

```

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^>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.

### 5.41. Zadanie

Licząc program z przykładu 5.39 na maszynie, możemy znaleźć się w sytuacji, w której maszyna już długo pracując nie kończy jakoś iteracji. Tymczasem może nam już kończyć się czas przeznaczony na liczenie i chcielibyśmy znać dotychczasowe wyniki albo po prostu chcemy sprawdzić stan liczenia. Jak można by w tym celu zmienić program wykorzystując do tego procedurę `kbon`?

### 5.42. Procedura `inchar`

Procedura `inchar` (skrót od `in-character`, czytaj: `inkar`, `inkarakter`), należy do bezparametrowych procedur pseudofunkcyjnych, dających w wyniku wykonania wartość liczbowa. Dla procedury `inchar` jest to wartość liczbowa, odpowiadająca pojedynczemu kodowi właściwemu, jaki zostaje w czasie wykonywania tej procedury odczytany przez czytnik z taśmy papierowej (przyporządkowanie liczb kodom zostało omówione w paragrafie 5.8). Uzyskana wartość zostaje podstawiona w miejscu wywołania procedury jako wartość `inchar`. Wywołanie procedury ma postać zmiennej o nazwie `"inchar"` i wchodzi w skład wyrażen arytmetycznych.

Należy zapamiętać, że przez procedurę `inchar` nie można otrzymać liczby odpowiadającej symbolowi niewłaściwemu (lista takich symboli została podana w paragrafie 5.8), ponieważ symbole te sterują procedurami wejścia i ten ich charakter jest przez procedurę `inchar` respektowany. Natomiast na `inchar` zostaje podstawiona wartość liczbowa, odpowiadająca pierwszemu napotkanemu kodowi właściwemu. Ponadto w przypadku `UPPER CASE` wartość przyporządkowana odczytywanemu aktualnie kodowi właściwemu jest automatycznie podwyższana o 128 tak, że np. wartością `inchar` dla `"p"` jest 39 a dla `"P"` 167.

Wartości `inchar` są traktowane jako wartości typu integer. A oto przykłady wywołania procedury `inchar`:

- 1) `u:= inchar`
- 2) `if inchar=64 then go to if inchar=30 then S3 else S5`

W p r z y k ł a d z i e 1 na zmienną liczbową `u` zostaje podstawiona wartość liczbowa, odpowiadająca pierwszemu z kolei kodowi właściwemu, jaki zostaje odczytany z taśmy papierowej na skutek napotkania przez maszynę wykonującą program w danej instrukcji nazwy `"inchar"`.

W p r z y k ł a d z i e 2 maszyna napotyka w danej instrukcji dwukrotnie nazwę `"inchar"`, na co za każdym razem reaguje odczytaniem jednego kodu właściwego z taśmy i podstawieniem przyporządkowanej mu liczby w miejsce aktualnego `"inchar"`. Oczywiście na ogół liczba podstawiona na pierwsze `"inchar"` będzie różna od podstawionej na drugie `"inchar"`. W danym przypadku, jeżeli pierwszy napotkany na taśmie papierowej kod właściwy jest kodem powrotu karetki, któremu odpowiada właśnie liczba 64, to instrukcja staje się równoznaczna z instrukcją skoku warunkowego

go to if inchar=30 then S3 else S5

Jeśli następny napotkany kod właściwy jest kodem tabulatora, któremu odpowiada liczba 30, powyższa instrukcja staje się równoznaczna z go to S3. Jeśli natomiast kod ten nie jest kodem tabulatora, powyższa instrukcja staje się równoznaczna z go to S5. Jeżeli jednak już pierwszy kod właściwy nie jest kodem powrotu karetki, cała dana instrukcja staje się równoznaczna z pustą i następny kod właściwy na taśmie papierowej już w ogóle w czasie wykonywania tej instrukcji nie zostanie odczytany.

Należy pamiętać, że czytanie liczby kończy się odczytaniem jednego tylko terminatora i wobec tego procedurą inchar następującą po czytaniu liczby będzie odczytywany pierwszy kod właściwy po terminatorze kończącym zapis tej liczby. Tym kodem właściwym może być następny terminator (wszystkie terminatory są kodami właściwymi) albo kod jednego z symboli pustych, tzn. SPACE lub   , albo pierwszy z symboli wchodzących do zapisu następnej liczby.

