) else
LK[nrk,1]:=drumplace+wyp-3;
end;
pd:= false;
go to NEXTCHAR;
end  poczatku krysztalu
else
if lc=67 then
begin  comment koniec krysztalu;
if ¬pk then ERROR(2,nrk)
else
begin
pk:=pd:= false;
LK[nrk,2]:=drumplace+wyp-1;
if LK[nrk,2]-LK[nrk,1]+1>db then
begin
LK[nrk,2]:=LK[nrk,1]+db-1;
ERROR(3,nrk)
end;
end;
pd:= false;
go to NEXTCHAR
end  koniec krysztalu
else
if lc=68 then
begin   comment  koniec textu,wysylamy reszte bufora,sprawdzamy,czy
krysztal zakonczony;
if pk then
begin
ERROR(4,nrk);
nrk:=nrk-1;
end   if  pkk;
if wyp>0 then  todrum(wyp,BUF[1]);
end  koniec textu
else
begin
ERROR(6,nrk);
pd:= false;
go to NEXTCHAR
end
end pd czyli rozpoznawania symbolu zlozonego
else
begin
p:= false;
go to NEXTCHAR
end;

SOS[2]:=nrk;
SOS[7]:=nafr:=drumplace;
comment w tym miejscu tasma jest wczytana. listing przy kluczu 1;

if key(1) then  output(3,afr,drumplace);
```

```
comment . obsluga bledow znalezionych podczas czytania tasmy;

if lb>0 then
begin
fe:= true;
if lb>maxb then
begin
format('?»«»«ilosc␣bledow␣przekracza␣dopuszczalna␣wartosc?
»«»«informacje␣tylko␣o␣pierwszych␣1111␣bledach?');
print(maxb);
lb:=maxb;
end if lb> maxb;
line(3);
print('bledy␣w␣strukturze␣krysztalow␣');
line(2);
for k:=1 step 1until lb do
begin
comment  wydruk komunikatow obledach;
i:=NB[k,1];
if i>0 then
print('␣␣I␣␣')
else
print('␣␣B␣␣');
if i=2 thenprint('␣␣nieoczekiwany␣') else
if i=4 then print('koniec␣programu␣wewnatrz␣krysztalu␣'). else
if i=5 then print('␣␣nieoczekiwany␣') else
if i=3 then print(' za dlugi krysztal ') else
if i=1thenprint(' za duzo krysztalow ') else
if i=6 thenprint(' zly ogranicznik ') else
   print('␣␣blad ␣0');
format('1111?');
print(NB[k,2]);
end for k - wydruku komunikatow;
end lb>0;
end pierwszego przebiegu czytajacego i obrabiajacego tasme;
line(3);

comment    jesli w pierwszym przebiegu nie bylo bledow fatalnych,
           to przechodzimy do nastepnego etapu;

if fe then
begin  comment  tasma wejsciowa byla bledna;
print('koniec␣translacji?
  ');
setoutput(0);
print('?bledy w danych          SORRY?');
stop;
end  if fe;

comment × × × × ×× × × × × × ×  × × × × × × × × × × × × × ×;

comment  tu tasma wejsciowa byla poprawna;

if nrk>0 then
begin
      comment  rozpoczynamy blok scanne-ra przegladajacy krysztaly i
konstruujacy opisy macierzy w tablicy symboli.wyrazenia arytmetyczne
tlumaczone na symbole kodowane jako liczby,liczby ujemne oznaczaja
adresy w tablicy symboli.tekst calego k-rysztalu jest w buf,tam tez
```

umieszczamy przeklad. Krysztal typu dim nie zostawia przekladu - podobnie jak wyrazenie, w ktorym znaleziono bledy. W obu przypadkach znak adresu poczatku krysztalu /wLK/ ustawiamy na minus;

```
booleannextd,n,dl;
integer lo,li,kk,ak,lc,i,j,w,k,p;
integer array TMAXA[1:nrk];

integer procedure ZNAK;
begin comment procedura pomija wszelkie smieci i zwraca kod
pierwszego znaku znaczacego.li - ostatnio pobrane miejsce w buforze,
kk - koniec tekstu w buforze;
integer a,p;
ZNAK:=0;
a:=0;
CZYT:
li:=li+1;
if li≤kk then
begin comment czytamy z bufora;
dl:= false;
a:=BUF[li];
if a>63 then
begin comment obcinamy duze litery;
a:=a-64;
dl:= true;
end a>63;
comment teraz sprawdzamy czy smieci;
usak(a);
skz(CZYT);
osak(11);   skz(CZYT);
osak(1);    skz(CZYT);
osak(1);    skz(CZYT);
osak(1);    skz(CZYT);
osak(1);    skz(CZYT);
osak(13);   skz(CZYT);
osak(2);    skz(CZYT);
osak(1);    skz(CZYT);
osak(11);   skz(CZYT);
osak(2);    skz(CZYT);
osak(2);    skz(CZYT);
osak(1);    skz(CZYT);
osak(14);   skz(CZYT);
osak(1);    skz(CZYT);
osak(1);    skz(CZYT);
comment jesli nie wyskoczylismy,to dobry znak;
if dl then a:=a+64;
ZNAK:=a;
end  czytamy z bufora;
end procedury ZNAK;

