Zadanie 8-14. Przekształć plik PLIKDIR utworzony poleceniem DIR /S > PLIKDIR wydanym w odniesieniu do napędu dysków elastycznych A (B) na plik ASCII, w którego każdym wierszu mieściłyby się informacje o poszczególnych plikach zapisanych na dyskietce znajdującej się w napędzie A (B): etykieta dysku (*volume*), nazwa pliku, rozszerzenie i jego rozmiar; dane powinny być zapisane w sposób umożliwiający import pliku do bazy danych (np. umieszczone w cudzysłowach i oddzielone od siebie przecinkami).

Zadanie 8-15. Przejrzyj tablicę z nazwami plików i wypisz tylko te nazwy, które występują w niej więcej niż jeden raz.

Przejdziemy teraz do następnego etapu, jakim jest tworzenie algorytmów i schematów działania programów. Po poznaniu sposobów przedstawiania algorytmów, skupimy uwagę w rozdz. 8.4.3. na tworzeniu algorytmów do (niektórych z) podanych zadań.

8.4. Algorytmy. Schematy działania programów

Algorytm określa „przepis" na zrealizowanie przez program postawionego zadania. Przepis ten ma postać ściśle określonych reguł i ma zapewniać zakończenie postępowania w skończonej liczbie kroków.

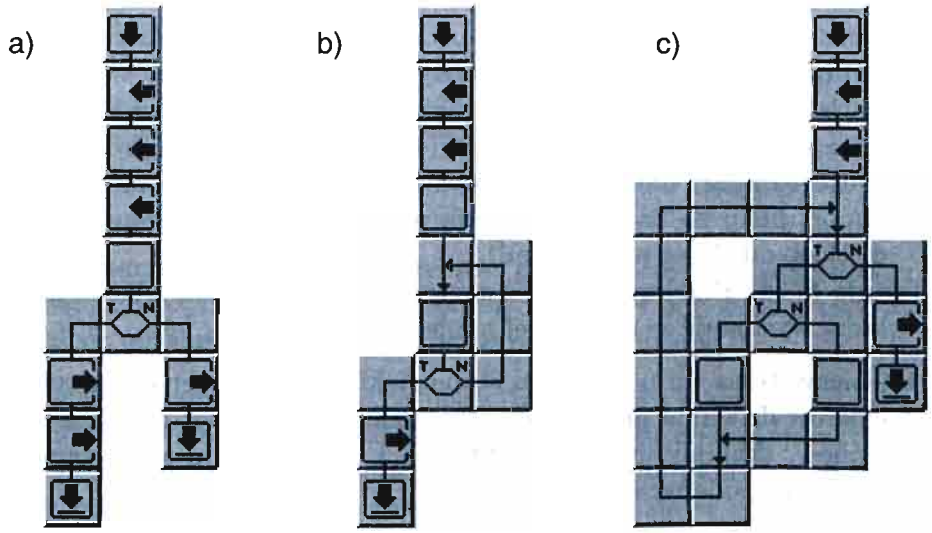
Znasz zapewne jakiś przepis kulinarny. Zapoznawszy się nim wiesz, jakie produkty powinieneś przygotować, żeby zabrać się za przyrządzenie potrawy lub napoju. Wiesz także, jakie czynności powinieneś wykonywać i w jakiej kolejności, a także kiedy (w jakiej sytuacji) należy je zakończyć.

Algorytmy służące do rozwiązywania problemów są przedstawiane z zachowaniem większej staranności co do formy algorytmu niż przy prezentowaniu przepisów kulinarnych. Do typowych form należy postać krokowa oraz sieć działań.

Nauka algorytmów może być wspomagana odpowiednimi programami (np. ELI). Umożliwiają one zarówno przedstawienie graficzne algorytmu (rys. 8.1), jak i demonstrowanie jego działania.

Przy przedstawianiu algorytmów będziemy używać zmiennych (takich jak w matematyce), oznaczanych symbolami złożonymi z jednej bądź wielu liter, i symboli podstawowych operacji matematycznych. Do umieszczenia danych i zmiennych będziemy korzystać (w razie potrzeby) z tablic podobnych do stosowanych w arkuszach kalkulacyjnych i bazach danych. Wykonywać będziemy także operacje na znakach i tekstach, podobne do znanych Ci z poprzednich rozdziałów.

Zmienną możesz sobie wyobrażać jako komórkę arkusza, tyle że opatrzoną **nazwą** (co w arkuszu jest też możliwe, ale nie jest konieczne).



Rys. 8.1. Przedstawienie w programie komputerowym algorytmów zawierających: a) badanie warunku; b) powtarzanie obliczeń; c) badanie warunku i powtarzanie obliczeń

Używając we wzorach nazwy zmiennej odwołujemy się do jej wartości, tak jak w arkuszu, używając adresu komórki — do jej zawartości.

Wartość zmiennej w trakcie obliczeń zmienia się, zatem na jednym etapie obliczeń zmienna o nazwie x może mieć wartość 0, a na innym np. 257; na jednym etapie obliczeń warunek $x = 7$ może być fałszywy, a na innym prawdziwy. Ze względu na taki „dynamiczny" charakter zmiennych używa się niekiedy specjalnego symbolu := oznaczającego **przypisanie** zmiennej znajdującej się po lewej stronie symbolu, wartości wyrażenia zapisanego po prawej stronie symbolu. Na przykład $x := 2$ oznacza nadanie zmiennej x wartości 2, zaś $x := x + 2$ oznacza zwiększenie wartości zmiennej x o 2 (najpierw na podstawie dotychczasowej wartości zmiennej x obliczana jest wartość wyrażenia $x + 2$ po prawej stronie, a później wartość ta nadawana jest zmiennej x). Będziemy posługiwać się przede wszystkim opisem słownym („nadaj zmiennej wartość", „zwiększ wartość zmiennej o"), lecz niekiedy zastosujemy również ów symbol.

Działania przedstawione w algorytmie mogą być realizowane w różny sposób: w pamięci, pisemnie, „ręcznie" lub też za pomocą programu komputerowego. Ręczne realizowanie algorytmu może mieć miejsce także podczas posługiwania się programem komputerowym, np. systemem zarządzania bazami danych; polega ono wówczas na wydawaniu przez człowieka kolejnych poleceń prowadzących do wykonania poszczególnych działań zawartych

w algorytmie. Realizowanie algorytmu za pomocą programu komputerowego prowadzi do samoczynnego wykonywania działań przez program (z wyjątkiem tych, w których twórca programu założy udział człowieka, na przykład związanych z wprowadzaniem danych).

Schemat działania programu realizującego algorytm jest zazwyczaj podobny do algorytmu. Można go przedstawiać w identycznej formie. W programie określa się więcej szczegółów niż w algorytmie; mogą one znaleźć odbicie w schemacie działania, który wówczas będzie zawierał więcej informacji niż algorytm. Na przykład przedstawiając algorytm pomija się często problem wprowadzania danych zakładając, że są one dostępne, natomiast w programie trzeba poświęcić uwagę ich wprowadzeniu (określić, za pomocą jakiego urządzenia wejściowego? w jaki sposób?).