Za pomocą procedury inchar można sterować czytaniem przez maszynę taśmy papierowej. Na przykład można kazać maszynie tak długo czytać taśmę papierową bez reagowania na odczytywane kody, aż pojawi się żądany kod. Jeśli temu kodowi odpowiada - na przykład - liczba c, można to zrobić za pośrednictwem instrukcji

A: if inchar=c then go to A else ...

która spowoduje wykonanie instrukcji napisanej po else dopiero po odczytaniu przez maszynę kodu odpowiadającego liczbie c. To samo można uzyskać za pośrednictwem instrukcji

for i:= 0 while inchar=c do;

gdzie po do użyto instrukcji pustej, a zmienna kontrolowana i:= 0 ma charakter czysto formalny.

Za pomocą procedury inchar można również sygnalizować zakończenie czytania serii liczb, jeśli po niej na taśmie papierowej umieścimy kod charakterystyczny, nie używany poprzednio.

### 5.43. Zadanie

Jak należałoby zmienić program z zadania 5.35 w przypadku, gdy liczba punktów obserwacyjnych nie jest znana, natomiast wiemy, że na taśmie papierowej z danymi trójkami liczb x,y,w między kolejnymi trójkami oprócz terminatora kończącego jest jeszcze co najmniej jeden kod właściwy, a po ostatniej trójce - bezpośrednio po terminatorze kończącym zapis ostatniej liczby - jest kod symbolu < poprzednio nie używanego?

### 5.44. Zadanie

Jak zmienić program z zadania poprzedniego w przypadku, gdy nie znamy kodu kończącego na taśmie papierowej serię da-

nych, ale wiemy, że jest on również podany jako pierwszy kod właściwy na tej taśmie?

#### 5.45. Procedura typechar

Procedura typechar (skrót od type-charakter, czytaj: tajpkar, tajpkarakter) jest bardzo podobna do procedury inchar, z tym że wartością typechar jest liczba przyporządkowana symbolowi, który należy napisać na maszynie do pisania na dany przez maszynę sygnał. Sygnałem tym jest zapalenie się zielonego światełka na pulpicie, z chwilą gdy maszyna napotka w programie nazwę "typechar" i zatrzyma się. Ponadto przez procedurę typechar można uzyskać sytuację UPPER CASE, tzn. podwyższenie wartości o 128, tylko w przypadku bezpośredniego poprzedzenia danego symbolu przyciśnięciem klawisza UPPER CASE. W przeciwnym razie procedura typechar zakłada zawsze sytuację LOWER CASE.

A oto przykłady wywołania procedury writechar:

- 1) m:= typechar
- 2) if typechar=inchar then Z:= inon?

W p r z y k ł a d z i e 1 mamy do czynienia z podstawieniem na zmienną m liczby odpowiadającej symbolowi przyciśniętemu na klawiaturze maszyny do pisania w momencie, gdy maszyna zatrzymała się i dała znać zapaleniem zielonego światełka, że czeka na podanie symbolu. Należy pamiętać, że na maszynie do pisania nie należy tu przyciskać klawiszy odpowiadających symbolom niewłaściwym.

W p r z y k ł a d z i e 2 mamy do czynienia z instrukcją warunkową. Jeśli na maszynie do pisania w momencie zapalenia się zielonego światełka przyciśniemy klawisz z symbolem odpowiadającym kodowi, który w chwilę potem zostaje odczytany z taśmy na skutek wywołania procedury inchar, to cała instrukcja spowoduje podstawienie na zmienną Z liczby, która po wspomnianym kodzie jest wyperforowana na taśmie papierowej. Gdybyśmy przycisnęli inny klawisz (ale z symbolem właściwym), dana instrukcja stałaby się równoznaczna z pustą. Gdybyśmy przycisnęli klawisz z symbolem niewłaściwym, podstawienie na typechar wartości nie nastąpiłoby, maszyna zareagowałaby na przyciśnięty klawisz w zwykły sposób i nadal czekałaby na przyciśnięcie klawisza z symbolem właściwym.

Procedurą typechar można sterować programem poprzez maszynę do pisania, sterować czytaniem taśmy papierowej, wyprowadzać dowolne kody na taśmę papierową poprzez instrukcję

outchar(typechar)

5.46. Zadanie

Zaprojektować instrukcję skoku do etykiety A albo B w zależności od przyciśniętego symbolu na maszynie do pisania, za pośrednictwem procedury typechar.

