

obliczeniem wartości wyrażenia  $a_{3-a+1}$  i wyperforowaniem obliczonej wartości według tegoż co poprzednio wzorca.

W p r z y k ł a d z i e 3 mamy do czynienia z alternatywnym wyrażeniem łańcuchowym. W zależności od tego, czy aktualna wartość zmiennej  $c$  jest większa od aktualnej wartości zmiennej  $d$  czy nie, wartość wyrażenia  $c_{2+d}2$  po jej obliczeniu zostaje wyperforowana według wzorca  $\langle dd.ddd \rangle$  lub  $\langle dddd \rangle$ .

W p r z y k ł a d z i e 4 występują dwie zmienne:  $a$  i  $b$ . Zgodnie z podanymi poprzednio regułami widzimy, że  $a$  musi być parametrem formalnym jakiejś procedury, której ciało zawiera daną instrukcję output. Parametr ten musi być kategorii string i w momencie wywołania procedury na jego miejsce musi być podstawiony parametr aktualny w postaci wyrażenia łańcuchowego, pozwalającego określić konkretny wzorzec jako wartość tego parametru. Zmienna  $b$  występująca w danej instrukcji output jest oczywiście zmienną liczbową, której wartość będzie wyperforowana według wzorca, jaki otrzymamy po podstawieniu na parametr  $a$  parametru aktualnego.

P r z y k ł a d 5 jest podobny do poprzedniego, z tym że występuje tu alternatywa zarówno dla parametru formalnego kategorii string, jak i dla zmiennej liczbowej. Zgodnie z regułami  $A$  i  $D$  muszą być zmiennymi booleowskimi,  $B$  i  $C$  parametrami formalnymi kategorii string, a  $E$  i  $F$  zmiennymi liczbowymi. Gdy obie zmienne  $A$  i  $D$  mają wartość true, cała instrukcja jest równoznaczna z instrukcją output  $(B,E)$ , a więc z instrukcją w postaci rozpatrzonej w poprzednim przykładzie. Gdy  $A$  ma wartość true, a  $D$  false, cała instrukcja jest równoznaczna z instrukcją output  $(B,F)$ . Gdy  $A$  ma wartość false, a  $D$  true, cała instrukcja jest równoznaczna z output  $(C,E)$ . Wreszcie gdy obie zmienne  $A$  i  $D$  mają wartość false, cała instrukcja jest równoznaczna z output  $(C,F)$ .

P r z y k ł a d 6 jest podobny do przykładu 2. Pewne wątpliwości może budzić fakt, że jednym z parametrów danej instrukcji output jest inna instrukcja output i powstaje konkurencja wzorców. Otóż wzorzec  $\langle -ddd \rangle$  podany w wewnętrznej instrukcji output obowiązuje tylko w granicach tej instrukcji, w danym razie tylko dla perforacji wartości zmiennej  $m$ . Na zewnątrz tej instrukcji obowiązuje wzorzec instrukcji zewnętrznej, tj.  $\langle +d.ddd_m+d \rangle$ , zarówno dla perforacji wartości wyrażenia  $X+Y$ , jak i  $u-\sin(v)$ .

Na zakończenie niniejszego paragrafu zwróćmy jeszcze uwagę na fakt, że nie można drukować ani perforować bezpośrednio po sobie dwu liczb, gdyż spowodowałoby to zlanie się ich w jedną całość. Między dwiema drukowanymi czy perforowanymi liczbami muszą być umieszczone jakieś odstępy, przecinki czy teksty rozdzielające. Do tego będą służyły niektóre z dalszych procedur.

### 5.11. Zadanie

Rozwiązać zadanie 3.12-a, traktując zmienną  $X$  jako lokalną i zmieniając podany poprzednio program wykorzystując in-

strukcję "dla". Końcowa wartość zmiennej  $X$  ma być wyperforowana w formie liczby dziesiętnej z pięcioma cyframi po kropce a jedną przed kropką, z ewentualną perforacją zera przed kropką, z perforacją zarówno znaku plus, jak i minus.

### 5.12. Procedura write

Procedura write (czytaj: rajt) jest analogiczna do output z tym że nazwa "output" jest zastąpiona przez "write", a wyniki nie są perforowane na taśmie lecz drukowane przez maszynę do pisania. Zarówno w instrukcji output, jak i write parametrami na liście mogą być wszystkie instrukcje typu out- i write-. Na przykład analogicznie do instrukcji podanej w przykładzie 6 poprzedniego paragrafu możemy mieć:

```
write(†+d.ddd†, †, X+Y, outsp(4), output(†-ddd†,m), writcr,
      u-sin(v))
```

### 5.13. Zadanie

Rozwiązać zadanie 3.12-b, traktując zmienną  $X$  jako lokalną i zmieniając podany poprzednio program wykorzystując instrukcję "dla". Końcowa wartość zmiennej  $X$  ma być wydrukowana na maszynie do pisania w formie liczby zmiennoprzecinkowej z mantysą posiadającą jedną cyfrę znaczącą przed kropką a pięć po kropce, z cechą jednocyfrową, z drukiem jedynie minusa przed mantysą, a z drukiem zarówno plusa jak i minusa dla cechy.

### 5.14. Procedura outtext

Wywołanie procedury outtext (czytaj: auttekst) ma postać następującej instrukcji:

```
outtext(lista parametrów)
```

z tym, że parametrami mogą być albo wyrażenia łańcuchowe, których wartościami są teksty ujęte w nawiasy  $\langle \langle i \rangle \rangle$ , albo dowolne instrukcje wyjścia. W szczególności wyrażenie łańcuchowe może redukować się do tekstu. Pierwszym parametrem nie musi być wyrażenie łańcuchowe, może być nim również dowolna instrukcja wyjścia. Jak w procedurach output i write, tak i tutaj, parametry można oddzielać albo przecinkami, albo ciągami znaków o następującej budowie:

```
)nazwa złożona z liter:(
```

Działanie procedury outtext jest następujące:

- parametry są uwzględniane według ich kolejności na danej liście: jeżeli parametrem jest tekst, zostaje on wyperforowany (z pominięciem nawiasów  $\langle \langle i \rangle \rangle$  na taśmie papiero-

wej); jeżeli parametrem jest wyrażenie łańcuchowe, najpierw ustala się jego wartość, którą musi być zawsze jakiś tekst, i następnie tekst ten zostaje wyperforowany jak wyżej; jeżeli parametrem jest wreszcie jakaś instrukcja wyjścia, zostaje ona wykonana, po czym przechodzi się do następnego parametru na liście aż do jej wyczerpania;

- znaki , i odstępy są w tekstach równoważne i perforowane jako kod SPACE, natomiast powroty karetki są perforowane jako kod CAR RET, o czym była już mowa w paragrafie 5.8.