W książce nie będziemy czynić wyraźnego rozróżnienia między algorytmami i wyrażonymi w podobny sposób schematami działania programów. Należy jednak pamiętać, że algorytm jest ogólnym przepisem rozwiązania problemu i do realizacji nie wymaga komputera, natomiast schemat programu jest już tworzony z myślą o realizacji komputerowej algorytmu i stanowi wzorzec do tworzenia programu.

8.4.1. Algorytm w postaci krokowej

Każdy algorytm może być przedstawiony w postaci krokowej, zawierającej listę kroków algorytmu, wykaz działań podejmowanych w każdym z kroków oraz ewentualnie warunki przechodzenia do innych kroków. Normalnie poszczególne kroki algorytmu są wykonywane w kolejności naturalnej (pierwszy, drugi, trzeci...), jednak w opisie kroku może być wskazany inny krok, który powinien być wykonywany jako następny w kolejności; przejście do innego kroku niż następny może ponadto zależeć od spełnienia pewnych warunków.

Najprostszy algorytm składałby się z jednego lub kilku kroków wykonywanych po kolei. Można przedstawić w taki sposób algorytm obliczenia średniego zużycia paliwa przez samochód po kolejnym zatankowaniu do pełna.

Algorytm-1

Krok 1. Oblicz przejechaną drogę $b - a$, jako różnicę obecnego stanu licznika kilometrów b i wskazania licznika podczas ostatniego tankowania do pełna a.

Krok 2. Podziel ilość wlanego od tego czasu paliwa h (w litrach) przez obliczoną długość drogi (w kilometrach).

Krok 3. Pomnóż otrzymaną liczbę przez 100 (ponieważ zużycie paliwa jest wyrażane w litrach na sto kilometrów) i przedstaw ją jako wynik.

Krok 4. Zakończ działanie.

Danymi dla algorytmu są liczby: a, b oraz h (niekiedy zaznacza się to wymieniając czynność ich odczytywania).

Podobnie prosty układ jak algorytm 1 mogą mieć programy. Nie znasz jeszcze programów pisanych w uniwersalnym języku programowania, trudno więc odwołać się do przykładu, który byłby Ci znany. Znasz jednak program zapisany w pliku CZESTO.BAT (rozdz. 2.5.2), wykonujący w poszczególnych krokach polecenia systemu operacyjnego (a nie języka programowania). Spróbuj przedstawić jego działanie w postaci krokowej.

Zadanie rozwiązania równania $ax + b = c$ dla dowolnych wartości współczynników a, b oraz c jest o tyle bardziej złożone od zadania obliczenia zużycia paliwa przez samochód, że trzeba uwzględnić różne możliwe sytuacje, także takie, które przy obliczaniu zużycia paliwa nie powinny się zdarzyć. Metodę rozwiązywania zapewne pamiętasz, choćby z rozdz. 5, gdzie rozwiązywałeś nieco bardziej rozbudowane równanie. Kiedy współczynnik a jest różny od zera, to po przeniesieniu współczynnika b na drugą stronę równania wystarczy otrzymaną różnicę $c - b$ podzielić przez a; kiedy natomiast $a = 0$, wtedy wartość lewej i prawej strony równania nie zależy od zmiennej x, zatem równanie albo jest zawsze prawdziwe (gdy $b = c$), albo jest sprzeczne (gdy $b \neq c$).

Algorytm rozwiązywania równania $ax + b = c$ dla wszelkich możliwych wartości współczynników a, b i c można zapisać w następujący sposób:

Algorytm 2

Krok 1. Odczytaj wartości liczb a, b i c. Jeżeli $a = 0$, to przejdź do kroku 3.

Krok 2. Oblicz $(c - b)/a$ i podaj wynik jako rozwiązanie równania. Przejdź do kroku 4.

Krok 3. Jeżeli $b = c$, to napisz komunikat, że równanie jest prawdziwe dla każdej wartości niewiadomej x, a w przeciwnym razie, że nie ma żadnych rozwiązań.

Krok 4. Zakończ działanie programu.

O ile wszystkie kroki algorytmu 1 są wykonywane w porządku naturalnym, o tyle w algorytmie 2 występują przejścia do kroków o numerze innym niż następny; niektóre z nich są bezwarunkowe, a inne zależą od spełnienia

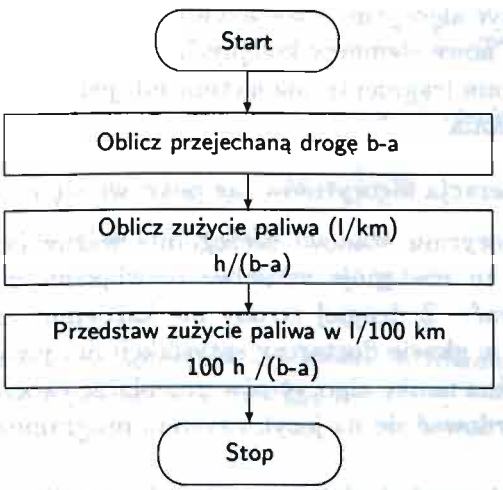
warunków. Na przykład, jeżeli a jest różne od zera, to po kroku pierwszym jest wykonywany krok drugi, w przeciwnym zaś razie — trzeci. Po kroku drugim zawsze jest wykonywany krok czwarty (z pominięciem trzeciego). Zauważ, że obliczenia zostaną zawsze zakończone po wykonaniu skończonej liczby kroków (1, 2 i 4 lub 1, 3 i 4).

Na podstawie tak wyrażonego algorytmu 2 będzie możliwe napisanie programu źródłowego w języku programowania.