5.47. Zadanie

Jak można by zmienić program z przykładu 5.39, aby nie korzystać z przełączników, a sterować programem za pośrednictwem procedury typechar?

5.48. Zadanie

Na taśmie papierowej mamy wyperforowanych kilka serii danych do programu z zadania 5.44. Jak należałoby zmienić ten program, aby można było w trakcie liczenia za pośrednictwem procedury typechar wybierać żadaną serię danych? Jakiego założenia musi spełniać w tym celu taśma papierowa z danymi?

5.49. Zadanie

Jaką instrukcją w programie można spowodować wyperforowanie na taśmie papierowej na perforatorze kodu powrotu karetki? Podać cztery następujące typowe rozwiązania:

- 1) odrębną procedurę standardową,
- 2) za pośrednictwem procedury outtext,
- 3) za pośrednictwem procedury outchar,
- 4) przez uderzenie na maszynie do pisania klawisza CAR RET, gdy maszyna będzie oczekiwała na podanie wartości typechar.

5.50. Procedura lyn

Procedura lyn (czytaj: lin) jest bezparametrową procedurą pseudofunkcyjną, dającą wartość typu integer i bardzo podobną do procedury inchar. Różnica między nimi polega na tym, że dla procedury lyn nie zastrzega się, aby pobierane kody były kodami właściwymi. Innymi słowy, procedura lyn nie traktuje symboli niewłaściwych jako sterujących procedurami wejścia, ale - przeciwnie - traktuje je na równi z innymi, przyporządkowując im odpowiednią wartość liczbową. Gdyby na przykład kodem czekającym na czytelnika na czytanie był na taśmie papierowej kod UPPER CASE, a maszyna znajdowała się w sytuacji LOWER CASE, to wywołanie procedury lyn, na przykład w instrukcji

c:= lyn

spowoduje podstawienie na lyn liczby 60 jako przyporządkowanej UPPER CASE, natomiast pozostawi maszynę w sytuacji LOWER CASE. Wywołanie procedury inchar spowodowałoby tu przedstawienie maszyny na sytuację UPPER CASE, a na inchar zostałaby podstawiona liczba odpowiadająca pierwszemu napotkanemu na taśmie papierowej kodowi właściwemu.

Z powyższego charakteru procedury lyn wynika, że jej wartościami mogą być tylko liczby całkowite od 0 do 127 włącznie, podczas gdy wartościami inchar mogły być liczby całkowite między 0 a 254 włącznie.

### 5.51. Zadanie

Należy rozwiązać zadanie 5.48 w przypadku, gdy wybierana seria danych na taśmie papierowej rozpoczyna się i kończy charakterystycznym dla niej kodem litery „Z” (w sytuacji LOWER CASE), z tym że seria ta zawiera grupę danych ujętych – jakby nawiasami – kodami PUNCHOFF i PUNCHON. Należy rozwiązać trzy warianty:

1) mają być wczytane wszystkie dane oprócz zawartych między PUNCHOFF i PUNCHON,

2) mają być wczytane tylko dane zawarte między PUNCHOFF i PUNCHON,

3) mają być wczytane wszystkie dane, łącznie z zawartymi między PUNCHOFF i PUNCHON.

### 5.52. Procedura char

Procedura char (skrót od character, czytaj: kar, charakter), należy do bezparametrowych procedur pseudofunkcyjnych i jest podobna do inchar. Jej wykonanie daje wartość liczbową typu integer odpowiadającą ostatniemu kodowi właściwemu wprowadzonemu do maszyny przez jakąkolwiek z procedur:

input, inone, typein, inchar,  
typechar, setchar, outcopy, writecopy.

Różnica między procedurami inchar i char polega jedynie na tym, że pierwsza z nich uruchamia urządzenia wejściowe (czytelnik lub maszynę do pisanie) i pobiera z nich jeden kod właściwy, natomiast druga nie uruchamia żadnego z urządzeń wejściowych, a wykorzystuje ostatni z poprzednio pobranych kodów właściwych, który jest przechowywany w maszynie.