Oto przykłady wywołania procedury outtext:

- 1) outtext(<<Wyniki:.,.,x=.,>,output(<d.dd>,x),  
<.,.,.,y=.,>,output(<-.dddd>,y),<.,.,>)
- 2) outtext(nazwa,<<Funkcja,f(x):  
.,.,x=.,.,.,0.1.,.,0.2.,.,0.3.,.,0.4.,.,0.5>)

Wykonanie instrukcji outtext podanej w przykładzie 1 przebiega następująco:

- zostaje wyperforowany tekst

Wyniki:.,.,x=.

- zostaje wyperforowana aktualna wartość zmiennej x według wzorca <d.dd>,

- zostaje wyperforowany tekst

.,.,.,y=.

- zostaje wyperforowana aktualna wartość zmiennej y według wzorca <-.dddd>,

- na zakończenie zostają wyperforowane cztery kody odstępów.

Wykonanie instrukcji outtext podanej w przykładzie 2 przebiega następująco:

- zostaje wyperforowany tekst stanowiący aktualną wartość parametru formalnego o nazwie "nazwa" (parametr "nazwa" jest tu parametrem kategorii string jakiejś procedury, w której ciele występuje dana instrukcja outtext), z pominięciem - jak zwykle - nawiasów << i >,

- zostaje wyperforowany tekst

Funkcja,f(x):

potem kod powrotu karetki (ponieważ tekst występujący tu jako parametr danej instrukcji outtext zawiera po "Funkcja,f(x):" powrót karetki), a następnie dalszy ciąg tekstu aż do "0.5".

Należy zapamiętać, że procedura outtext zakłada wejście do niej w sytuacji LOWER CASE i w tejże sytuacji następuje zakończenie jej wykonania.

### 5.15. Zadanie

Jak będzie wyglądał odcinek taśmy papierowej wyperforowanej na skutek wykonania instrukcji

```
outtext(⟨x1=1⟩, output(⟨dd⟩,13), ⟨<
...y1=1⟩, output(⟨-d⟩,-5))
```

i jak będzie wyglądał przedruk tego odcinka taśmy papierowej na flexowriterze?

### 5.16. Procedura writetext

Procedura writetext (czytaj: rajttekst) jest analogiczna do outtext, z tym, że nazwa "outtext" jest zastąpiona przez "writetext", a teksty nie są perforowane na taśmie, lecz drukowane na maszynie do pisania. Znaki, i odstęp są tu równie równoważne i drukowane jako odstęp, natomiast powroty karetki występujące w tekstach, stanowiących parametry procedury writetext, powodują powroty karetki na maszynie do pisania.

A oto przykład instrukcji writetext:

```
writetext(⟨Wyniki: ...x=1⟩, write(⟨d.dd⟩,x),
output(⟨-.dddd⟩,y), ⟨<...⟩)
```

Jej wykonanie przebiega następująco:  
Na maszynie do pisania zostaje napisany tekst

Wyniki: x=

z jednym odstępem po znaku =, po czym zostaje wydrukowana aktualna wartość zmiennej x według wzorca ⟨d.dd⟩, następnie zostaje wyperforowana na perforatorze wartość zmiennej y według wzorca ⟨-.dddd⟩ i całość kończy się wykonaniem na maszynie do pisania czterech odstępów.

### 5.17. Zadanie

Jaki tekst wypisze maszyna wykonując instrukcję

```
writetext(write(⟨-n.dddd10+d⟩, 2xa-3xb, (writetext(⟨<...⟩),
a/2-b/2-105), ⟨<...wartosci_kontrolne
Suma1=1⟩, write(⟨dddd⟩,a+b), ⟨<
.....KONIEC...DRUKU⟩)
```

przy założeniu, że karetki maszyny do pisania w chwili rozpoczęcia druku znajduje się w położeniu początkowym, a zmienne a i b mają wtedy wartości: a=583.4, b=406.9 ?

### 5.18. Procedura outsp

Wywołanie procedury outsp (skrót od outspace, czytaj: autspejs) ma postać następującej instrukcji:

outsp(wyrażenie arytmetyczne)

Działanie takiej instrukcji jest następujące:

- zostaje obliczona wartość wyrażenia arytmetycznego stanowiącego aktualny parametr danej instrukcji outsp,
- na perforatorze zostaje tyle razy wyperforowany kod odstępu (SPACE), ile wynosi obliczona wartość parametru (zaokrąglona do najbliższej liczby całkowitej).

Jeśli wartość parametru nie będzie dodatnia, nie zostaje wyperforowany żaden znak. Przykład instrukcji outsp:

outsp(n+m-7)

Należy zapamiętać, że procedura outsp nie ma odpowiednika pomiędzy procedurami typu write-.

### 5.19. Zadanie

Zaprojektować instrukcję wyjściową, która spowodowałaby wyperforowanie odcinka taśmy papierowej z zakodowanym następującym tekstem:

X,Y,Z =     -1.63759             -5.44932             -8.28672

przy założeniu, że podane tu wartości są aktualnymi wartościami zmiennych X,Y i Z.

### 5.20. Procedura outcr

Procedura outcr (skrót od out-carriage-return; czytaj: autkeridż-ritern) jest procedurą bezparametrową. Jej wywołanie ma postać instrukcji

outcr

a działanie polega na wyperforowaniu na perforatorze na taśmie papierowej kodu CAR RET. Należy pamiętać, że w GIER-ALGOLu powrót karetki oznacza również i równoczesne przejście do nowego wiersza.

### 5.21. Zadanie

Zaprojektować instrukcję wyjściową, która spowodowałaby wyperforowanie odcinka taśmy papierowej z zakodowanym następującym tekstem

U1	12	U2	12	U3	12	U4	12	U5
6447	10	8215	10	3039	10	1888	10	4501

przy założeniu, że podane tu wartości są aktualnymi wartościami zmiennych wymienionych w nagłówku.

Dla ułatwienia podano liczby odstępów między liczbami danymi.

### 5.22. Procedura writecr

Procedura writecr (skrót od write-carriage-return; czytają: rajt-keridż-ritern) jest procedurą bezparametrową, której wywołanie ma postać instrukcji

writecr

a działanie polega na wykonaniu przez maszynę do pisania powrotu karetki wraz z przejściem do nowego wiersza.

### 5.23. Zadanie

Zaprojektować instrukcję wyjściową, która spowodowałaby wypisanie automatyczne na maszynie do pisania aktualnych wartości zmiennych p,q,r,s w następującym szyku:

-3.643<sub>10</sub>-4

+0.012<sub>10</sub>-9

+8.496<sub>10</sub> 2

-2.505<sub>10</sub> 1

### 5.24. Procedury outclear i outsum. Kontrola wyperforowanej przez maszynę taśmy

Procedura outclear (czytają: autkliar) jest procedurą bezparametrową, której wywołanie ma postać instrukcji

outclear

Procedura ta służy do przygotowania maszyny do mającego nastąpić liczenia sumy kontrolnej, o której już była mowa w paragrafie 5.9, i ma podwójne działanie: z jednej strony zeruje sumę kontrolną wewnątrz maszyny, co jest konieczne dla rozpoczęcia liczenia nowej sumy kontrolnej podczas perforowania taśmy, z drugiej strony perforuje na perforatorze na taśmie papierowej kod CLEAR CODE. Jeśli w przyszłości ta-

śmę taką będzie się wczytywać do maszyny, napotkany w czasie czytania kod CLEAR CODE spowoduje wyzerowanie się sumy kontrolnej w maszynie i tym samym przygotowanie jej do kontrolnego liczenia sumy kontrolnej podczas czytania taśmy.