boolean procedure LIT(a);
value a;
integer a;
begin
comment procedura zwraca true gdy ajest kodem litery;
if a<64 thena:=a+64;
LIT:=(a>81∧a<90)∨(a>96∧a<106)∨(a>112∧a<122)
end  procedury LIT;
```

```
booleanprocedure DLIT(n);
integer n;
comment  DLIT= true gdy njest kodem  duzej litery;
DLIT:=(n>81∧n<90)∨(n>96∧n<106)∨(n>112∧n<122);
booleanprocedure CYF(n);
comment  CYF=tue , gdy n jestkodem cyfry;
integer n;
CYF:= n=16∨(n>0∧n<10);

integer procedure SLOWNIK(N);
value N;
integerN;
begin   comment procedura rozpoznaje symbol i przyporzadkowuje mu kod.
                gdy nierozpoznany,to kod= 0;
integer S;
setoutput(3);
usak(N);
osak(26);     skz(e1);
osak(46);     skz(e2);
osak(8);      skz(e3);
osak(1777);   skz(e4);
osak(15);     skz(e5);
osak(3);      skz(e6);
osak(7);      skz(e7);
osak(6);      skz(e8);
ssak(27);     skz(e9);
osak(1);      skz(e10);
osak(3);      skz(e11);
osak(7755310);   skz(e12);
osak(80862);       skz(e13);
comment  jesli doszlismy tutaj,to symbol nie zostal rozpoznany;
S:=0;   go to st;

e1:   S:=401;     goto st;
e2:   S:=  1;     go to st;
e3:   S:=103;     goto st;
e4:   S:= -2;     go to st;
e5:   S:= -1;     go to st;
e6:   S:=502;     go to st;
e7:   S:=201;     gotost;
e8:   S:=102;     go to st;
e9:   S:=101;     goto st;
e10:  S:=  2;     go to st;
e11:  S:=202;     go to st;
e12:  S:=501;     go to st;
e13:  S:=503;
st:
SLOWNIK:=S;
end   procedury SLOWNIK;

procedure UPAKZ(A,Y);
value A;
integer A,Y;
begin  comment  procedura przesuwa bo 6 bitow w lewo,na powstale mie-
sce wstawia 6 dolnych bitow z a;
integer b;     b:=Y;
usak(63);   mlka(A);
comment  obcielismy a do dolnych 6-ciu bitow;
usak(b);     lnk(+6);
```

```
dlak(A);   pska(b);
Y:=b;;
end   procedury UPAKZ;

integer procedure UNPACK(n,i);
value n,i;
integer n,i;
begin  comment   procedura zwraaca kod znaku zapisanego na i-tej
   od   lewewj szostce bitow.i=1,2,3,4;      integer  m,p;
p:=4-i;
for i:=1 step 1 until p do
begin
usak(n);       pnk(+6);
pska(n);
end  for i     ;
m:=63;   usak(m);       mlka(n);
UNPACK:=n;
end  procedury UNPACK;

procedure UKOD(n);
value n;
integer n;
begin  comment procedura wstawia parametr na pierwsze wolne w buforze
                komunikacji z bebnem;
if ¬fe then
begin
lo:=lo+1;
if  lo>db then  ERROR(-1000,0)
else
BUF[lo]:=n;
end  ¬ fe
end   procedury UKOD;

integer procedure KONWERT;
begin  comment procedura  czyta kolejne cyfry az do terminatora i
        oblicza wartosc dziesietna wczytanej liczby calkowitej.
        jesli liczba ma wiecej niz 4 cyfry,to zwraca wartosc 0. po
        wyjsciu na lc terminator liczby;
integer k,w;
k:=w:=0;
for k:=k+1 while CYF(lc) do
begin
if k>4 thenw:=0
else
begin
if lc=16 then  lc:=0;
w:=10×w+lc;
end;
lc:=ZNAK;
end  for;
KONWERT:=w;
end   procedury KONWERT;

integer procedure PISZID;
begin comment lc zawiera pierwszy znak identyfikatora.piszemy ten
   id. na beben.zwracamy wartosc  adresu pierwszego slowa zapisu
                ( zawierajacego dlugosc id);
integer wp,kp,k;
integer array BP[1:65];
```