A oto przykłady wywołania procedury char:

1) go to if char=137 then A else B

2) c:= char

W p r z y k ł a d z i e 1 mamy do czynienia z instrukcją skoku warunkowego. Jeżeli ostatnio odczytanym z taśmy papierowej kodem właściwym był kod znaku ) w sytuacji UPPER CASE, instrukcja ta jest równoznaczna z instrukcją go to A. Jeżeli ostatnio odczytanym był inny kod właściwy, instrukcja jest równoznaczna z instrukcją go to B.

W p r z y k ł a d z i e 2 następuje podstawienie na zmienną c liczby odpowiadającej ostatnio odczytanemu z taśmy papierowej kodowi właściwemu. Jeżeli to odczytanie miało miejsce w sytuacji UPPER CASE, wartość char jest powiększona o 128, jak w procedurze inchar.

Za pomocą procedury char można wygodnie rozpoznawać po terminatorach kończących, czy seria danych na taśmie papierowej została zakończona. Nie potrzeba wtedy po terminatorze kończącym jeszcze jednego kodu charakterystycznego jak w przypadku użycia procedury inchar. Procedura inchar jest jednak lepsza, gdy serię montujemy z kilku taśm z danymi, kończących się normalnymi terminatorami, a zakończenie serii dajemy oddzielnie wyperforowanym kodem charakterystycznym dla danej serii.

### 5.53. Zadanie

Rozwiązać zadanie 5.44 posługując się procedurą char zamiast inchar. Jakie założenia musi spełniać wtedy taśma papierowa z danymi?

### 5.54. Procedura setchar

Wywołanie procedury setchar (skrót od set-character, czytaj: setkar, setkarakter) ma postać następującej instrukcji:

setchar(wyrażenie arytmetyczne)

jak na przykład

1) setchar(137)

2) setchar(3-zxsign(y+2))

Działanie procedury setchar jest następujące:

- zostaje obliczona wartość wyrażenia arytmetycznego stanowiącego aktualny parametr danej instrukcji setchar,
- otrzymana wartość jest zaokrąglona do wartości typu integer i wzięta modulo 128 (tzn. zamiast niej zostaje wzięta dodatnia reszta z podzielenia jej przez 128),
- tak otrzymaną liczbę przechowuje się w maszynie tak długo, aż nie pojawi się wywołanie jednej z następujących procedur:

input, inone, typein, inchar,  
typechar, outcopy, writecopy



i wtedy przechowywana liczba zostaje wykorzystana tak, jakby odpowiadający jej kod był doperforowany na taśmie papierowej przed pierwszym czekającym na czytanie rzędkiem, jeżeli napotkaną procedurą jest input, inone, inchar, outcopy, writecopy, oraz tak jakby odpowiadający jej symbol był jako pierwszy napisany na maszynie do pisania zanim przyciśniemy jakikolwiek klawisz, jeżeli napotkaną procedurą jest typein albo typechar.

Wywołanie procedury lyn nie zmienia liczby przechowywanej w maszynie na skutek wywołania procedury setchar.

Należy tu podkreślić, że przez wywołanie procedury setchar można przechować tylko liczby odpowiadające kodom właściwym. Wywołanie procedury setchar z niewłaściwym argumentem nie powoduje zmiany przechowywanej liczby.

Procedura setchar umożliwia nam zbadanie pierwszego czekającego na czytanie kodu właściwego, jak gdyby nie ruszając go. Wystarczy w tym celu zastosować po sobie dwie instrukcje:

```
c:= inchar; setchar(c);
```

Dalsze czytanie taśmy będzie po tym wyglądało tak, jakby wyglądało, gdyby powyższych instrukcji w ogóle nie było. Tu natomiast jest możliwe przetestowanie wartości c, a więc przetestowanie pierwszego czekającego na czytanie kodu właściwego. Wyjątek stanowi czytanie za pośrednictwem procedury lyn w przypadku, gdy na czytanie czeka kod niewłaściwy: gdyby powyższych dwu instrukcji w ogóle nie było, procedura lyn odczytałaby ów kod niewłaściwy; tymczasem wstawienie wspomnianych dwu instrukcji powoduje przeskok do pierwszego kodu właściwego.

### 5.55. Zadanie

Jakie kody będą przechowywane w maszynie na skutek wywołania instrukcji podanych w przykładach 1 i 2 poprzedniego paragrafu, gdy aktualnymi wartościami y i z są odpowiednio -5 i 6?