Procedura outsum (czytaj: autsam) jest procedurą bezparametrową, której wywołanie ma postać instrukcji

outsum

Procedura ta służy do zakończenia liczenia przez maszynę żądanej sumy kontrolnej i ma następujące działanie:

- powoduje wyperforowanie na perforatorze na taśmie papierowej kodu STOP CODE a następnie kodu SUM CODE,
- po wyżej wymienionych kodach zostaje wyperforowany kod będący funkcją aktualnej sumy kontrolnej w maszynie (charakter tej funkcji nie został przez firmę GIER ujawniony).

Jeśli w przyszłości taśma tak sporządzona będzie wczytywana z powrotem do maszyny, suma kontrolna będzie w czasie czytania przez maszynę ponownie liczona poczynawszy od kodu CLEAR CODE i po dojściu do kodów STOP CODE i SUM CODE porównana z następującym po SUM CODE kodem sumy kontrolnej obliczonej podczas perforacji. Jeśli obie sumy kontrolne będą zgodne (tzn. będą zgodne wartości wspomnianej wyżej funkcji sumy kontrolnej), maszyna przechodzi do dalszego wykonywania programu. Jeśli znajdzie niezgodność wspomnianych sum kontrolnych, maszyna do pisania drukuje tekst

sum fails

(czytaj: sam fejls, co znaczy: suma szwankuje) i cała maszyna zatrzymuje się. Po przyciśnięciu jakiegokolwiek klawisza na maszynie do pisania, maszyna kontynuuje czytanie taśmy perforowanej.

#### 5.25. Procedura outchar

Procedura outchar (skrót od out-character; czytaj: autkar, autkarakter) jest procedurą jednoparametrową i jej wywołanie ma postać instrukcji

outchar(wyrażenie arytmetyczne)

Działanie procedury polega na wyperforowaniu na taśmie papierowej na perforatorze takiego kodu, który odpowiada wartości liczbowej wyrażenia stanowiącego parametr aktualny danej instrukcji outchar. Tabela liczb i odpowiadających im symboli i kodów była podana w paragrafie 5.8. Jeżeli wartość aktualnego parametru nie jest całkowita, zostaje zaokrąglona do najbliższej liczby całkowitej. Jeśli wartość ta jest większa niż 127, zostaje wzięta modulo 128, tzn. zamiast niej bierze się resztę z podzielenia jej przez 128. Jeśli np. wartość aktualnego parametru byłaby 530, to zostałaby wyperforowany kod odpowiadający liczbie 18 czyli symbolom s lub S, ponieważ  $530 = 4 \times 128 + 18$ .

Należy tu zapamiętać, że procedurą outchar można otrzymywać także kody nieodpowiadające żadnemu z używanych symboli. Jak o tym już była mowa w paragrafie 5.8, do takich nale-



zą kody odpowiadające liczbom 10, 15, 26, 42, 45, 46, 47 oraz liczbom od 65 do 127 włącznie.

Dla ustalenia odpowiedniości między liczbami i ich kodami możemy posługiwać się metodą opisaną w paragrafie 5.8, albo tabelką podaną w tymże paragrafie.

Należy zapamiętać, że procedura outchar może popsuć kontrolę sumową, mianowicie w przypadku instrukcji

outchar(28)      albo      outchar(61),

które powodują perforację kodów CLEAR CODE albo SUM CODE.

Używając procedury outchar, trzeba również uważać na perforowanie, gdzie należy, kodów LOWER CASE i UPPER CASE, ażeby zapewnić sobie przy późniejszym wykorzystaniu wyperforowanej taśmy druk poprawnych symboli. Należy tu pamiętać, że procedury output, write, outtext i writetext zakładają wejście do nich w sytuacji LOWER CASE i w tejże sytuacji następuje zakończenie ich wykonania.

A oto przykłady instrukcji outchar:

- 1) outchar(30)
- 2) outchar(2*x*1-*j*)
- 3) outchar(if *c*>0 then 0 else 32)

W p r z y k ł a d z i e 1 mamy do czynienia z instrukcją, której wykonanie spowoduje wyperforowanie na taśmie papierowej na perforatorze kodu symbolu TAB, tzn. tabulatora.

W p r z y k ł a d z i e 2 parametrem instrukcji outchar jest wyrażenie arytmetyczne, którego wartość jest najpierw obliczana. Zakładając na przykład, że w chwili wywołania danej instrukcji outchar mamy aktualne wartości *i*=5 i *j*=2, widzimy, że dana instrukcja jest wtedy równoznaczna z outchar(8) i spowoduje perforację kodu cyfry 8 (lub co na jedno wychodzi - okrągłego nawiasu otwierającego).

W p r z y k ł a d z i e 3 parametrem instrukcji outchar jest alternatywne wyrażenie arytmetyczne. Jeżeli w chwili wywołania danej instrukcji pewna zmienna *c* ma wartość dodatnią, to dana instrukcja jest równoznaczna z outchar(0) i spowoduje wyperforowanie kodu odstępu. Jeżeli natomiast *c* ma wartość niedodatnią, dana instrukcja jest równoznaczna z outchar(32) i spowoduje wyperforowanie kodu minusa.

## 5.26. Zadanie

Rozwiązać zadanie 5.21, używając tabulatora za pośrednictwem procedury outchar, zakładając, że koniki tabulatora przy późniejszym przedruku taśmy papierowej będą ustawione na pierwsze cyfry drukowanych liczb (opis działania tabulatora podano w paragrafie 5.7).



5.27. Zadanie

Zastąpić instrukcję

outtext( $\langle X, \_ \rangle$ )

kilkoma instrukcjami outchar.

5.28. Procedura writechar

Procedura writechar (skrót od write-character; czytaj: rajt-kar, rajt-karakter) jest analogiczną do outchar, z tym że odpowiednie symbole są drukowane na maszynie do pisania. Wynika stąd, że dla niektórych liczb, dla których nie ma odpowiedniego symbolu, nie będzie wydrukowany żaden znak. Są to liczby: 10, 15, 26, 42, 45, 46, 47 oraz liczby od 65 do 127 włącznie.

5.29. Zadanie

1) Zaprojektować instrukcję, która by w przypadku, gdy pewna zmienna typu integer  $X$  jest podzielna przez 5, powodowała powrót karetki na maszynie do pisania, a w przypadku, gdy  $X$  nie jest podzielne przez 5, przeskok karetki do najbliższej kolumny zaznaczonej konikiem tabulatora.