```
wp:=1;     kp:=65;
PISZID:=0;
for wp:=wp+1 while wp≤kp∧(LIT(lc)∨CYF(lc)) do
begin
comment wczytujemy ident do bufora BP;
BP[wp]:=lc;
lc:=ZNAK;
end for wp - wczytywania id;
BP[1]:=wp-1;
drumplace:=nafr;
todrum(wp-1,BP[1]);
PISZID:=nafr+1024;
nafr:=drumplace
end procedury PISZID;

procedure SKIP;
begin comment procedura przewija tekst do napotkania srednika
(srednik niewczytany) lub do konca gdy nie ma srednika;
integer a,b;
b:=0;
for li:=li+1 while li≤kk∧b≠90 do b:=BUF[li];
if b=90 then li:=li-2;
end procedury skip;

procedure ERROR(a,b);
value a,b;
integer a,b;
begin comment procedura obslugi bledow;
integer x;
if lb=0then
begin comment znaleziono pierwszy blad w krysztale. inicjalizacja
         obslugi bledow;
lob:=2;
BUF[2]:=np;
LK[ak,1]:=-abs(LK[ak,1]);
LK[ak,2]:=-abs(LK[ak,2]);
fe:=true;
end if lb=0 - pierwszy blad;
lb:=lb+1;
if lb<maxb ∧ lob<li-1 then
begin
lob:=lob+1;
BUF[lob]:=a;
lob:=lob+1;
BUF[lob]:=b;
end
else
BUF[2]:=-abs(BUF[2]);
end procedury ERROR;

comment × ×× × × × × × ×× × × × × ××× × × × ×× ×;

comment zaczynamy dzialanie scannera.rozpoczynamy petle po ilosci
        krysztalow;
ils:=2;
np:=2;
for i:=1step 1 until maxs do
for j:=1 step1until4 do
TS[i,j]:=0;
for ak:=1 step 1 until nrk do
```

```
begin  comment  blok obrabiajacy jeden krysztal.Przy wcisnietym
kluczu 3 drukujemy tresc krysztalu na drukarce.Nastepnie
sprowadzamy tresc krysztalu do bufora BUF;

lb:=0;
fe:=false;
if key(5) then
begin
format('?? ␣␣␣krysztal numer␣␣111??');
print(ak);
output(3,LK[ak,1],LK[ak,2]+1);
end  outputu zrodlowego;
drumplace:=abs(LK[ak,1]);
fromdrum(LK[ak,2]-LK[ak,1]+1,BUF[1]);
kk:=LK[ak,2]-LK[ak,1]-2;

lc:=1;
li:=3;
lc:=ZNAK;
i:=ZNAK;
comment  obrobke zaczynamy od sprawdzenia typu krysztalu;
if lc=29∧i=52 then
begin
comment  napotkalismy na poczatku krysztalu d.zakladamy ,ze caly
krysztal zawiera deklaracje tablic. w tym bloku bedziemy je
rozpoznawac i wypelniac tablice symboli TS.
wracamy na poczatek tekstu;
li:=3;
lc:=ZNAK;
w:=0;
for i:=1,2,3 do
begin  comment  polykamy dim;
lc:=ZNAK;
UPAKZ(lc,w);
lc:=ZNAK;
end  czytania dim;
if w≠216676 then
ERROR(-8,w)
else
begin  comment poprawny poczatek krystalu;
nextd:= true;
for i:=1 while nextd∧lc≠0 do
begin  comment blok w ktorym rozpoznajemy jedna deklaracje;
comment tu spodziewamy sie napotkac nazwe;
if DLIT(lc) then
begin  comment  wczytywanie nazwy;
n:=true;
w:=0;
j:=0;
for j:=j+1 while DLIT(lc)∨CYF(lc) do
begin comment  doczytujemy reszte nazwy;
if j<5 then
UPAKZ(lc,w);
lc:=ZNAK;
end  doczytywania reszty;
if j>5 then
ERROR(14,w)
end if DLIT(lc) -wczytywania nazwy
else
begin
n:= false;
```