### 5.56. Zadanie

Na taśmie papierowej wyperforowano jakiś ciąg całkowitych liczb dodatnich. Ostatnia liczba kończy się terminatorem / poprzednio nigdzie na tej taśmie nie użytym. Napisać program, który spowodowałby dodanie do siebie tych wszystkich liczb z danej taśmy, które rozpoczynają się cyfrą 4, i wydrukowanie sumy na maszynie do pisania według wzorca <ndddd>.

### 5.57. Procedury wejściowo-wyjściowe

Procedury wejściowo-wyjściowe powodują kopiowanie taśmy papierowej albo przez perforowanie nowej taśmy na perforato-



rze (procedura outcopy), albo automatyczne pisanie na maszynie do pisania podłączonej bezpośrednio do maszyny (procedura writecopy). Są to zatem procedury, których wywołanie uruchamia zarówno czytnik taśmy papierowej, a więc urządzenie wejściowe, jak też jedno z urządzeń wyjściowych (albo oba, jak to było opisane w paragrafie 5.8). Jak zobaczymy niżej, kopiowanie połączone jest zazwyczaj z pewnymi drobnymi modyfikacjami.

### 5.58. Procedura outcopy

Wywołanie procedury outcopy ma postać następującej instrukcji:

outcopy(wyrażenie łańcuchowe)

z tym, że

- wyrażenie łańcuchowe ma tu zawsze wartość w postaci tekstu ujętego w nawiasy  $\langle \langle i \rangle \rangle$ ,

- tekst będący wartością danego wyrażenia łańcuchowego musi składać się z jednego albo co najwyżej dwu symboli.

Działanie instrukcji outcopy zależy od tego, czy wspomniany tekst składa się z jednego czy dwu symboli. W pierwszym przypadku kopiowanie taśmy papierowej rozpoczyna się od aktualnego miejsca przygotowanego do czytania na taśmie na czytniku, a kończy przy pierwszym napotkaniu kodu symbolu stanowiącego wspomniany tekst, który to symbol już nie jest kopiowany. W drugim przypadku zostaje uruchomiony najpierw tylko czytnik, a kopiowanie rozpoczyna się od pierwszego napotkanego kodu pierwszego z symboli stanowiących wspomniany tekst i kończy przy pierwszym napotkaniu (ale już po rozpoczęciu kopiowania) drugiego symbolu. Oba ograniczające symbole nie są kopiowane.

Kopiowanie nie obejmuje kodów CLEAR CODE, SUM CODE, PUNCH OFF, PUNCH ON (ale tylko w przypadku, gdy odgrywa rolę nawiasu zamykającego w stosunku do poprzednio otwartego przez PUNCH OFF), END CODE, TAPE FEED, które zachowują swój charakter sterujący procedurami wejścia. Kopiowanie nie obejmuje również nadmiernych kodów LOWER CASE i UPPER CASE.

A oto przykłady wywołania procedury outcopy:

1) outcopy( $\langle \langle i \rangle \rangle$ )

2) outcopy(ab)

W przykładzie 1 mamy do czynienia z instrukcją skopiowania odcinka czytanej taśmy papierowej, poczynwszy od pierwszego napotkanego kodu symbolu " $\langle \langle i \rangle \rangle$ " aż do następnego pierwszego napotkanego kodu " $\rangle \rangle$ ", z tym że same nawiasy ( $\langle \rangle$ ) nie zostaną skopiowane.

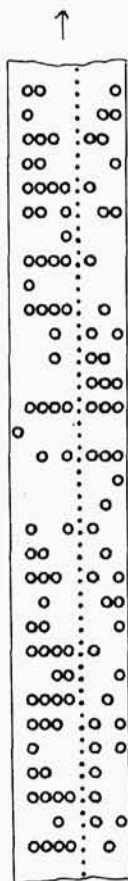
W przykładzie 2 parametrem nie jest tekst, ponieważ brak nawiasów  $\langle \langle i \rangle \rangle$ . Mamy tu zatem do czynienia z parametrem formalnym kategorii string. Dana instrukcja outcopy musi zatem występować w ciele jakiejś procedury, w deklaracji której parametr ab jest wyspecyfikowany jako string.

Dopiero po podstawieniu w wywołaniu tej procedury - jako parametru aktualnego - konkretnego tekstu jedno- lub dwusymbolowego, ujętego w nawiasy  $\langle \langle i \rangle \rangle$ , można określić odcinek taśmy, jaki ma być skopiowany.