2) Zaprojektować instrukcję, która nastawiłaby maszynę do pisania na druk czerwony.

5.30. Przykład programu z wykorzystaniem instrukcji wyjścia

Należy ułożyć program na stabilizowanie wartości funkcji

$$\psi(u) = \frac{\sqrt{u^2 + 1} - u}{\sqrt{u^2 + 1} + u} \quad (0 < \psi(u) \leq 1)$$

z dokładnością do 5 znaków dziesiętnych po kropce, z drukowaniem zera przed kropką, dla  $u = 0, 0.2, 0.4, \dots, 99.8$ , z drukiem wartości funkcji  $\psi(u)$  w 5 kolumnach na maszynie do pisania podłączonej bezpośrednio do maszyny cyfrowej, z tym że w pierwszej kolumnie byłyby wartości tej funkcji dla  $u$  całkowitych. Tablice powinny mieć nagłówki:

Tablice funkcji Psi

powinny mieć ponadto w nagłówku wiersz

u	.0	.2	.4	.6	.8
---	----	----	----	----	----

dający nagłówki poszczególnym kolumnom. Po lewej stronie powinna być jeszcze wydrukowana kolumna wartości całkowitych  $u$ . Całość powinna zmieścić się na arkuszu o szerokości 65 znaków drukarskich.

Z powyższych danych wnioskujemy, że na odstępy możemy poświęcić

$$65 - 2 - 5 \times 7 = 28$$

miejsc, ponieważ po doliczeniu kropki i zera przed kropką musimy przewidzieć 7 miejsc na druk jednej wartości funkcji  $\Psi$ , a ponadto dwa miejsca na druk wartości całkowitej  $u$ . Wobec tego przyjmujemy między kolumną wartości  $u$  i pierwszą kolumną wartości  $\Psi$  6 odstępów, między kolumnami wartości  $\Psi$  po 5 odstępów, a pozostałe 2 odstępy wykorzystamy robiąc po 1 odstępie przed kolumną wartości  $u$  i po ostatniej kolumnie.

Pozostaje jeszcze sprawa odstępów w nagłówku. Tekst "Tablice funkcji  $\Psi$ " zawiera 17 liter. Jeśli między wyrazami tego tekstu umieścimy po dwa odstępy, cały tekst zajmie 21 miejsc. Pozostałe  $65 - 21 = 44$  miejsca dzielimy na połowę i po 22 odstępów damy przed i po wymienionym tekście.

Między powyższym tekstem a wierszem z nagłówkami kolumn, damy jeden wiersz pusty dla lepszego wyglądu. Podobnie tytuł "Tablice funkcji  $\Psi$ " poprzedzimy kilkoma pustymi wierszami.

Aby nagłówki kolumn wypadły pośrodku odpowiednich kolumn, przyjmujemy dla wiersza z tymi nagłówkami następujący układ:

u.....0.....2.....4.....6.....8.  
10 odstępów 10 odstępów 10 odstępów 10 odstępów 10 odstępów

A oto żądany program:

```
begin comment Tablice wartosci funkcji Psi;
real w,a,b; integer u;
writcr; writcr; writcr;
writetext(⟨<.....Tablice..funkcji..Psi
u.....0.....2.....4.....6.....8
>);
for u:= 0 step 1 until 99 do begin write(⟨ddd>,u);
                                writetext(⟨<..>);
for w:= 0, 0.2, 0.4, 0.6, 0.8 do
begin writetext(⟨<.....>); a:= u+w; b:= sqrt(4*a+1);
write(⟨n.ddddd>,(b-a)/(b+a)) end; writcr end;
writcr; writcr; writcr end;
```

### 5.31. Zadanie

Napisać program dla przykładu podanego w poprzednim paragrafie, wprowadzając następujące zmiany:

- tablice mają być wyperforowane na taśmie papierowej na perforatorze,
- wiersze z wartościami funkcji Psi mają być grupowane po 5, tzn. po co piątym wierszu ma być dodany jeden wiersz pusty jako odstęp między wierszami.

### 5.32. Procedury wejścia

Podobnie jak wyprowadzanie wyników za pomocą instrukcji wyjścia, tak również wprowadzanie danych jest możliwe w każdym miejscu programu. Jest to bez wątpienia duże ułatwienie dla programisty.

Dane mogą być wprowadzane do maszyny albo drogą czytania perforowanej 8-kanalowej taśmy papierowej przez czytnik, podłączony bezpośrednio do maszyny albo drogą wypisywania ich na dany sygnał na maszynie do pisania, również podłączonej bezpośrednio do maszyny cyfrowej. Należy pamiętać, że czytanie perforowanej taśmy papierowej przez czytnik wymaga założenia odpowiedniej taśmy. Jeśli na czytniku w chwili wywołania danej procedury nie będzie żadnej taśmy, maszyna zatrzyma się i będzie czekała na jej założenie, po czym sama rozpocznie czytanie. Sygnałem do rozpoczęcia wprowadzania danych przez pisanie na maszynie do pisania jest zapalenie się zielonego światełka na jej pulpicie. Po zapaleniu tego światełka maszyna zatrzymuje się i czeka na dane z maszyny do pisania.

Wszystkie procedury wejścia (jak również procedury wejściowo-wyjściowe, o których będzie mowa później) mają pewne cechy wspólne, które omówimy w niniejszym paragrafie.

Wszystkie kody zawarte między kodami PUNCH OFF i PUNCH ON (z tymi dwoma włącznie) są w czasie wprowadzania danych całkowicie ignorowane przez maszynę. Ignorowane są również:

- kod TAPE FEED,
- rzędkie nie zawierające prócz dziurek prowadzących żadnych innych,
- rzędkie zawierające wszystkie możliwe dziurki tzn. następujące trzy rodzaje rzędków:

: 0000:000 : : : : 00000:000 : :	TAPE FEED  BLANK TAPE czyli pusta taśma  ALL HOLES czyli wszystkie dziurki
--	--

O kodzie TAPE FEED była już mowa w paragrafie 5.8. Należy tu dodać, że kod TAPE FEED jest ignorowany nie tylko przez czytnik podłączony do maszyny cyfrowej, ale również przez czytnik flexowriter'a, tak samo jak BLANK TAPE i ALL HOLES (czytaj: blank tejp, ol hoils). Pustego kodu tzn. BLANK TAPE nie można otrzymać na flexowriterze, natomiast można otrzymać na perforatorze przez przyciśnięcie odpowiedniego kłucza.

Wszystkie dziurki czyli ALL HOLES, można otrzymać na flexowriterze przyciskając równocześnie klawisze TAPE FEED i PUNCH ADRES.

Pojawienie się przy czytaniu rzędka o parzystej liczbie dziurek powoduje zatrzymanie się maszyny, co stanowi dobry sposób kontroli poprawności taśmy (wyjątek stanowią tu kody BLANK TAPE i ALL HOLES, ale tylko dlatego, że są one ignorowane). Samo pojawienie się rzędka o parzystej liczbie dziurek i zatrzymanie maszyny nie powoduje popsucia programu w maszynie. Wystarczy poprawić nieprawidłowy rząderek i odpowiednio założyć na czytnik, by czytanie biegło poprawnie dalej.