8.4.2. Algorytm w postaci sieci działań

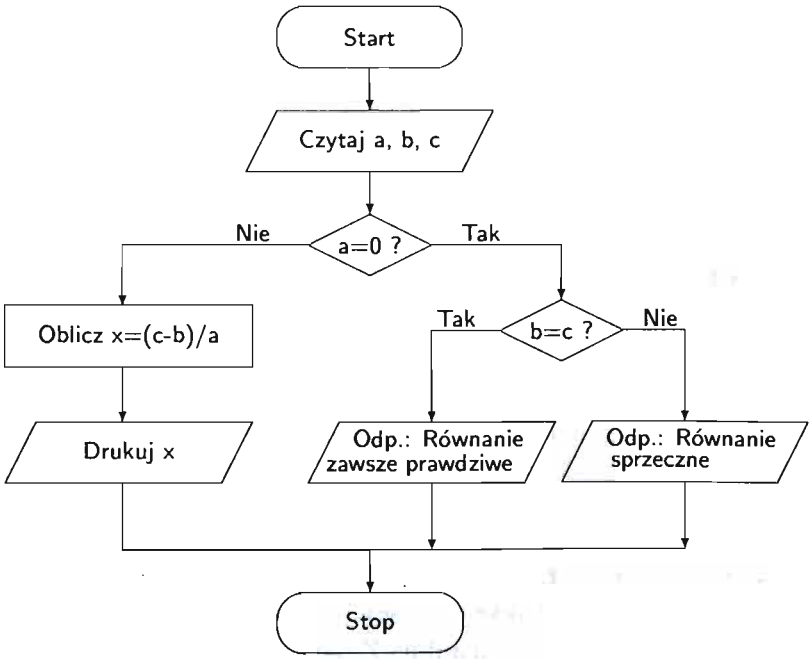
Inną, bardziej poglądową formą prezentacji algorytmu lub schematu działania programu jest **sieć działań**, zwana także **siecią czynnościową**. Poszczególnym krokom lub grupom kroków algorytmu odpowiadają tu bloki o różnym kształcie. Możliwości przechodzenia pomiędzy blokami są wskazywane przez linie ze strzałkami określającymi kierunek przejścia. Bloki podstawowe zawierają polecenia lub grupy poleceń odpowiadające wykonywaniu obliczeń i innych działań na danych (bloki prostokątne), a także odczytywaniu danych i prezentacji wyników (bloki o kształcie równoległoboku). W sieci działań występują także bloki warunkowe (o kształcie rombu), w których następuje sprawdzenie warunku. Z bloku warunkowego można przejść w jednym z dwóch kierunków zależnie od tego, czy warunek jest spełniony czy nie. Ponadto w sieci działań są stosowane bloki START i STOP (owalne), wskazujące rozpoczynanie działania i kończenie go.

Algorytm 1 ma bardzo prostą sieć działań (rys. 8.2).



Rys. 8.2. Algorytm 1 w postaci sieci działań

Algorytm 2 ma już strukturę bardziej złożoną (rys. 8.3).



Rys. 8.3. Sieć działań algorytmu 2 rozwiązania równania $ax + b = c$

8.4.3. Algorytmy rozwiązywania przykładowych zadań

Spróbujemy tworzyć algorytmy rozwiązywania zadań z rozdz. 8.3. Będziemy zwracać uwagę na nowe elementy kolejnych algorytmów i czasem ograniczymy się do omówienia fragmentu algorytmu lub jedynie koncepcji rozwiązywania danego zadania.

Uwaga. Numeracja algorytmów nie pokrywa się z numeracją zadań.

Tworzenie algorytmu stanowi szczególnie ważny etap tworzenia programu, ponieważ tu następuje właściwe rozwiązanie zadania, które program ma realizować. Z drugiej strony nie każdemu samo rozwiązywanie „na papierze" lub w głowie dostarczy satysfakcji motywującej do wytrwałej nauki. Wtedy można naukę algorytmów przeplatać tworzeniem programów. Należy tylko zdecydować się na język i system programowania (np. QBasic, Turbo Pascal).

Jeżeli w Twojej grupie będzie przyjęty taki sposób uczenia się, to przed przejściem do rozdziału poświęconego wybranemu systemowi programowania zapoznaj się z wiadomościami wstępnymi i uwagami na temat tworzenia programów źródłowych w rozdz. 8.5.

Zadanie 8-1

Rozwiązywanie równania kwadratowego $ax^2 + bx + c = 0$ przebiega, jak pamiętasz (rozdział 5.3.3; pliki RKWADR1, RKWADR2), w sposób następujący. Jeżeli jest to istotnie równanie kwadratowe, tzn. jeżeli współczynnik $a \neq 0$, to oblicza się najpierw *wyróżnik trójmianu* $\Delta = b^2 - 4ac$ i na podstawie jego znaku wnioskuje, czy równanie ma rozwiązania ($\Delta \geq 0$), a jeżeli tak, to jedno ($\Delta = 0$) czy dwa ($\Delta > 0$). Jeżeli istnieją dwa rozwiązania, to wyrażają się wzorami:
$x_1 = \frac{-b-\sqrt{b^2-4ac}}{2a}$ oraz $x_2 = \frac{-b+\sqrt{b^2-4ac}}{2a}$, a jeżeli jedno rozwiązanie, to $x = \frac{-b}{2a}$.
(Zauważ, że jeżeli $\Delta = 0$, to oba wzory mają tę samą wartość $x = \frac{-b}{2a}$).

Ze względu na badanie warunków i uzależnienie dalszego postępowania od wyniku badania, algorytm obliczeń można przedstawić wzorując się na algorytmie 2. Uczynimy to dla postaci krokowej.

Algorytm 3

Krok 1. Odczytaj wartości liczb a, b i c. Jeżeli $a = 0$, to napisz komunikat, że równanie nie jest równaniem kwadratowym i przejdź do kroku 5.

Krok 2. Oblicz wyróżnik trójmianu $\Delta = b^2 - 4ac$ i sprawdź jego znak.

Krok 3. Jeżeli $\Delta < 0$, to napisz komunikat, że równanie nie ma rozwiązań i przejdź do kroku 6.

Krok 4. Jeżeli $\Delta = 0$, to napisz komunikat, że równanie ma jedno rozwiązanie: $x = \frac{-b}{2a}$.

Krok 5. Jeżeli $\Delta \neq 0$, to napisz komunikat, że równanie ma dwa rozwiązania:
$x_1 = \frac{-b-\sqrt{b^2-4ac}}{2a}$ oraz $x_2 = \frac{-b+\sqrt{b^2-4ac}}{2a}$.

Krok 6. Zakończ działanie programu.

Ćwiczenie 8-3

Przedstaw algorytm znajdowania pierwiastków równania kwadratowego $ax^2 + bx + c = 0$ w postaci sieci działań.

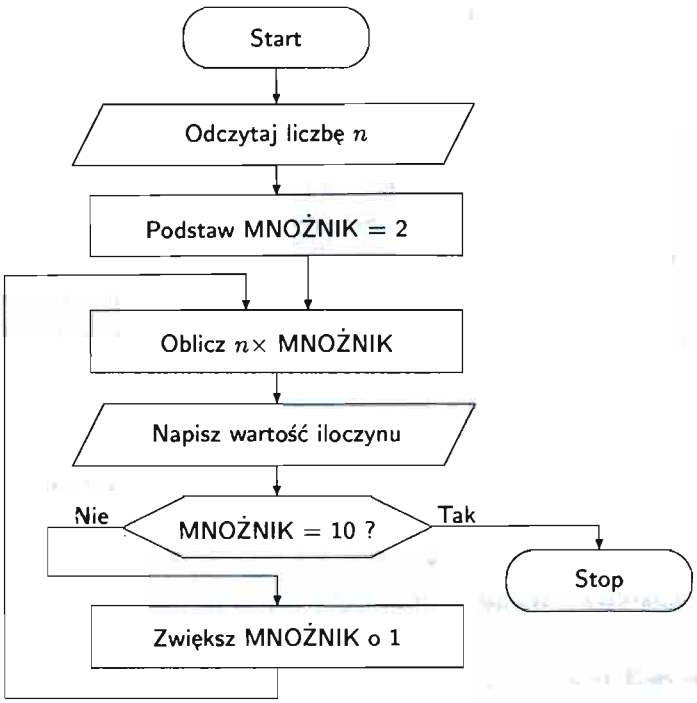
Zadanie 8-2

W realizacji zadania 8-2 można wyodrębnić takie operacje, jak: odczytywanie liczby, mnożenie dwóch liczb, wypisywanie komunikatów do użytkownika zawierających teksty i wartości liczbowe. Zamiast zapisywać w algorytmie dziewięć oddzielnych operacji mnożenia odczyta-