Warto zauważyć, że do wiernego kopiowania taśmy, tzn. bez respektowania szczególnego charakteru kodów niewłaściwych, nie ma potrzeby wykorzystywania maszyny cyfrowej, gdyż czynność tę może wykonać na przykład sam flexowriter.

### 5.59. Zadanie

Na czytniku maszyny założona jest taśma papierowa z następującymi wyperforowanymi kodami



Jak będzie wyglądała taśma wyperforowana na perforatorze na skutek wywołania instrukcji

outcopy( $\langle \langle i \rangle \rangle$ )

przy danym ustawieniu powyższej taśmy na czytniku?

### 5.60. Procedura writecopy

Procedura writecopy jest bardzo podobna do procedury outcopy. Różnica między nimi – prócz tego, że kopiowanie taśmy papierowej założonej na czytnik polega teraz na wydrukowaniu odpowiadających symboli na maszynie do pisania – istnieje jeszcze w zbiorze symboli kopiowanych: na maszynie do pisania nie wszystkie kody wyperforowane na taśmie mają swoje odpowiedniki. Kody nie mające odpowiedników na maszynie do pisania nie są oczywiście kopiowane.

A oto przykład wywołania procedury writecopy:

```
writecopy(if alfa>beta then <<;> else C)
```

W przykładzie tym wykonanie instrukcji writecopy zależy od aktualnych wartości pewnych zmiennych alfa i beta. Jeśli wartość zmiennej alfa jest w momencie wywołania danej instrukcji nie mniejsza niż wartość beta, dana instrukcja jest równoznaczna z instrukcją

```
writecopy(<<;>)
```

w przeciwnym razie z instrukcją

```
writecopy(C)
```

W pierwszym przypadku przedruk taśmy papierowej na maszynie do pisania nastąpi od aktualnie nastawionego miejsca taśmy na czytniku aż do pierwszego napotkanego średnika, wyłączając ten ostatni. W drugim przypadku parametrem procedury writecopy jest C, a zatem parametr formalny kategorii string jakiejś procedury, w ciele której występuje dana instrukcja writecopy. Dopiero po podstawieniu na C parametru aktualnego w postaci tekstu jedno- lub dwusymbolowego, ujętego w nawiasy << i >, można określić odcinek taśmy papierowej przeznaczony do kopiowania.

### 5.61. Zadanie

Na taśmie mamy wyperforowany następujący tekst:

Naglowek:(Numer problemu:)Podac numer;

Taśma ta swym początkiem jest założona na czytnik. Maszyna wykonując program napotyka w tej sytuacji instrukcję

```
writecopy(<<()>)
```

Jak będzie wyglądało wykonanie tej instrukcji?

## 5.62. Współpraca z bębnem

Jak o tym już była mowa, programy w zakresie dotychczas omówionym, pozwalają - ze względu na ograniczoność pamięci operacyjnej (ferrytowej) maszyny - na operowanie równocześnie około 600-700 zmiennymi. W przypadku użycia w programie równocześnie większej ilości zmiennych program nie dobiega końca, maszyna zatrzymuje się po zapełnieniu pamięci operacyjnej i drukuje na maszynie do pisania:

alas

albo

array

(ten drugi przypadek ma miejsce, gdy maszyna już w deklaracji rozpoznaje macierz o za dużych rozmiarach). Oznacza to niemożność wykonania przez maszynę danego programu, ale nie oznacza niemożności wykonania danego problemu, gdyż np. możemy zmienić jego program wykorzystując bęben magnetyczny.

Bęben magnetyczny jest urządzeniem, na którym można zapamiętać 12 800 liczb. Z tego na ogół około 5 800 miejsc jest zarezerwowanych dla przechowywania translatora i programu. Wolnych dla programującego jest na ogół około 7 000 miejsc. Im dłuższy program, tym mniej wolnych miejsc na bębnie.