W czasie czytania taśmy papierowej odbywa się automatycznie kontrola sumowa, o czym była już mowa w paragrafie 5.24.

Niektóre procedury wejścia powodują wprowadzenie do maszyny kodu jednego tylko symbolu, inne powodują wprowadzenie całych liczb zapisanych ciągami symboli. W takich przypadkach musimy pamiętać o daniu maszynie sygnału, kiedy kończy się zapis danej liczby. Odbywa się to w ten sposób, że póki maszyna otrzymuje kody cyfr lub kody znaków

+ - . x

zalicza je do zapisu danej liczby. Sygnałem zakończenia staje się pojawienie tzw. t e r m i n a t o r a, tj. litery albo jednego z następujących znaków:

$\vee \times / = ; [ ] ( ) | \wedge < > , : \text{ TAB PUNCH ON CAR RET STOP CODE}$

Jako terminatory mogą również służyć kody nie odpowiadające żadnym symbolom, jak na przykład kod odpowiadający liczbie 15, który można otrzymać na flexowriterze przyciskając równocześnie 3 i AUX CODE, czy kod odpowiadający liczbie 46, który można otrzymać na flexowriterze przyciskając równocześnie k i AUX CODE.

Pojawienie się natomiast znaków

SPACE —

jest przez translator ignorowane. Oba te ostatnie znaki nazywamy w tym przypadku symbolami pustymi. Mogą się one zatem przeplatać zarówno ze znakami informacyjnymi, tj. cyframi oraz + - <sub>10</sub>. jak również z terminatorami. Między poszczególnymi liczbami możemy umieszczać dowolną ilość terminatorów. Będą one w czasie wczytywania do maszyny ignorowane przez translator (z wyjątkiem pierwszego z nich, który — jak to już było powiedziane — sygnalizuje jedynie zakończenie zapisu liczby). Fakt ten wykorzystujemy często dla komentowania zapisanych liczb. Zamiast np. wyperforować na taśmie ciąg kodów odpowiadający ciągowi znaków

12.357, 18.632, 25.898,

wolimy nieraz wyperforować ciąg kodów, odpowiadający ciągowi znaków

a:= 12.357, b:= 18.632, c:= 25.898,

albo

dlugosc:= 12.357, szerokosc:= 18.632, wysokosc:= 25.898,

ponieważ dla maszyny wszystkie te trzy warianty są zupełnie jednakowe, natomiast przedrukowując dla kontroli treść perforowanej taśmy na flexowriterze otrzymujemy wszystkie symbole,

co pozwala na bardzo łatwą orientację w interpretacji poszczególnych liczb. Jest to szczególnie ważne w przypadkach wprowadzania do maszyny dużych ilości danych. W przypadku braku odpowiednich komentarzy możliwość omyłki byłaby duża.

Należy jeszcze zauważyć, że prócz możliwości dołączenia komentarzy mamy jeszcze możliwość podkreślania zarówno terminatorów jak i symboli informacyjnych, ponieważ podkreślenie należy w czasie wprowadzania danych do symboli pustych. Na przykład możemy niektóre z wprowadzanych liczb podkreślić dla ich wyróżnienia. Takie podkreślenia będą przez translator ignorowane na równi z odstępami.

### 5.33. Procedura input

Wywołanie procedury input (czytaj: input) ma postać następującej instrukcji:

input(lista parametrów)

z tym, że:

- parametrami mogą być zmienne liczbowe albo nazwy tablic (array),
- parametry mogą być oddzielane (podobnie jak to było w instrukcjach output) przecinkami albo ciągami znaków

)nazwa złożona z samych liter:(

Wykonanie takiej instrukcji polega na kolejnym odczytywaniu liczb z taśmy perforowanej, założonej na czytnik maszyny i podstawianiu ich jako wartości na kolejne parametry z listy podanej w danej instrukcji wejścia. Gdy takim parametrem jest nazwa tablicy, następuje tu podstawianie kolejnych czytanych wartości na wszystkie wyrazy tej tablicy według kolejności, którą można opisać następująco:

Jeśli parametrem w danej instrukcji wejścia jest A, gdzie A jest nazwą tablicy uprzednio zadeklarowanej jako

array A[i1:u1,i2:u2,...,in:un]

to podstawianie wartości odbywa się zgodnie z programem:

for i1:= i1 step 1 until u1 do

for i2:= i2 step 1 until u2 do

.....

for in:= in step 1 until un do

A[i1,i2,...,in]:= kolejno odczytana z taśmy liczba

gdzie i1, i2,..., in były zmiennymi roboczymi.

Z chwilą podstawiania wartości na jakikolwiek parametr, podstawienie wartości na parametr poprzedni jest już w pełni zakończone, co możemy wykorzystywać przy dalszych podstawieniach. Jeśli np. mamy wykonać instrukcję

input(p, M[p,4])



to najpierw zostanie podstawiona wartość na  $p$  i ta wartość dopiero pozwala określić zmienną  $M[p,4]$ , na którą ma być podstawiona wartość następna odczytana z taśmy.

Z powyższego wynika, że jeżeli w czasie wykonywania instrukcji `input` napotkamy na parametr będący nazwą tablicy, to przejście do następnego parametru będzie możliwe dopiero po zakończeniu podstawiania wartości dla wszystkich wyrazów tej tablicy. Jeśli zatem chcielibyśmy ustalić ilość liczb na taśmie papierowej, potrzebnych do wykonania danej instrukcji `input`, musielibyśmy podzielić jej parametry na dwie grupy: I. parametry będące zmiennymi liczbowymi, II. parametry będące nazwami tablic, i do ilości parametrów w I grupie dodać ilości wyrazów wszystkich tablic wymienionych w grupie II.

Dla przypomnienia zwracamy tu uwagę, że w liście parametrów nie należy mylić zmiennych indeksowanych z nazwami tablic. Te ostatnie bowiem nie różnią się zapisem od zmiennych bez indeksów, a jedynie uprzednie deklaracje decydują o tym, czy dany parametr jest nazwą zmiennej czy tablicy.

Przykłady instrukcji `input`:

- 1) `input(B)`
- 2) `input(XYZ,C[p,q,r])`
- 3) `input(p,q,r,C[p,q,r])`

W przykładzie 1 mamy do czynienia z jednym tylko parametrem  $B$ . Jeśli był on uprzednio zadeklarowany jako zmienna liczbowa, tzn. zmienna typu real albo integer, to wykonanie danej instrukcji `input` będzie polegało na odczytaniu przez czytnik jednej tylko liczby z taśmy perforowanej i podstawieniu jej na zmienną  $B$ . Jeśli natomiast  $B$  było uprzednio zadeklarowane jako nazwa tablicy, z taśmy odczytuje się tyle liczb, ile wyrazów ma tablica  $B$ , i sukcesywnie podstawia je na kolejne wyrazy tej tablicy, według zdefiniowanej wyżej kolejności.