```
ERROR(-9,lc);
SKIP
end;
 comment  jesli nazwa jest rozpoznana,to n=true,w zawiera
 upakowana nazwe,lc pierwszy znak po nazwie.Jesli nie to tekst jes
 przewiniety do srednika lub do konca;
if nthen
begin   comment  wstawiamy nazwe do tablicy symboli;
ils:=ils+1;
if ils>maxs then
begin  comment za wiele symboli.konczymy przetwarzac dim;
ERROR(-10,w);
SKIP;
nextd:= false
end  za duzo symboli
else
begin   comment  tu mozna wstawic symbol.przegladamy TS zeby
zobaczyc    czy taki symbol jest juz w tablicy;
TS[ils,1]:=w;
p:=ils-1;
for k:=3 step 1until p do
if TS[k,1]=w then
begin
ERROR(11,w);
ils:=ils-1
end;
comment  teraz czytamy zakresy.nie wiemy,czy bedzie 1 czy2.trzeba
to sprawdzic.   najpierw jednak musi byc (;
if lc≠72 then
begin
ERROR(-12,lc);
SKIP
end  lc≠72
else
begin   comment   czytamy zakresy;
k:=0;
ZAKR:
k:=k+1;
lc:=ZNAK;
p:= if CYF(lc) then KONWERT else
   if LIT(lc) then -PISZID else 0;
TS[ils,k+2]:=p;
if p=0 then ERROR(-13,TS[ils,1]);
commentpetla zaczynajaca sie od etykiety ZAKR WYKONA  sie tyle
razy,ile jest    zakresów w definicji (poprawnej);
if lc=48∧k=1 thengo toZAKR;
TS[ils,2]:=k;
end   czytamy zakresy;
comment  tu na lc powinien byc ) ⫽srednik jesli maja byc nastepne
deklaracje  lub ) ⫽koniec tekstu;
if lc≠80 then
begin
ERROR(-12,lc);
SKIP
end   lc≠80;
comment zakonczylismy rozpoznawanie jednej deklaracji;
end  ustawiania symbolu w ts
end if n - rozpoznana nazwa;
lc:=ZNAK;
nextd:=nextd∧lc=90;
lc:=ZNAK;
```

```
end  for iwhile nextd czyli petli rozpoznajacej jedna deklaracje;
IK[ak,1]:=-abs(IK[ak,1]);
end   poprawny poczatek krysztalu
end rozpoznawania krysztalu typu dim

else
begin
comment    druga glowna sciezka sterowania w scannerze - rozpozna-
wanie   wyrazenia arytmetycznego;
li:=3;
lo:=1;
lc:=ZNAK;
for j:=1while lc≠0do
if  lc=29 then
begin   comment    blok
rozpoznajacy symbole podkreslone. najpierw sprawdzamy
        czy symbol jednoliterowy;
w:=lc;
lc:=ZNAK;
UPAKZ(lc,w);
p:=SLOWNIK(w);
if p≠0 then UKOD(p)
else
begin   comment  symbol nie byl jednoznakowy;
for i:=1,2 do
begin  comment doczytujemy reszte znakow do symbolu;
lc:=ZNAK;
if lc≠29 then
begin
ERROR(-8,w);
go to KONIEC PODKR
end
else
UPAKZ(ZNAK,w);
end  for i - doczytywania symbolu;
p:=SLOWNIK(w);
if p≠0 then UKOD(p) else ERROR(-8,w);
end   symbol wieloznakowy;

KONIEC PODKR:
lc:=ZNAK;
end  symbolu podkreslonego

else
if LIT(lc) then
begin  comment
 identyfikator algolowy lub nazwa macierzy,zaleznie od kontekstu;
B:
if BUF[lo]=401 then
UKOD(-PISZID)
else

if DLIT(lc)  then
begin comment  wczytywanie nazwy macierzy;
w:=0;      i:=0;
for i:=i+1 while DLIT(lc)VCYF(lc) do
begin   comment  doczytujemy reszte nazwy;

if i<5 then
UPAKZ(lc,w);
lc:=ZNAK;
```

```
end doczytywania nazwy;
if i>5 then   ERROR(14,w);
comment  teraz w upakowana nazwe.przegladamy tablice symboli;
for i:=3 step 1until ils do
if w=TS[i,1] then go to JEST;
JEST:
if i=ils+1then ERROR(-7,w)   elseUKOD(-i);
end   wczytywania nazwy macierzy

else
begin comment wystapienie litery jest nieprawidlowe;
ERROR(-9,lc);
lc:=ZNAK
end  litera nieprawidlowo
end  rozpoznawania nazwy - if lit
else
begin   comment   symbol niepodkreslony i nie nazwa;
p:=SLOWNIK(lc);
if p≠0thenUKOD(p) else ERROR(-12,lc);
lc:=ZNAK;
end niepodkreslony i nie nazwa;
comment
  tu koniec rozpoznawania jednego symbolu.jesli nie koniec tekstu,
 to lc zawiera pierwszy niezbadany znak.wracamy na poczatek petli
for j   while lc≠0;
end
  koniec rozpoznawania krysztalu zawierajacego wyrazenie arytm.;
comment
wysylamy na beben przeklad krysztalu lub liste opisow bledow;
feg:=feg∨fe;
if fe thenlo:=lob;
BUF[1]:=lo;
drumplace:=abs(LK[ak,1]);
if ¬(LK[ak,1]<0∧LK[ak,2]>0) then
todrum(lo,BUF[1]);
end   for ak - petli po ilosci krysztalow;