nej liczby kolejno przez 2, 3, ...,10 posłużymy się powtarzaniem operacji mnożenia w tym samym kroku algorytmu z każdorazowym zwiększaniem mnożnika o 1, tzn. od 2 do 10. Sam mnożnik potraktujemy tak jak zmienną w matematyce lub jak komórkę w arkuszu kalkulacyjnym, do której możemy wpisać dowolną liczbę.

Algorytm 4

Krok 1. Odczytaj liczbę n.
Krok 2. Nadaj zmiennej MNOŻNIK wartość 2.
Krok 3. Oblicz iloczyn $n \times$ MNOŻNIK i przedstaw wynik.
Krok 4. Zwiększ zmienną MNOŻNIK o 1. Jeżeli MNOŻNIK jest równy 10, to zakończ działanie; w przeciwnym razie przejdź do kroku 3.



Rys. 8.4. Sieć działań algorytmu 4 do zadania 8-2

Algorytm 4 ma nowy element struktury — **pętlę** (rys. 8.4), służącą do wielokrotnego wykonania tych samych kroków 2, 3 i 4, oraz umożliwiającą zakończenie działania algorytmu, kiedy jest spełniony warunek osiągnięcia przez zmienną MNOŻNIK wartości 10. Każdy obieg pętli zwany jest **iteracją**. W tym przypadku liczba powtórzeń pętli jest stała i wynosi — ile?

Ćwiczenie 8-4

Zastanów się, co byś zmodyfikował w algorytmie 4, żeby zakres liczb, przez które jest mnożona odczytana liczba n, również był odczytywany (podawany przez użytkownika).

Zadanie 8-3

Zadanie to można rozwiązywać dwoma sposobami: albo zapamiętywać wszystkie wprowadzone liczby, albo zapamiętywać jedynie niezbędne wyniki zbiorcze. Posłużymy się drugim sposobem.
W zmiennej SUMA będziemy zapamiętywać sumę wprowadzonych liczb od początku działania algorytmu, w zmiennej LICZ — numer kolejny wprowadzonej liczby, co umożliwi nam obliczanie wartości średniej, oraz w zmiennej NAJWIĘKSZA — największą z liczb. Przed rozpoczęciem odczytywania liczb musimy nadać zmiennej SUMA wartość początkową 0. Zmiennej NAJWIĘKSZA po odczytaniu pierwszej liczby nadamy wartość równą tej liczbie. Każdą z następnych liczb będziemy porównywać z NAJWIĘKSZA: jeżeli odczytana liczba okaże się większa, to jej wartość nadamy zmiennej NAJWIĘKSZA.

Algorytm 5

Krok 1. Nadaj zmiennym SUMA oraz LICZ wartość 0.
Krok 2. Odczytaj (wprowadzoną przez użytkownika) liczbę n.
Krok 3. Jeżeli $n < 0$, zakończ obliczenia.
Krok 4. Zwiększ wartość zmiennej SUMA o n i wartość zmiennej LICZ o 1. Jeżeli LICZ $= 1$ lub $n >$ NAJWIĘKSZA, to nadaj zmiennej NAJWIĘKSZA wartość n.
Krok 5. Wypisz wyniki, podając wartość zmiennych: SUMA i NAJWIĘKSZA. Przejdź do kroku 2.

Czy po przeanalizowaniu algorytmu zgadzasz się z tym, że w zmiennej NAJWIĘKSZA jest przechowywana największa z dotychczas porównywanych liczb (czy umiałbyś wytłumaczyć to koleżance lub koledze w razie potrzeby)? Czy forma komunikatu wydaje Ci się poprawna?

Nowym elementem tego algorytmu w stosunku do poprzednich są zmienne, w których pamiętane są wyniki podczas kolejnych obiegów pętli. Szczególną taką zmienną jest zmienna LICZ, pełniąca funkcję **licznika** iteracji (kolejny numer zmiennej jest równy liczbie obiegów pętli); tego typu zmienne występują w wielu algorytmach zawierających pętle.

W podobny sposób można gromadzić punkty zdobywane w grach lub podczas testów.

W odróżnieniu od algorytmu 3 liczba obiegów pętli nie jest znana z góry. Ponadto obliczenia wewnątrz pętli mogą w ogóle nie być wykonane, jeżeli pierwsza odczytana liczba będzie ujemna, ponieważ warunek jest badany na początku, natomiast w algorytmie 3 musiałyby być wykonane nawet przy innej wartości końcowej, ponieważ warunek zakończenia pętli był badany po wykonaniu obliczeń.

Zadanie 8-4

Koncepcja rozwiązania zadania 8-4 nie dla każdego musi być oczywista. Czynność porządkowania liczb jest podobna do układania rozdanych kart do gry lub do porządkowania banknotów w kolejności nominałów, możesz więc łatwiej wyobrazić sobie jej przebieg.

Odczytane liczby ustawimy „obok siebie" lub „jedna nad drugą" w tablicy LICZBY. Tablica taka to jakby jeden wiersz lub jedna kolumna w arkuszu kalkulacyjnym. Ponieważ w arkuszu wiersze są oznaczane literami, a kolumny numerami, wygodniej więc nam będzie posłużyć się wyobrażeniem kolumny. Do każdej z liczb będziemy się odwoływać dodając do nazwy LICZBY numer pozycji w tablicy wyrażony liczbą, np. LICZBY(2), lub nazwą zmiennej, np. LICZBY(POZYCJA), gdzie POZYCJA jest nazwą zmiennej.

Algorytmów porządkowania liczb jest wiele. Korzystałeś z nich w bazach danych, nie analizując jednak sposobu ich działania. Ponieważ umiesz już znajdować największą z n liczb, możesz więc posłużyć się tą umiejętnością, wykorzystując algorytm 4 po odpowiednich modyfikacjach.

Zadanie 8-4 możesz rozwiązać następująco. Wyznaczyć najpierw największą z wszystkich n liczb w tablicy i ustawić ją na ostatniej pozycji (n), odpowiednio przesunąwszy część pozostałych liczb w razie potrzeby; następnie wyznaczyć największą z pozostałych $n-1$ liczb i ustawić ją na pozycji przedostatniej $(n-1)$. Postępowanie takie można powtarzać n razy (lub $n-1$), do ustawienia wszystkich liczb.