W przykładzie 2 instrukcja `input` zawiera dwa parametry, z których drugi jako indeksowany nie może być nazwą tablicy, a tylko zmienną liczbową. Pierwszy parametr  $XYZ$  może być zarówno nazwą zmiennej, jak i tablicy, o czym muszą rozstrzygać uprzednie deklaracje.

W przykładzie 3 mamy do czynienia z czterema parametrami, z których wszystkie są zmiennymi. Pierwsze trzy bowiem określają indeksy dla czwartego, a więc nie mogą być nazwami tablic, a czwarty jest indeksowany, więc też jest nazwą zmiennej. Wykonanie danej instrukcji `input` będzie zatem polegało na odczytaniu przez czytnik maszyny jednej liczby z taśmy papierowej i podstawieniu jej na zmienną  $p$ , potem na odczytaniu drugiej liczby i podstawieniu jej na zmienną  $q$ , potem na odczytaniu trzeciej liczby i podstawieniu jej na zmienną  $r$ , wreszcie odczytaniu czwartej liczby i podstawieniu jej na zmienną  $C[p,q,r]$ , której indeksy zostały określone przez poprzednie trzy podstawienia.

Jeżeli ciąg symboli informacyjnych dla kolejnej czytanej liczby składa się z samych minusów (nie licząc oczywiście symboli pustych), to powoduje to pozostawienie dla odpowiedniej zmiennej jej dotychczasowej wartości i przejście do następnej zmiennej (tzn. ewentualnie do następnego wyrazu tablicy) i do następnej liczby na taśmie. Możliwość takiego ciągu

gu jest bardzo dogodna nie tylko dlatego, że zamiast wielu symboli potrzebnych dla powtórzenia danej wartości danej zmiennej można użyć tylko jednego minusa, ale również dlatego, że wartość którą danej zmiennej chcemy pozostawić, może być wynikiem pewnych obliczeń i nie być nam znana. Powyższa możliwość użycia ciągu samych minusów istnieje tylko dla procedury input i nie ma jej w innych procedurach wejścia.

A oto przykłady ciągów symboli służących do zapisu liczby wraz z terminatorem kończącym zapis:

- |                                |                                      |
|--------------------------------|--------------------------------------|
| 1) 456 789,                    | 5) liczba rownan n= 36,              |
| 2) a:= +2.63 <sub>10</sub> -3; | 6) pi:= 3.141 592 65;                |
| 3) <u>AA = 1687;</u>           | 7) C[12,27]                          |
| 4) M[j,k]= -/                  | 8) funkcja(-.56934 <sub>10</sub> +2) |

W przykładzie 1 cyfry są oddzielone odstępem, który w czasie czytania przez maszynę zostaje zignorowany. Terminatorem kończącym zapis liczby jest przecinek.

W przykładzie 2 ciąg znaków a:= służy do skomentowania, na jaką zmienną będzie podstawiana wartość +2.63<sub>10</sub>-3. Komentarz ten będzie zignorowany w czasie czytania. Terminatorem kończącym jest średnik.

W przykładzie 3 ciąg znaków AA = jest również tylko komentarzem ignorowanym w czasie czytania. Zapis został podkreślony dla wyróżnienia. Podkreślenie to będzie w czasie czytania przez maszynę również zignorowane. Terminatorem kończącym jest dwukrópek.

W przykładzie 4 ciąg znaków M[j,k]= jest komentarzem objaśniającym, że wartość zmiennej M[j,k] ma pozostać niezmienną. Sprawia to następujący później minus. Znak jest terminatorem kończącym.

W przykładzie 5 ciąg znaków

liczba rownan n =

stanowi komentarz dla liczby 36 a przecinek jest terminatorem kończącym.

W przykładzie 6 ciąg znaków pi:= jest komentarzem, odstęp między cyframi zostaną przy wprowadzaniu do maszyny zignorowane, terminatorem kończącym jest średnik.

W przykładzie 7 mamy zapisane dwie liczby. Znaki C[ stanowią komentarz, przecinek jest terminatorem pierwszej liczby, znak ] terminatorem drugiej liczby. Cały zapis przejrzysto komentuje, jakie znaczenie mają wprowadzane liczby 12 i 27.

W przykładzie 8 - podobnie jak w przykładzie poprzednim - dowcipnie skomentowano wprowadzaną liczbę. Ciąg znaków

funkcja(

stanowi tu komentarz, a nawias) terminator kończący zapis liczby -.56934<sub>10</sub>+2.



Procedura input testuje zgodność składni zapisu liczby z uprzednio podanymi regułami. W razie wykrycia błędu, na maszynie do pisania zostaje automatycznie wydrukowany tekst

correct input value, end in LC:

albo

correct input value, end in UC:

co znaczy: popraw wartość wprowadzoną, koniec był w LOWER CASE (w drugim przypadku w UPPER CASE). Maszyna zatrzymuje się i oczekuje, aż operator wypisze poprawnie na maszynie do pisania żadaną liczbę z terminatorem kończącym, który musi być w LOWER CASE albo UPPER CASE zgodnie z podanym sygnałem (należy zwrócić uwagę, że ani LOWER CASE ani UPPER CASE same nie są terminatorami). Niespełnienie ostatniego warunku może spowodować błędną pracę maszyny.

Należy zwrócić jeszcze uwagę na to, że nie ma procedury wejścia, za pomocą której można by wprowadzać dane do maszyny analogicznie do procedury input ale z maszyny do pisania.

### 5.34. Zadanie

Wykorzystując procedurę z zadania 4.8 ułożyć program na rozwiązanie układu równań, którego współczynniki są w odpowiedniej kolejności wyperforowane na taśmie papierowej, z tym że poprzedza je na tej taśmie liczba równań danego układu. Zakładamy, że w momencie wykonania danego programu taśma powyższa jest założona na czytnik maszyny. Wyniki mają być wyperforowane na taśmie papierowej według wzorca  $\langle +n.ddddd \rangle$ . Późniejszy druk ma dać pierwiastki w jednej pionowej kolumnie. Zakładamy, że macierz współczynników nie zawiera więcej niż 600 wyrazów, tzn. mieści się w pamięci operacyjnej maszyny.

### 5.35. Zadanie

Ułożyć program na obliczenie współczynników regresji i korelacji, gdy dane obserwacyjne są wyperforowane na taśmie papierowej w następującym porządku:

liczba punktów obserwacyjnych  $m$ ,  
współrzędna  $x_1$  pierwszego punktu,  
współrzędna  $y_1$  pierwszego punktu,  
waga  $w_1$  dla pierwszego punktu,  
współrzędna  $x_2$  drugiego punktu,  
współrzędna  $y_2$  drugiego punktu,  
waga  $w_2$  dla drugiego punktu  
itd.