drumplace:=nafr;
SOS[10]:=drumplace;
if ils>0 then
todrum(ils×4,TS[1,1]);
SOS[3]:=ils;
begin
integer top,i,n,s,pm,sp,ilop,ii,m,nrop,aa,lbp,dkk,maxa;
integerarrySTOS[1:ds],LPG[1:nrk,1:2],BP[1:dbp];

 procedure PUSH(a);
value a;
integer a;
begin  commentprocedura wklada na stos sprawdzajac przepelnienie;
top:=top+1;
if top>ds then
begin
ERROR(-40,ak);
go toENDPETLI
end if top>ds
else
STOS[top]:=a;
end  procedury PUSH;

integer procedure POP;
begin
```

```
comment  procedura pobiera wierzołek stosu i zwraca jego
wartosc.gdy stos pusty to 0;
integer x;
if top>0 then
begin
POP:=x:=STOS[top];
top:=top-1;
end top>0
else
POP:=x:=0
end    procedury POP;

procedure OPERATOR(a);
value a;
integer a;
begin
comment  procedura obsluguje realizacje operatora o kodzie a;
integer arga,argb;
nrop:=nrop+1;
arga:=POP;
if arga=0 then
begin
ERROR(-31,nrop);
top:=0;
i:=kk
end arga=0;
comment  tu drugi argument juz pobrany;
if a<500 then
begin   comment  operator a jest dwuargumentowy.pobieramy
pierwszy argument;
argb:=POP;
if argb=0then
begin
ERROR(-31,nrop);
top:=0;
i:=kk;
end    argb=0
end a<500 - operator dwuargumentowy
else
argb:=0;
comment  tu ustalony pierwszy agrument;
aa:=aa+1;
format('?␣operator=1111␣␣␣␣argumenty(1111␣,␣1111)␣␣␣␣wynik=
1111?');
comment  wpisujemy do bufora 4 parametry do przyszlej
generacji kodu dla rozpoznanej operacji;
if lbp+4>dbp then
begin    comment  bufor pelny,trzeba go wyslac na beben;
todrum(lbp,BP[1]);
lbp:=0;
end  wysylania na beben;
comment  tu na pewno jest miejsce w buforze;
lbp:=lbp+1;
comment  kod operacji musimy przekodowac przed wyslaniem;
BP[lbp]:=
if a=501 then     1 else
if a=502 then   256 else
if a=503 then    32 else
if a=401 then     4 else
if a=201 then    16 else
```

```
if a=202 then  64 else
if a=101 then   8 else
if a=102 then 128 else
if a=2   then 2048 else 0;

lbp:=lbp+1;
BP[lbp]:=argb;
lbp:=lbp+1;
BP[lbp]:=arga;
lbp:=lbp+1;
BP[lbp]:=-aa-512;
comment  parametry ustawione;
if a#2then PUSH(-aa-512);

end  procedury operator;

SOS[8]:=drumplace;
setinput(1);
setoutput(3);
li:=db;
for ii:=1 step 1 until nrk do
if LK[ii,1]>0 then
begin
comment   krysztal nr ii zawiera wyrazenie arytmetyczne;
ak:=ii;
fe:=false;
lb:=0;       np:=3;
lc:=1;       ilop:=0;
s:=drumplace;
drumplace:=LK[ii,1];
fromdrum(db,BUF[1]);
drumplace:=s;
kk:=BUF[1];      lo:=1;
for n:=2 step 1 until kk do
begin
s:=BUF[n];
if s<0 then      UKOD(s)     else
begin
comment  napotkany symbol jest ogranicznikiem;
if s=1 then
PUSH(s)
else
if s=103 then
begin     comment ,nawias ) odczytujemy ze stosu az do nawiasu (;

for pm:=POP while pm>1 do UKOD(pm);
if pm=0  then ERROR(-20,ilop);
end odczytywania ze stosu do (
else
if s=2then
begin
comment  napotkalismy = po jej prawej stronie powinien byc
dokladnie jeden identyfikator;

pm:= if n#kk-1 then 0 else BUF[kk];
n:=kk;
if pm<-2 then
begin    comment OK jest jeden identyfikator
           wypisujemy wszystko ze stosu,sprawdzajac,czy nie ma
```