O ile współpraca z bębmem w zakresie translatora jest całkowicie zautomatyzowana i programujący może się nią w ogóle nie interesować, o tyle współpraca z bębmem w zakresie miejsc swobodnych nie jest całkowicie automatyczna i odbywa się za pośrednictwem dwóch specjalnych procedur: to drum i from drum oraz zmiennej standardowej drumplace. Są to procedury służące do programowego przesyłania informacji z pamięci operacyjnej na bęben i odwrotnie. Procedura to drum (znaczy: na bęben; czytaj: tu dram), służy do kopiowania zawartości pamięci operacyjnej na bębnie, podobnie jak procedura output służyła do kopiowania zawartości pamięci operacyjnej na taśmie papierowej. Procedura from drum (znaczy: z bębna; czytaj: from dram) służy do kopiowania zawartości bębna w pamięci operacyjnej, podobnie jak procedura input służyła do kopiowania zawartości taśmy papierowej w pamięci operacyjnej. Różnica leży m.in. w szybkości: przerzut informacji z pamięci operacyjnej na bęben lub odwrotnie trwa bez porównania krócej niż analogiczny przerzut na taśmę papierową poprzez mechaniczne urządzenie, jakim jest perforator. Natomiast w porównaniu do szybkości operacji wykonywanych w granicach pamięci operacyjnej, szybkość przerzutu informacji między pamięcią operacyjną a bębmem jest kilkadziesiąt razy wolniejsza. Należy tu jednak pamiętać i o tym, że poza świadomym korzystaniem z bębna przez programistę odbywa się jeszcze współpraca z bębmem automatyczna, o czym już wspomnieliśmy. Z tego też powodu nie jest również opłacalne zbytnie obciążanie pamięci operacyjnej, gdyż powoduje to zwiększenie przerzutów automatycznych na bęben i z powrotem, i znacznie bardziej opłacalne może okazać się programowe przemieszczenie części materiału na bęben. Dlatego staramy się, aby ilości zmiennych, którymi operujemy równocześnie w pamięci operacyjnej, nie przekraczały 500. Można to uzyskać nawet w dużych programach, jeżeli umiejętnie wykorzystamy pamięć bębnową lub buforową. Na-

leży zauważyć, że rezerwacja miejsc pamięci dla zmiennych, etykiet itp. jest robiona z chwilą wejścia do bloku z daną deklaracją. Rezerwacja taka jest automatycznie kasowana z chwilą opuszczenia tego bloku. Wynika stąd, że liczba zarezerwowanych miejsc w pamięci operacyjnej stale się zmienia w trakcie wykonywania programu.

Z pamięci operacyjnej na bęben lub odwrotnie można przesyłać jedynie tablice. Wymaga to nieraz nadania w programie przesyłanemu na bęben materiałowi formy tablicy.

### 5.63. Zmienna standardowa drumplace

Zmienna standardowa drumplace (znaczy: miejsce na bębnie; czytaj: dramplejs) może być traktowana jako bezparametrowa procedura pseudofunkcyjna. Jej wartość liczbowa jest zawsze typu integer i określa miejsce na bębnie, gdzie ma być zapisana lub skąd odczytana kolejna tablica. Po przesłaniu tablicy z pamięci operacyjnej na bęben lub odwrotnie, wartość drumplace zmienia się automatycznie. Aby po zapisaniu tablicy na bębnie można ją było później odnaleźć dla przepisania do pamięci operacyjnej, konieczne jest uprzednie zapamiętanie odpowiedniej wartości drumplace. Można to zrobić przez podstawienie drumplace jako wartości na jakąś zmienną, np.:

```
c:= drumplace
```

bezpośrednio przed instrukcją powodującą zapis danej tablicy na bębnie.

Natomiast znajomość aktualnej wartości liczbowej drumplace jest dla programującego zazwyczaj zbyteczna. Wystarcza bowiem wiedzieć, że ta aktualna wartość została podstawiona np. na zmienną c i że ta zmienna daną wartość reprezentuje. Należy zapamiętać, że wartości drumplace rozpoczynają się od możliwie największej, a po każdym przepisywaniu na bęben lub z bębna automatycznie zmniejszają się.

Gdy wartość drumplace przekroczy możliwości bębna, tzn. bęben będzie już zapełniony, a program będzie żądał dalszych na nim zapisów, maszyna zatrzyma się i na maszynie do pisania automatycznie wydrukuje

```
drum alas
```

Przykłady użycia zmiennej standardowej drumplace:

```
1) a:= drumplace-5xc
```

```
2) drumplace:= c
```

W p r z y k ł a d z i e 1 na zmienną a zostaje podstawiona aktualna wartość drumplace zmniejszona o pięciokrotną wartość c.