Algorytm 6

Krok 1. Odczytaj liczbę porządkowanych liczb N (≥ 1) oraz same liczby, umieszczając je w tablicy LICZBY na kolejnych pozycjach od 1 do N.

Krok 2. Zmiennej NUMERITERACJI nadaj wartość N.

Krok 3. Jeżeli NUMERITERACJI ≤ 1, to przedstaw liczby z tablicy LICZBY w kolejności od 1 do N jako wynik końcowy i zakończ działanie (w przeciwnym razie przejdź do kroku 4).

Krok 4. Wyznacz największą z liczb w tablicy LICZBY na pozycjach od 1 do NUMERITERACJI (zapamiętaj ją pod nazwą NAJWIĘKSZA) oraz numer jej pozycji w tablicy (POZNAJWIĘKSZA). Jeżeli POZNAJWIĘKSZA jest równa zmiennej NUMERITERACJI, to przejdź do kroku 7.

Krok 5. W tablicy LICZBY przesuń na pozycje o numerze mniejszym o jeden liczby z pozycji od POZNAJWIĘKSZA + 1 do NUMERITERACJI.

Krok 6. Zapisz w tablicy LICZBY na pozycji NUMERITERACJI wyznaczoną w kroku 3 największą liczbę (NAJWIĘKSZA).

Krok 7. Zmniejsz o 1 NUMERITERACJI i przejdź do kroku 3.

Algorytm ten wymaga krótkiego komentarza. Jeżeli liczba n (N) jest równa 1, to do rozwiązania zadania nie trzeba podejmować żadnych działań (jednego banknotu ani jednej karty nie będziesz „układał w kolejności", nawet przy największym zamiłowaniu do porządku).

Po wyznaczeniu największej liczby ją samą „zabiera się" z dotychczasowego miejsca w tablicy i w kroku 6 zapisuje na pozycji o największym numerze w porządkowanej części tablicy (jest to zarazem numer iteracji), ale uprzednio przesuwa się „w lewo" lub „do góry" na zwolnione miejsce te liczby, które znajdowały się na pozycjach o numerach większych (ostatnia z przesuwanych liczb zwolni miejsce na zapisanie największej liczby). Postępowanie to jest zilustrowane poniżej dla tablicy (zapisanej poziomo) zawierającej 8 liczb.

```
pozycja tablicy                1   2   3   4   5   6   7   8
liczby na początku            24  36  -9  11 -33  20   1   6

N = 8
największa liczba (z ośmiu)       36
tablica po przesunięciu       24  -9  11 -33  20   1   6  36
N = 7
największa liczba (z siedmiu) 24
tablica po przesunięciu       -9  11 -33  20   1   6  24  36
N = 6
największa liczba (z sześciu)             20
tablica po przesunięciu       -9  11 -33   1   6  20  24  36
N = 5
największa liczba (z pięciu)      11
tablica po przesunięciu       -9 -33   1   6  11  20  24  36
N = 4
największa liczba (z czterech)            6
przesunięcie niepotrzebne
```

```
N = 3
największa liczba (z trzech)          1
przesunięcie niepotrzebne
N = 2
największa liczba (z dwóch)  -9
tablica po przesunięciu      -33  -9   1   6  11  20  24  36
```

Jeżeli wyznaczona w kolejnej iteracji w kroku 4 największa liczba w porządkowanej części tablicy jest na swojej właściwej pozycji, to nie trzeba ani przesuwania liczb w tablicy (krok 5), ani zapisywania liczby (krok 6).

Algorytm kończy działanie, gdy do uporządkowania zostaje jedna liczba, ponieważ liczba ta znajduje się na właściwej, tj. pierwszej pozycji w tablicy LICZBY.

Rozpoznajesz w algorytmie znaną Ci pętlę i badanie warunku. Nowym elementem oprócz tablicy, jako struktury danych, jest potraktowanie innego algorytmu (4) jako polecenia zapisanego w jednym kroku obecnego algorytmu 6. Ukazuje to pewną swobodę odnośnie szczegółowości opisywanych w algorytmie działań (powinny być zrozumiałe, ale niekoniecznie przedstawione drobiazgowo), a także możliwość traktowania jednych algorytmów jako elementów innych algorytmów, co upraszcza ich opis. W programach podobne uproszczenie opisu można uzyskać dzięki procedurom (podprogramom), które mogą realizować algorytmy pomocnicze (np. algorytm 4).

Działanie realizowane w kroku 5 jest też działaniem złożonym (możesz spojrzeć na nie jako na pomocniczy algorytm). Może ono być realizowane za pomocą znanej Ci już pętli, umożliwiającej wielokrotne wykonanie operacji przypisania zmiennym nowych wartości. Jeżeli np. podczas porządkowania tablicy w zakresie pozycji od 1 do 8 największa liczba zostanie znaleziona na pozycji 5, to w tablicy LICZBY należy najpierw przesunąć na pozycję piątą liczbę z pozycji szóstej, potem na pozycję szóstą z siódmej i w końcu na siódmą z ósmej. Kolejność podstawiania jest ważna, ponieważ wykonując te działania w kolejności odwrotnej zmieniłbyś wartości zapamiętanych w tablicy liczb.

Ćwiczenie 8-5

Zastanów się, w jaki sposób porządkujesz karty lub banknoty i spróbuj opisać swoje postępowanie w formie algorytmu.

Zadania 8-5, 8-6 i 8-7

Te zadania omówimy łącznie, ponieważ algorytmy do jednych mogą być wykorzystywane w następnych jako pomocnicze.

Zakładamy, że umiemy sprawdzić, czy dana liczba naturalna m jest podzielna przez liczbę naturalną k. Zadanie rozłożenia liczby naturalnej n na czynniki pierwsze może być rozwiązane metodą dzielenia tej liczby przez kolejne liczby naturalne „aż do skutku", tzn. aż do otrzymania ilorazu równego 1 lub całej liczby n, która w takiej sytuacji musi być uznana za liczbę pierwszą.

Algorytm 7

Krok 1. Odczytaj liczbę naturalną N (≥ 1).

Krok 2. Zmiennej CZYNNIK nadaj wartość 2, a zmiennej ILORAZ wartość N.

Krok 3. Jeżeli ILORAZ = 1 lub CZYNNIK > N, to zakończ działanie (w przeciwnym razie przejdź do kroku 4).

Krok 4. Zbadaj, czy liczba ILORAZ jest podzielna przez liczbę CZYNNIK. Jeżeli nie, to przejdź do kroku 6.

Krok 5. Podziel ILORAZ przez CZYNNIK, napisz komunikat stwierdzający, że kolejny czynnik ma wartość równą zmiennej CZYNNIK, i przejdź do kroku 3.

Krok 6. Zwiększ CZYNNIK o 1 i przejdź do kroku 3.