```
            nawiasow (.na koncu wypisujemy = i ident;
s:=pm;
for pm:=POP while pm>0 do
if pm=1 then
ERROR(-21,ilop)
else
UKOD(pm);
UKOD(s);
UKOD(2);
BUF[1]:=lo;       lo:=0;
end if pm<-2 - poprawne podstawienie
else
ERROR(-22,ilop);
end if s=2 - obrabiania podstawienia
else
begin comment  napotkany symbol jest ogranicznikiem roznym
od (,),=  dopisujemy go na stos po ewentualnych odczytaniach
stosu;
sp:=s div 100;
format('111');
for sp:=sp while sp≤(if top>0 then STOS[top] div 100 else 0) do
UKOD(POP);
PUSH(s);
ilop:=ilop+1;
end napotkany ogranicznik rozny od ( ) =
end napotkany ogranicznik
end for n - przegladania jednego krysztalu;
TMAXA[ak]:=ilop;
if lo≠0 then ERROR(-23,0);
comment  translacja krysztalu na onp zakonczona;
comment
  jesli byly bledy, to konczymy obrobke krysztalu - wyskok do
konca petli po ilosci krysztalow;
if fe then go to ENDPETLI;

comment  tu onp jest poprawne, wykonujemy nastepny etap
translacji;
comment
 poczatek programu translacji onp na kod      × × × × × × × × × × ;
np:=4;       nrop:=0;      aa:=0;
lbp:=0;      top:=0;
LPG[ii,1]:=drumplace;
kk:=BUF[1];
for i:=2 step 1 until kk do
begin comment  przegladamy onp;
s:=BUF[i];
if s<0 then   PUSH(s)
else
OPERATOR(s);
end for i- przegladania onp dla probnej translacji krysztalu;
if top>0∧(¬fe) then ERROR(-32,top);
comment  koniec translacji, trzeba jeszcze ewentualnie wyslac
      koncowke z bufora na beben;
if lbp≠0 then
begin
todrum(lbp,BP[1]);
end lbp≠0;
LPG[ii,2]:=drumplace-LPG[ii,1];
ENDPETLI:
if fe then
```

```
begin
feg:=true;
BUF[1]:=lob;
nafr:=drumplace;
drumplace:=-LK[ii,1];
todrum(lob,BUF[1]);
drumplace:=nafr;
end  if fe;
end  petli po ilosci krysztalow
nafr:=drumplace;

if feg then
begin   comment   stwierdzono bledy w tekscie zrodlowym.
      informacje o bledach sa na bebnie,sciagamy je i
      drukujemy komunikaty;
for ak:=1 step1 until nrk do
if LK[ak,1]<0∧LK[ak,2]<0 then
begin
drumplace:=-LK[ak,1];
fromdrum(db,BUF[1]);
format('??krysztal␣nr␣␣111?
sprawdzanie ␣poprawnosci␣przerwano␣po␣␣11␣przebiegu??');
m:=abs(BUF[2]);      print(ak,m);
n:=BUF[1] div 2 -1;
if n>maxb thenn:=maxb;
n:=n×2+2;            i:=1;
for i:=i+2while i<n do
begin  comment na podstawie kolejnych dwojek liczb
         drukujemy komunikaty o bledach;
k:=BUF[i];   l:=BUF[i+1];
if k≤0 then
print('?␣␣B␣␣')
else       print('?␣␣I␣␣');
if m=2 then
begin  comment bledy z drugiego przebiegu (scanner);
if k=-8 then print('zly symbol zlozony') else
if k=-9 thenprint('nazwa␣oczekiwana␣') else
if k=-10 thenprint('za␣duzo␣nazw␣') else
if k=-11 then print('podwojna␣deklaracja␣') else
if k=-12 thenprint('niedozwolony␣znak␣') else
if k=-13 then print('zla␣wartosc␣zakresu␣') else
if k=-7 then print('niezadeklarowana␣nazwa␣') else
if k=14 then print('za␣duga␣nazwa␣') else
begin
k:=0;     print('??blad 0??');
end;
if k=-9∨k=-12 then
outchar(1)
else
if k≠0 then
for k:=1,2,3,4 do
outchar(UNPACK(1,k));
end if m=2 - bledy scannera
else
begin   comment  bledy translacji na onp i na operacje (CZWORKI);
if k=-20 then print('brak␣(␣') else
if k=-21 then print('brak␣)␣') else
if k=-22 thenprint('zle␣podstawienie␣') else
if k=-23 then print('brak␣podstawienia␣') else
if k=-31 thenprint('-argument␣') else
if k=-32 then print('+argument␣') else
```