W p r z y k ł a d z i e 2 na drumplace zostaje podstawiona wartość aktualna c. Jeśli c reprezentuje wartość drumplace, którą otrzymaliśmy dla wprowadzenia jakiejś tablicy



na bęben, to podstawienie `drumplace := c` poprzedza instrukcję przepisania tejże tablicy z powrotem do pamięci operacyjnej, określając miejsce, w którym dana tablica znajduje się na bębnie.

#### 5.64. Procedura to drum

Procedurę `to drum` (znaczy: na bęben, czytaj: tu dram) można traktować zarówno jako нефunkcyjną, jak też jako pseudofunkcyjną, tzn. można ją wywoływać zarówno w postaci osobnej instrukcji

`to drum(nazwa tablicy)`

jak też wewnątrz wyrażeń arytmetycznych, jak np.:

`a := c - to drum(B)`

W tym drugim przypadku wartość `to drum` jest zawsze typu integer.

W obu przypadkach parametrem procedury `to drum` jest nazwa tablicy, jaka na skutek wywołania tej procedury zostaje z pamięci operacyjnej przepisana na bęben.

W czasie przepisywania tablicy, np. tablicy A, z pamięci operacyjnej na bęben, wartość `drumplace` automatycznie zmienia się tak, jakby została wykonana instrukcja

`drumplace := drumplace + to drum(A)`

Innymi słowy, wartość `to drum(A)` określa zmianę `drumplace` w czasie wykonania `to drum(A)`. Wartość ta jest zawsze ujemna, tzn. `drumplace` zmniejsza się po `to drum(A)`.

Kolejność zapisu wyrazów tablicy na bębnie jest zawsze taka, jaka była opisana dla procedury `input` w paragrafie 5.33, ale w odwrotnym porządku: początek tablicy dla najmniejszej wartości `drumplace`, koniec dla największej.

#### 5.65. Procedura from drum

Procedura `from drum` (znaczy: z bębna, czytaj: from dram) jest analogiczna do `to drum` z tym, że jej wywołanie powoduje przepisanie tablicy z bębna do pamięci operacyjnej. Przez wywołanie `from drum(A)` wartość `drumplace` również zmienia się jakby według instrukcji:

`drumplace := drumplace + from drum(A)`

gdzie wartość `from drum(A)` jest zawsze ujemna.

Warto zauważyć, że tablica przepisywana z bębna do pamięci operacyjnej nie musi być dosłownie tą samą tablicą, która poprzednio została na dane miejsce przepisana na bęben. Obie tablice muszą mieć ten sam typ (real, integer albo Boolean) i tę samą liczbę wyrazów, ale układ tych wyrazów w wiersze

i kolumny może być różny. Kolejność zapisu i czytania wyrazów tablicy jest jednak zawsze taka, jaka była opisana dla procedury input w paragrafie 5.33, tylko w odwrotnym porządku: początek tablicy dla najmniejszej wartości drumplace, koniec dla największej.

### 5.66. Zadanie

Tablicę A należy z pamięci operacyjnej przepisać na bęben, a następnie z bębna podstawić jej wyrazy do tablicy B w pamięci operacyjnej. Napisać odpowiedni fragment programu.

### 5.67. Zadanie

Tablice A, C, D i P zostały wprowadzone do programu deklaracją

```
array A[1:m,1:n], C[1:n], D[1:m], P[1:1]
```

Tablica A została przesłana z pamięci operacyjnej na bęben programem

```
c:= drumplace;
```

```
to drum(A);
```

po czym została w pamięci operacyjnej wymazana.  
Jak należy zaprogramować:

- 1) podstawienie na P[1] wartości A[i,j],
- 2) podstawienie na C i-go wiersza tablicy A,
- 3) podstawienie na D j-ej kolumny tablicy A.

### 5.68. Zadanie

Rozwiązać zadanie poprzednie w przypadku, gdy tablica A jest za duża dla pamięci operacyjnej i jej deklaracja jak wyżej spowodowałaby wydrukowanie przez maszynę sygnału alarmowego

```
array
```

Wobec tego tablica A nie jest w ogóle deklarowana i wprowadzona na bęben z taśmy papierowej za pośrednictwem tablicy C programem