Mają być obliczone i wydrukowane na maszynie do pisania następujące wielkości:

Średnie arytmetyczne:

$$X = \frac{\sum_{k=1}^m w_k x_k}{\sum_{k=1}^m w_k}$$

$$Y = \frac{\sum_{k=1}^m w_k y_k}{\sum_{k=1}^m w_k}$$

Standardowe odchylenia:

$$S_x = \sqrt{\frac{\sum_{k=1}^m w_k (x_k - X)^2}{\sum_{k=1}^m w_k}}$$

$$S_y = \sqrt{\frac{\sum_{k=1}^m w_k (y_k - Y)^2}{\sum_{k=1}^m w_k}}$$

Kowariancja:

$$S_{xy}^2 = \frac{\sum_{k=1}^m w_k (x_k - X)(y_k - Y)}{\sum_{k=1}^m w_k}$$

Współczynniki regresji:

$$R_{yx} = \frac{S_{xy}^2}{S_x^2}$$

$$R_{xy} = \frac{S_{xy}^2}{S_y^2}$$

Współczynnik korelacji:

$$r = \frac{S_{xy}^2}{S_x S_y}$$

W jednym wierszu mają być wydrukowane wartości  $X$  i  $Y$ , w następnym  $S_x$ ,  $S_y$  i  $S_{xy}^2$ , a w trzecim  $R_{yx}$ ,  $R_{xy}$  i  $r$ . Wszystkie wartości mają być wydrukowane z komentarzami według wzorca  $\{<d.dddd_{10}+d\}$ . Program ma być tak ułożony, aby można było sukcesywnie wprowadzać do maszyny po jednej trójce liczb  $x_k, y_k, w_k$ , bez pamiętania w maszynie wszystkich danych obserwacyjnych, których może być bardzo dużo.

### 5.36. Procedura inone

Procedura inone (czytaj: in-lan) jest bezparametrową procedurą pseudofunkcyjną. Jej wywołanie ma postać zmiennej liczbowej o nazwie "inone", jak np. w następujących dwu przykładach instrukcji podstawienia:

- 1) Alfa := Alfa + inone
- 2) UK[inone, inone] := inone

Wykonanie procedury inone przebiega następująco:

Z chwilą gdy maszyna wykonując program napotyka zmienną o nazwie "inone", zostaje odczytana kolejna liczba z taśmy papierowej, aktualnie założonej na czytnik maszyny, i podstawiona jako wartość inone. Wartość ta jest traktowana jako wartość typu real i stanowi część składową wyrażeń arytmetycznych, jak np. w pierwszym z podanych ostatnio przykładów.

Należy zwrócić uwagę na różnicę między zmienną liczbową a wywołaniem procedury inone: na zmienną można podstawiać wartości poprzez umieszczenie jej po lewej stronie jakiejś instrukcji podstawienia, natomiast nie ma sensu umieszczenie nazwy "inone" po lewej stronie takiej instrukcji. Ponadto użycie nazwy jakiejś zmiennej wielokrotnie, o ile w międzyczasie nie ma miejsca podstawienie na nią (w formie instrukcji podstawienia) nowej wartości, powoduje stałe podstawianie na tę zmienną tej samej aktualnej wartości. Tymczasem pojawienie się wielokrotnie nazwy "inone" powoduje podstawienie na jej miejsce za każdym razem na ogół innej wartości, mianowicie za każdym razem świeżo odczytanej z taśmy (wyjątek stanowi tu przypadek, gdy na taśmie papierowej zakodowano wielokrotnie jedną i tę samą liczbę). I tak np. wykonanie instrukcji podanej w drugim z ostatnio przytoczonych przykładów wymaga przygotowania na taśmie papierowej aż 3 liczb do czytania.

Gdyby tymi liczbami były kolejno np.

2, 3, 1.83456<sub>2</sub>,

wykonanie wspomnianej instrukcji byłoby równoważne wykonaniu instrukcji

$UK[2,3] := 1.83456_{10}2;$

Zwróćmy tutaj uwagę, że ze względu na sposób wykonania przytoczonej instrukcji może być celowe przygotowanie aktualnego wycinka taśmy papierowej z danymi liczbami za pomocą właśnie ciągu znaków

$UK[2,3] := 1.83456_{10}2;$

Z takiego ciągu znaków na taśmie zostaną odczytane kolejno jedynie liczby 2, 3 i  $1.83456_{10}2$ . Początkowe terminatory  $UK$  [ zostaną zignorowane, przecinek po 2 będzie terminatorem kończącym zapis tej liczby, znak ] będzie terminatorem kończącym zapis liczby 3, terminatory := zostaną zignorowane i wreszcie średnik będzie terminatorem kończącym zapis liczby  $1.83456_{10}2$ . Natomiast drukując dla kontroli zawartość danej taśmy na flexowriterze otrzymujemy pełny zapis z terminatorami łącznie, co bardzo dobrze interpretuje rolę wczytywanych liczb 2, 3 i  $1.83456_{10}2$ .

W procedurze inone obowiązują wszystkie reguły podane dla procedury input, z wyłączeniem możliwości użycia ciągu samych minusów dla przypadku, gdy zmiennej chcemy pozostawić jej ostatnią wartość. Użycie takiego ciągu w procedurze inone nie jest dozwolone i dałoby wynik niezdefiniowany.

### 5.37. Zadanie

Jak można by w zadaniu 5.34 uniknąć wprowadzenia zmiennej integer m wykorzystując procedurę inone?

### 5.38. Procedura typein

Procedura typein (czytaj: tajp-yn) jest całkowicie podobna do procedury inone, z tym, że za każdym razem napotkania w czasie wykonywania programu nazwy "typein" maszyna zatrzymuje się, na pulpicie zapala się zielone światelko i maszyna oczekuje od operatora wypisania na maszynie do pisania liczby wraz z terminatorem kończącym, po którym automatycznie przechodzi do dalszego wykonywania programu.

Procedura typein może służyć np. do sterowania programem w czasie jego wykonywania przez maszynę. Jeśli np. S jest nazwą jakiegoś przełącznika, to umieszczenie w programie instrukcji

go to S[typein]

pozwala w czasie wykonywania programu wybrać adres skoku w zależności od otrzymywanych dotychczas wyników.