Przedstawiony algorytm jest mało efektywny, tzn. są algorytmy rozwiązujące to zadanie przy znacznie mniejszej liczbie działań, ale jest prosty i chyba nie budzi wątpliwości co do swej skuteczności. Możesz jednak zastanowić się, czy rzeczywiście jest potrzebne dzielenie liczby n przez każdą kolejną liczbę naturalną od 2 do n. Co by się stało, gdybyś pominął wszystkie liczby parzyste począwszy od 4? Może największy dzielnik (CZYNNIK) nie musi przekraczać $n/2$, a może nawet pierwiastka kwadratowego z liczby n? A czy w samym algorytmie 7 warunek zakończenia obliczeń można zmienić z CZYNNIK > N na CZYNNIK > ILORAZ?

Jeżeli wyniki rozkładu liczby n na czynniki chciałbyś wykorzystywać w dalszych obliczeniach, to musiałbyś je zapamiętać, np. w kroku 5 zapisywać każdy czynnik w kolejnej pozycji tablicy (i zarazem zliczać ilość tych czynników).

Najmniejszą wspólną wielokrotność i największy wspólny dzielnik dwóch liczb m oraz n można wyznaczyć na podstawie znanego rozkładu każdej z nich na czynniki. Można więc potraktować algorytm 7 jako algorytm pomocniczy, który byłby wykorzystany do rozłożenia na czynniki obu liczb. Pozostałoby porównanie zbiorów czynników. Część wspólna zbiorów dawałaby rozkład na czynniki największego wspólnego dzielnika, a suma zbiorów — rozkład najmniejszej wspólnej wielokrotności. Na przykład jeżeli liczbom 48 i 180 odpowiadają zbiory czynników: $\{2, 2, 2, 2, 3\}$ i $\{2, 2, 3, 3, 5\}$, to ich

częścią wspólną jest zbiór $\{2, 2, 3\}$, a sumą zbiór $\{2, 2, 2, 2, 3, 3, 5\}$. Mnożąc czynniki zawarte w wyznaczonych w ten sposób zbiorach otrzymujemy szukane wielkości. Zatem dla podanych przykładowo liczb 48 i 180 największy wspólny dzielnik wynosi 12 ($12 = 2 \times 2 \times 3$), a najmniejsza wspólna wielokrotność 720.

Porównywanie zapamiętanych czynników może wydać Ci się dość skomplikowane w zapisie formalnym (chociaż z ręcznym przeprowadzeniem go nie miałbyś zapewne żadnych problemów). Jeżeli tak jest, to spróbuj rozważyć inny sposób wyznaczania największego wspólnego dzielnika, podobny do zastosowanego w algorytmie 7. Sprawdzaj podzielność przez kolejne liczby naturalne obu liczb i dziel je wtedy tylko, kiedy obie są podzielne; jednocześnie obliczaj iloczyn tych czynników. Po zakończeniu algorytmu iloczyn ten będzie równy największemu wspólnemu dzielnikowi NWD. Najmniejszą wspólną wielokrotność wyznaczysz ze wzoru $m \times n/NWD$. Wzór ten nie powinien Cię zdziwić, jeżeli przypomnisz sobie rozkład liczb na czynniki. W iloczynie liczb czynniki wspólne wystąpią dwukrotnie: raz w liczbie m, a drugi raz w n, zatem należy je usunąć dzieląc przez nie. W rezultacie należy podzielić iloczyn przez największy wspólny dzielnik.

Algorytm 8

Krok 1. Odczytaj liczby naturalne M i N.

Krok 2. Zmiennej CZYNNIK nadaj wartość 2, zmiennej ILORAZM wartość M, a zmiennej ILORAZN wartość N. Zmiennej NWD nadaj wartość 1.

Krok 3. Jeżeli jest spełniony choć jeden z warunków: ILORAZM = 1 lub ILORAZN = 1, lub CZYNNIK > M, lub CZYNNIK > N, to przejdź do kroku 7.

Krok 4. Zbadaj, czy są spełnione oba warunki: liczba ILORAZM jest podzielna przez liczbę CZYNNIK oraz liczba ILORAZN jest podzielna przez liczbę CZYNNIK. Jeżeli nie, to przejdź do kroku 6.

Krok 5. Podziel ILORAZM przez CZYNNIK i podziel ILORAZN przez CZYNNIK. Pomnóż NWD przez CZYNNIK i przejdź do kroku 4.

Krok 6. Zwiększ CZYNNIK o 1 i przejdź do kroku 3.

Krok 7. Oblicz NWW mnożąc M przez N i dzieląc iloczyn przez NWD.

Krok 8. Zakończ obliczenia.

Uwaga. Do znalezienia największego wspólnego dzielnika można posłużyć się innymi algorytmami. Do najciekawszych należy opracowany w sta-

rożytności algorytm przypisywany Euklidesowi. Jest on nie tylko ciekawy jako metoda, lecz i efektywny. Jeżeli przyszłoby Ci czekać ze zniecierpliwieniem, aż komputer zakończy długo trwające obliczenia (co dla dużych liczb mogłoby się zdarzyć), to zapewne doceniłbyś kwestię wyboru algorytmu pod kątem jego efektywności. Niekiedy inne jeszcze względy decydują o przydatności tego bądź innego algorytmu do danego zadania. Nie będziemy omawiać tego problemu, zapamiętaj jednak, że w razie trudności z wykonywaniem obliczeń prostymi metodami warto zainteresować się, czy nie zostały opracowane metody (algorytmy) lepsze.

Jeżeli zainteresowałeś się wspomnianym algorytmem, to możesz się z nim zapoznać (a jeżeli zechcesz, to i zrealizować w postaci programu):

Algorytm 9

Krok 1. Odczytaj liczby naturalne M i N.
Krok 2. Jeżeli M > N, to podstaw do M resztę z dzielenia całkowitego M przez N; w przeciwnym razie podstaw do N resztę z dzielenia całkowitego N przez M.
Krok 3. Jeżeli żadna z liczb M, N nie jest równa 0, to przejdź do kroku 2.
Krok 4. Jako wynik przedstaw tę z liczb M, N, która jest różna od 0.
Krok 5. Zakończ obliczenia.

Uwaga. Największą wspólną wielokrotność liczb M i N mógłbyś obliczyć tak jak w algorytmie 8.

Trzecie z rozważanych zadań jest już łatwe. Skoro dysponujesz algorytmem wyznaczającym najmniejszą wspólną wielokrotnść i największy wspólny dzielnik dla dowolnej pary liczb naturalnych, poradzisz sobie z dodawaniem ułamków. Oznaczmy ich liczniki przez L1 i L2, mianowniki zaś przez M1 i M2. Postępowanie można wyrazić za pomocą algorytmu 10.