```
if k=-40 then print('przepelnienie stosu') else
print('??blad o??');
end   bledy przebiegu 3 i 4;
end  for i - drukowania komunikatow;
if BUF[2]<0 then
 print('??ilosc bledow przekracza dopuszczalna wartosc
informacje o pozostalych bledach tego krysztalu pominiete??');
end for ak,if LK   PETLI PO ilosci krysztalow;
end  if feg - bledy w programie zrodlowym
else
begin  comment  transmisja wynikow do drugiego przebiegu
          za posrednictwem bebna;
if nrk>0 then
begin
if key(7) then
begin  comment wydruk tablicy symboli;
print('?tablica symboli?');
for i:=3 step 1 until ils do
begin
lc:=TS[i,1];
for k:=1,2,3,4 do  outchar(UNPACK(lc,k));
format('␣␣␣1111␣␣1111?');
for k:=2,3,4 do  print(TS[i,k]);
end for i
end  wydruku TS;
drumplace:=nafr;
SOS[6]:=drumplace;
todrum(nrk×2,LK[1,1]);
SOS[9]:=drumplace;
todrum(nrk×2,LPG[1,1]);
SOS[11]:=drumplace;
todrum(nrk,TMAXA[1]);
end;
end    bloku translujacego na czworki i na onp
end
end;
setoutput(0);
if feg then
begin
print('?bledy w danych                    SORRY?');
stop
end
else
begin
format('111111?');
print('?OK load mat2
rez 1 do',drumplace)
end      ;
drumplace:=1;
todrum(11,SOS[1]);
setoutput(3);
print('??koniec translacji?????');
end;
```

TEKST PROGRAMU MAT2.

```
begin
integer ak,i,db,n,s,pm,lo,nrk,kk,lc,sp,ilop,ii,j,k,l,m,nrop,
aa,dbp,maxa,lbp,dkk,ils;
integer array SOS[1:11];
boolean main;
drumplace:=1;
fromdrum(11,SOS[1]);
db:=SOS[1];
if db<0 then
begin
setoutput(0);
print('?bledy w pierwszym przebiegu SORRY');
stop
end if db<0;
comment tu pierwszy przebieg byl poprawny,pobieramy z SOS
informacje o lokacji na bebnie wynikow pierwszego przebiegu
          generujemy kod;
nrk:=SOS[2];
ils:=SOS[3];
main:=¬key(15);
dbp:=256;
begin
integer array BUF[1:db],BP[1:dbp],LK[0:nrk,1:2],
             LPG[1:if nrk>0 then nrk else 1,1:2],
             TS[1:if ils>0 then ils else 1,1:4];

procedure PPRINT(a);
string a;
begin
for k:=11,13 do
if key(k) then
begin
setoutput(k-10);
print(a)
end
end   procedury PPRINT;

procedure OUTINT(a);
value a;
integer a;
begin
format('1111');
for k:=11,13 do
if key(k) then
begin
setoutput(k-10);
print(a)
end
end   procedury  OUTINT;

procedure OOUTCHAR(a);
value a;
integer a;
begin
for k:=11,13 do
if key(k) then
begin
setoutput(k-10);
```

```
outchar(a)
end
end  procedury OOUTCHAR;

procedure OUTID(a);
value a;
integer a;
begin  comment
procedura wypisuje identyfikator algolowy zapisany na bebnie
                od slowa o adresie a,na wyjscia wybrane dla wynikow;
integer array BP[1:65];
integer x,y,z;
z:=drumplace;
drumplace:=a;
fromdrum(65,BP[1]);
x:=BP[1];
for y:=2 step 1 until x do
OOUTCHAR(BP[y]);
drumplace:=z;
end procedury OUTID;
procedure output(pocz,kon);
value pocz,kon;
integer pocz,kon;
begin  comment  procedura wyprowadza obszar bebna zawarty miedzy
                pocz i kon na urzadzenie nr 1;
integer i,ilb,r,j;
procedure out(a);
integer a;
if a>0 then
begin
    integer j;
fromdrum(a,BUF[1]);
for j:=1 step 1until a do
OOUTCHAR(BUF[j]);
end  procedury out;
ilb:=(kon-pocz) div db;
r:=kon-pocz-db×ilb;
drumplace:=pocz;
for j:=1 step 1 until ilb do
out(db);
out(r);
end  procedury output;

integer procedure UNPACK(n,i);
value n,i;
integer n,i;
begin  comment   procedura zwraca kod znaku zapisanego na i-tej
                od  lewej szostce bitow. i=1,2,3,4;
integer m,p;
p:=4-i;
for i:=1step 1 until p do
begin
usak(n);    pnk(+6);    pske(n)
end  for i    ;
m:=63;
usak(m);     mlka(n);
UNPACK:=n;
end procedury UNPACK;
```