Procedura typein jest również bardzo wygodna przy wszelkiego rodzaju metodach relaksacji, ponieważ umożliwia wprowadzanie poprzez maszynę do pisania nowych wartości na zmienne w zależności od dotychczasowych wyników obliczeń, np.:

x:= typein; y:= typein; z:= typein;

Dla ułatwienia orientacji, na jaką zmienną mamy w danej chwili podstawić wartość, dobrze jest poprzedzać wywołanie procedury typein instrukcją writetext z odpowiednim komentarzem. I tak ostatni przykład można by zaprogramować następująco:

writetext( $\langle x := \_ \rangle$ ); x:= typein; writetext( $\langle \_ y := \_ \rangle$ ); y:= typein;

writetext( $\langle \_ z := \_ \rangle$ ); z:= typein;

albo krócej:

writetext( $\langle x,y,z := \_ \rangle$ ); x:= typein; y:= typein; z:= typein;

### 5.39. Przykład

Podamy przykład programu sterowanego za pomocą procedur wejścia z maszyny do pisania. Będzie to program dla rozwiązania układu równań

$$\begin{aligned}x^3 - 2y^3 + x^2 + y - 1 &= 0, \\2x^3 + 3y^3 + x - 2y^2 + 6 &= 0,\end{aligned}$$

za pomocą poszukiwania minimum funkcji

$$R(x,y) = (x^3 - 2y^3 + x^2 + y - 1)^2 + (2x^3 + 3y^3 + x - 2y^2 + 6)^2$$

metodą gradientową, tzn. metodą największego spadku. Wychodząc z dowolnego punktu  $(x_0, y_0)$  obliczamy  $R(x_0, y_0)$ , a następnie staramy się dotrzeć do takiego punktu  $(x_0 + \Delta x, y_0 + \Delta y)$  z warunkiem, aby wektor  $(\Delta x, \Delta y)$  miał określoną długość  $h$ , tzn.

$$\sqrt{(\Delta x)^2 + (\Delta y)^2} = h,$$

w którym wartość funkcji  $R(x,y)$  byłaby możliwie najmniejsza. Można wykazać, że wektor  $(\Delta x, \Delta y)$  powinien mieć kierunek wektora  $(\frac{\partial R}{\partial x}, \frac{\partial R}{\partial y})$ , tzn. powinno być

$$\Delta x = c \cdot \frac{\partial R}{\partial x}, \quad \Delta y = c \cdot \frac{\partial R}{\partial y}$$

gdzie  $c$  jest stałą, a pochodne cząstkowe  $\frac{\partial R}{\partial x}$  i  $\frac{\partial R}{\partial y}$  należy obliczyć w punkcie  $(x_0, y_0)$ . Dla celów obliczeniowych wystarczy zamiast wektora  $(\frac{\partial R}{\partial x}, \frac{\partial R}{\partial y})$  przyjąć wektor o zbliżonym do niego kierunku, a mianowicie wektor  $(R_x, R_y)$ , gdzie

$$R_x = R(x_0 + h, y_0) - R(x_0, y_0), \quad R_y = R(x_0, y_0 + h) - R(x_0, y_0).$$

Wtedy przyrosty  $\Delta x$  i  $\Delta y$  można obliczać ze wzorów:

$$\Delta x = -h \cdot \frac{R_x}{\sqrt{(R_x)^2 + (R_y)^2}}$$

$$\Delta y = -h \cdot \frac{R_y}{\sqrt{(R_x)^2 + (R_y)^2}}$$

Wprowadzając oznaczenie

$$p = \frac{h}{\sqrt{(R_x)^2 + (R_y)^2}}$$

otrzymujemy

$$\Delta x = -pR_x, \quad \Delta y = -pR_y$$

Z tych wzorów skorzystamy w następującym programie. Posługując się tym programem z początku posuwamy się dłuższymi skokami, tzn. większymi wartościami  $h$ , a następnie w zależności od wyników obliczeń zmieniamy  $h$  za pomocą maszyny do pisania i procedury typein. Aby upewnić się, czy zadanie nie ma większej liczby rozwiązań, rachunek przeprowadzamy wychodząc z różnych punktów początkowych, dla których wartości  $x, y$  wprowadzamy za pośrednictwem procedury typein.

A oto program:

```
begin comment Rozwiązanie układu równan;
real x,y,h,r1,r2,Rx,Ry,p;
switch S:= S1,S2,S3,S4,S5,S6;
real procedure R(u,v); value u,v; real u,v;
R:= (u3-2xv3+u2+v-1)2+(2xu3+3xv3+u-2xv2+6)2;
S1: writetext(⟨<...S=...⟩); go to S[typein];
S2: writecr; writetext(⟨<x,y=...⟩);
    x:= typein; y:= typein; go to S1;
S3: writecr; writetext(⟨<h=...⟩); h:= typein; go to S1;
S4: writecr; write(⟨<-d.dddd,+,d⟩,writetext(⟨<x=...⟩),x,
    writetext(⟨<...y=...⟩),y,writetext(⟨<...R=...⟩),R(x,y)); go to S1;
S5: r1:= R(x,y);
S6: Rx:= R(x+h,y)-r1; Ry:= R(x,y+h)-r1;
    p:= h/sqrt(Rx2+Ry2);
    r2:= R(x-pRx,y-pRy);
```

```

if r1>r2 then begin r1:= r2;  x:= x-pXRy;  y:= y-pXRy;
                                go to S6 end;

go to S4 end;

```

#### 5.40. Procedura kbon

Procedura kbon (czytaj: kej-bi-an) jest również bezparametrową procedurą pseudofunkcyjną - w wyniku jej wykonania otrzymujemy wartość logiczną, czyli wartość typu Boolean podstawioną w miejsce "kbon" w wyrażeniach booleowskich. Wartość kbon zależy od ustawienia specjalnego klucza KB na pulpicie maszyny. Jeśli klucz KB włączymy (ręcznie), wartość kbon będzie true, w przeciwnym razie false (stąd nazwa procedury : KB on). Procedura kbon umożliwia zatem ręczne sterowanie programem z pulpitu.

Przykłady:

- 1) if kbon then go to S3
- 2) if kbon then write( $\downarrow$ -d.dddd<sub>10</sub>+ $\downarrow$ ,writecr,X)
- 3) b:= kbon/\n>12

W p r z y k ł a d z i e 1 mamy instrukcję warunkową. Jeśli na pulpicie maszyny klucz KB jest w chwili wywołania danej instrukcji wciśnięty, instrukcja ta jest równoznaczna z instrukcją skoku do etykiety S3. Jeśli klucz KB nie jest wciśnięty, instrukcja jest równoznaczna z pustą.

W p r z y k ł a d z i e 2 mamy do czynienia z warunkowym drukiem aktualnej wartości jakiejś zmiennej X według podanego wzorca po uprzednim zrobieniu powrotu karetki do nowego wiersza. Druk nastąpi, jeżeli w momencie wywołania danej instrukcji klucz KB będzie wciśnięty, w przeciwnym razie druku nie będzie. Z tego rodzaju instrukcji korzystamy, gdy żałujemy czasu na zbyt częste drukowanie pośrednich wartości, a chcemy drukować tylko na doraźne żądanie wyrażone przyciśnięciem klucza KB.

W p r z y k ł a d z i e 3 użyto procedury kbon dla określenia wartości jakiejś zmiennej booleowskiej b. Na zmienną tę będzie podstawiona wartość true wtedy i tylko wtedy, gdy w momencie wykonywania przez maszynę danej instrukcji podstawienia będzie wciśnięty klucz KB i równocześnie aktualna wartość pewnej zmiennej n będzie większa od 12. W przeciwnym razie na zmienną b zostanie podstawiona wartość false.