Algorytm 10

Krok 1. Odczytaj liczby naturalne L1, M1, L2 i M2.
Krok 2. Wyznacz największy wspólny dzielnik NWDM oraz najmniejszą wspólną wielokrotność NWWM dla liczb M1 i M2.
Krok 3. Oblicz liczniki ułamków po sprowadzeniu ich do wspólnego mianownika równego NWWM, tj. odpowiednio NL1 := L1 × M2 / / NWDM oraz NL2 := L2 × M1 / NWDM.
Krok 4. Oblicz sumę liczników NL := NL1 + NL2 i wyznacz największy wspólny dzielnik NWD2 nowego licznika NL i mianownika NWWM. Jeżeli jest równy 1, to przejdź do kroku 6.

Krok 5. Podziel NL oraz NWWM przez wyznaczony największy wspólny ich dzielnik NWD2.
Krok 6. Przedstaw NL i NWWM jako licznik i mianownik ułamka stanowiącego wynik.
Krok 7. Zakończ obliczenia.

Przedstawione postępowanie odpowiada obliczeniom pisemnym (ręcznym), gdzie może zależeć na tym, żeby liczby nie były duże. Przy obliczeniach komputerowych algorytm ten można by uprościć obliczając sumę ułamków bez wyznaczania wspólnych dzielników i wielokrotności, tj. licznik jako L1 × × M2 + L2 × M1, a mianownik jako M1 × M2, i dopiero wówczas wyznaczyć największy wspólny dzielnik licznika i mianownika i skrócić ułamek.

Zadanie 8-8

Omówimy jedynie koncepcję rozwiązania tego zadania. Dodawanie liczb wielocyfrowych jest oparte na dodawaniu liczb jednocyfrowych. Gdy wynik dodawania dwóch liczb jednocyfrowych jest liczbą dwucyfrową, wówczas pierwszą cyfrę (odpowiadającą dziesiątkom) zapisuje się jako przeniesienie. Jeżeli po dodaniu do siebie wszystkich par liczb jednocyfrowych znajdujących się na odpowiadających sobie pozycjach (pod sobą) wystąpi choć jedno (niezerowe) przeniesienie, to trzeba powtórzyć operację w celu dodania przeniesień. Jak widać na podanym przykładzie, może być konieczne kilkakrotne powtórzenie.

```
1 składnik          5 4 6
2 składnik          2 5 8
                    -----
sumy                7 9 4
przeniesienia         1
                    -----
sumy                7 0 4
przeniesienia       1
                    -----
wynik               8 0 1
```

Trudność zadania nie polega tu na obliczeniach, ponieważ te są dla Ciebie trywialne, lecz na sposobie ich przedstawienia. Ponieważ chodzi o naśladowanie obliczeń wykonywanych pisemnie, przyjmiemy więc, że dane zapisujemy w tablicy odpowiadającej papierowi „w kratkę" — jedna komórka tablicy jednej kratce. W algorytmie można zauważyć następujące elementy:

- dodawanie dwóch znajdujących się pod sobą liczb jednocyfrowych z zapisaniem poniżej (w linii sum) jednocyfrowego wyniku i jeszcze

niżej (w linii przeniesień) o jedną pozycję w lewo ewentualnego przeniesienia;
- sprawdzanie, czy w linii przeniesień znajduje się choć jedno (niezerowe) przeniesienie.

Do realizacji dodawania dwóch liczb jednocyfrowych mógłbyś albo utworzyć „tabliczkę dodawania" (analogiczną do tabliczki mnożenia) z określanym odrębnie wynikiem jednocyfrowym i przeniesieniem, albo badać, czy suma liczb jest większa od 9 (czyli dwucyfrowa), i jeżeli tak to odejmować 10 od sumy i różnicę traktować jako wynik, a wartość przeniesienia przyjmować równą 1.

Jeżeli z operacji dodawania dwóch liczb jednocyfrowych uczynić algorytm na tyle uniwersalny, żeby mógł być używany na różnych „wysokościach" i „szerokościach" naszego papieru w kratkę, to cały algorytm staje się prosty. Nietrywialne może być przedstawianie działań na ekranie, ale to należy już do programu, a nie do algorytmu. Zauważ, że w zapisie ręcznym często nie pisze się zer, lecz pozostawia puste miejsca; przy tworzeniu programu realizującego algorytm wymaga to szczególnej uwagi.

Ćwiczenie 8-6

Zastanów się nad algorytmem podobnego typu dla innego działania arytmetycznego (odejmowania, mnożenia). Opisz jego koncepcję.

Zadanie 8-9

Zadanie to dotyczy przede wszystkim sporządzania wykresów za pomocą programów komputerowych. Systemy programowania umożliwiają zazwyczaj pracę w trybie graficznym, różne są jednak ich możliwości tworzenia obrazu (wykraczające niekiedy poza pojęcia definiowane w języku programowania). Do najprostszych należy zmiana ogólnych parametrów obrazu, przede wszystkim koloru tła właściwego obrazu, koloru kursora i koloru obrzeża, oraz parametrów indywidualnych poszczególnych punktów — pikseli. Ograniczymy się tu do zmiany koloru poszczególnych punktów, przy czym uwzględnimy jedynie dwa: kolor odróżniający od tła (np. jasny punkt na ciemnym tle — piksel „zapalony") i kolor tła (piksel „zgaszony").

Położenie pikseli określa się w układzie współrzędnych xy za pomocą liczb całkowitych. Jest ich tyle, ile zapewnia karta graficzna (i jej tryb pracy), a więc np. dla karty VGA — 640 w poziomie i 480 w pionie. Piksele są numerowane od 0 (dla karty VGA 0,...639; 0,...479), przy czym pionowe są numerowane zazwyczaj od góry do dołu, co może sprawiać trudność wobec

naszego przyzwyczajenia do układu współrzędnych xy. Jeżeli oryginalne dane do tworzenia wykresów są wyrażone w innej skali lub za pomocą liczb rzeczywistych, to muszą być przekształcone na liczby całkowite z przyjętego zakresu. W algorytmie przyjmiemy, że do zmiany współrzędnych oryginalnych x i y na współrzędne wykresu XW i YW służą odpowiednio funkcje FX i FY.

Sam schemat tworzenia ruchomego wykresu jest prosty. Każdej kolejnej chwili czasu t_n odpowiada jeden punkt o współrzędnych $x(t_n), y(t_n)$, gdzie $x(t)$ oraz $y(t)$ są danymi zależnościami (wzorami). Musisz określić przedział czasu od chwili początkowej t_0 do końcowej t_k i przyjąć wielkość kroku Δt, o jaki czas będzie zwiększany. Powinieneś również określić zakres wartości przyjmowanych przez poszczególne współrzędne, a w każdym razie zakres wartości x i y, jakie będą odwzorowywane na wykresie. Możesz wykonać w tym celu pomocnicze obliczenia, wyznaczając wartości maksymalne x oraz y jak w algorytmie 5 do zadania 8-3 i w podobny sposób ich wartości minimalne. Możesz też wartości te oszacować lub nawet założyć dowolnie. Zamiast określania czasu końcowego możesz przyjąć inny warunek zakończenia algorytmu (programu).

Algorytm 11

Krok 1. Nadaj zmiennej CZAS wartość początkową t_0.