```
procedure OUTARG(a);
value a;
integer a;
begin
    comment  procedura generuje instrukcje ustawiajace parametry
             argumentu lub wyniku w polu parametrow;
if a<-1024 then
begin     comment
 wypisujemy instrukcje ustawiajace w polu parametrow
                    adres zmiennej rzeczywistej o danej nazwie;
OOUTCHAR(48);
PPRINT('ref(');
OUTID(-a-1024);
PPRINT('),0,0');
end
else
if a<-512 then
begin
comment wypisujemy instrukcje ustawiajace akumulator w polu
parametrow;
OOUTCHAR(48);
OUTINT(a+512);
PPRINT(',0,0');
end.
else

begin   comment
 wypisujemy instrukcje ustawiajace macierz w polu parametrow;
a:=-a;
if a=1 then PPRINT(',16384,0,0') else
if a=2 thenPPRINT(',32786,0,0') else
begin   comment  nazwa w tablicy symboli;
PPRINT(',ref(');
lc:=TS[a,1];
for sp:=1,2,3,4 do
begin  comment  wypisanie nazwy;
n:=UNPACK(lc,sp);
n:=if n=16 or (n>0 ∧n<10) then n else n+64;
OOUTCHAR(n);
end  wypisywania nazwy;
PPRINT('[1');
if TS[a,2]=2 then
PPRINT(',1');
PPRINT('])');
comment  wypisane wywolane ref,teraz zakresy;
for sp:=3,4 do
begin
lc:=TS[a,sp];
OOUTCHAR(48);
if lc=0then OUTINT(1) else
if lc>0 then OUTINT(lc) else
OUTID(-lc-1024);
end for sp - wypisywania zakresow;
end   nazwa w tablicy symboli;
end  ustawiania macierzy;
end   procedury OUTARG;

comment sprowadzamy z bebna tresc tablic z poprzedniego
przebiegu;
LK[0,2]:=SOS[5];
drumplace:=SOS[6];  if nrk≥0 then
```

```
begin
fromdrum(nrk×2,LK[1,1]);
drumplace:=SOS[9];
fromdrum(nrk×2,LPG[1,1]);
drumplace:=SOS[10];
fromdrum(ils×4,TS[1,1]);
end;
wait(' keys ');
PPRINT('

');
if main then
PPRINT('?begin
integer a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12,a13,a14,a15,a16,a17,
a18 ;
integer procedure OPER;
begin
dokj(a12);
format('44444,');
print(a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12,a13,a14,a15,a16,a17,
a18);
line(1);
OPER:=0;
end;
procedure xmat20000set;
begin   comment inicjalizacja zmiennych opisujacych pole
pamieci roboczej    ;
usak(+18);   snz(ET);   usak(14810);
ET:  pska(a14);   pska(a15);   usak(+17);
pska(a13);  ssak(2);   rska(a15);
dokj(a13);   odkj(a14);   a16:=a14;
odkj(a16);   a17:=a15;   a18:=a13;
end   xmat20000set;

a12:=0;
        ');
for ak:=1 step 1until nrk do
begin
i:=LK[ak-1,2];
ii:=LK[ak,1];
output((if i<0 then -i else i)+1,if ii<0 then -ii else ii);
if LK[ak,1]>0 then
begin
drumplace:=LPG[ak,1];
dkk:=LPG[ak,2];
kk:=0;
PPRINT('
begin integer xmat200001,xmat200002,xmat200003;
xmat20000set;   xmat200001:=3797;   xmat200002:=0;
for xmat200003:=
');
for i:=1 while kk<dkk do
begin     comment  sprowadzamy kolejne fragmenty kodu do bufora;
lbp:=1;
m:=if dkk-kk<dbp then  dkk-kk else dbp;
fromdrum(m,BP[1]);
for ii:=1 while lbp<m do
```

```
begin   comment
 pobieramy z bufora kolejne czworki parametrow i generujemy
                ostateczny kod;
nrop:=BP[lbp];
lbp:=lbp+1;
OUTINT(nrop);
for k:=1,2,3 do
begin  comment
pobieramy numery argumentow i wyniku i generujemy instrukcje
ustawiajace pole parametrow;
l:=BP[lbp];
lbp:=lbp+1;
if l#0 then
OUTARG(l)
else
PPRINT(',0,0,0');
end   for k;
PPRINT(' ,OPER,
');
end  for ii pobierania czworek i generacji kodu;
kk:=kk+m;
end   for i - sprowadzania kolejnych fragmentow
;PPRINT('
0 do
begin dokj(xmat200002);
usak(xmat200003); mok1(xmat200001); pska(+xmat200002);
if xmat200002=11 then xmat200002:=0   end
end    ')
end
end
end  for ak - petli po ilosci krysztalow
          wyprowadzamy jeszcze fragment programu od ostatniego
        krysztalu do konca programu;

i:=LK[nrk,2];
output((if i<0 then -i else i)+1,SOS[7]-4);
if main then
PPRINT('?end?');
OOUTCHAR(68);
PPRINT('                                    ?');
end  integer array
;  setoutput(0);
print('?program wygenerowany?');
end?
```