Krok 2. Oblicz współrzędne (oryginalne) punktu: X = x(CZAS) oraz Y = y(CZAS). Oblicz odpowiadające im współrzędne punktu ekranu: XW = FX(X) i YW = FY(Y). „Zapal" punkt ekranu o współrzędnych XW,YW.

Krok 3. „Zgaś" punkt ekranu o współrzędnych XW,YW (po odczekaniu pewnego czasu, potrzebnego do dogodnej prezentacji ruchomego wykresu na ekranie).

Krok 4. Sprawdź warunek zakończenia algorytmu (na przykład CZAS $\geq t_k$) i jeżeli jest spełniony, to zakończ działanie.

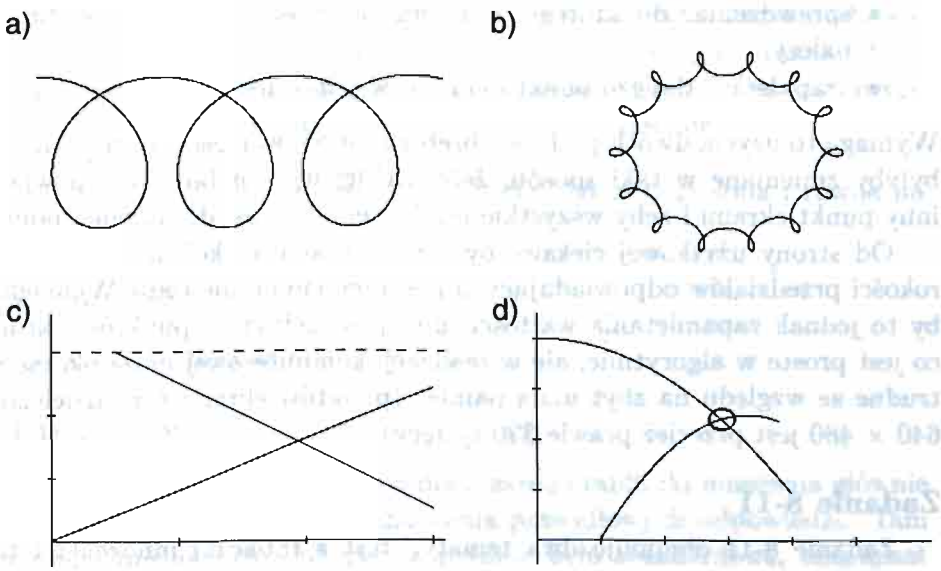
Krok 5. Zwiększ zmienną CZAS o Δt i przejdź do kroku 2.

Schemat działania jest prosty. Nowym elementem jest sformułowanie warunku w postaci CZAS $> t_k$ zamiast CZAS $= t_k$. Dotychczas operowaliśmy liczbami całkowitymi i wiadomo było, że przy zwiększaniu zmiennych o 1 będą one przyjmować wszystkie kolejne wartości, tu natomiast przyrost czasu Δt może być wielkością rzeczywistą, której żadna wielokrotność nie musi być równa wartości t_k. Użycie relacji $\geq$ zamiast $=$ zapewnia zakończenie obliczeń. Przy realizacji komputerowej problem ten jest jeszcze ważniejszy wobec niedokładnej reprezentacji liczb rzeczywistych i niedokładnych obliczeń.

Innym nowym elementem jest „spowalnianie" obliczeń (w kroku 3) w celu dostosowania sposobu prezentacji wyników do upodobań i możliwości odbiorcy. Jest to element programu (a nie algorytmu obliczeń), związany ze sprzętem komputerowym i konkretnym systemem programowania.

Sam tryb graficzny musi być w programie inicjowany, ponieważ naturalnym trybem dla uniwersalnych języków programowania jest tryb tekstowy.

Można w podobny sposób przedstawiać trajektorię ruchu części w mechanizmach (patrz rys. 8.5a), ruchu cząstek i planet (patrz rys. 8.5b), badać koincydencję dwóch ruchów (patrz rys. 8.5cd); chodzi nie tylko o przecięcie trajektorii, ale o spotkanie się dwóch ciał w jednym miejscu w tym samym czasie, jak na przykład pojazdów, które w typowych zadaniach wyjeżdżają z różnych miejsc i jadą z różnymi prędkościami), przebieg zmian wielkości fizycznych, na przykład napięcia i prądu w obwodzie elektrycznym itd.



Rys. 8.5. Wykresy: a) ruchu punktu na kole toczącym się po prostej; b) ruchu punktu na kole toczącym się po kole; c) ruchu dwóch pojazdów jadących ze stałą prędkością; d) ruchu dwóch ciał (rzut poziomy i rzut ukośny) z zaznaczonym miejscem zderzenia

Na ekranie możesz umieścić układ współrzędnych, zaznaczając osie, a jeżeli zechcesz zadać sobie nieco więcej trudu, to także skalę liczbową (lub co najmniej kreski na osiach oznaczające kolejne wielokrotności jednostek).

Zarówno tworzenie skali, jak i tworzenie poszczególnych elementów układu współrzędnych, np. kresek na osiach, możesz wyodrębnić w postaci oddzielnego algorytmu (podprogramu).

Ćwiczenie 8-7

Jeżeli zainteresowałeś się sporządzaniem wykresów, to gdy zapoznasz się z instrukcjami w systemie programowania, spróbuj przedstawić na wykresie tory punktu opisane zależnościami:

$x(t) = sin(t)$, $y(t) = cos(t)$, $t \geq 0$
$x(t) = 2^{-0,05t} sin(t)$, $y(t) = 2^{-0,05t} cos(t)$, $t \geq 0$
$x(t) = sin(mt)$, $y(t) = cos(nt)$, $t \geq 0$, $m, n = 1, 2, 3 \ldots$

Zadanie 8-10

Rozwiązanie tego zadania sprowadza się do:

- przyporządkowania zmiennym x i y punktów ekranu,
- obliczenia w każdym punkcie ekranu wartości badanej funkcji,
- sprawdzenia, do którego z podanych przedziałów wartość funkcji należy,
- „zapalenia" danego punktu ekranu w odpowiednim kolorze.

Wymaga to użycia dwóch pętli, w obrębie których wartości zmiennych x i y byłyby zmieniane w taki sposób, żeby każdej ich kombinacji odpowiadał inny punkt ekranu i żeby wszystkie punkty ekranu zostały uwzględnione.

Od strony użytkowej ciekawe byłoby oglądanie reakcji na zmianę szerokości przedziałów odpowiadających poszczególnym kolorom. Wymagałoby to jednak zapamiętania wartości funkcji w każdym z punktów ekranu, co jest proste w algorytmie, ale w realizacji komputerowej może okazać się trudne ze względu na zbyt małą pamięć (punktów ekranu o rozdzielczości 640 × 480 jest przecież prawie 320 tysięcy).

Zadanie 8-11

Zadanie 8-11 obejmuje dwa tematy: test z tabliczki mnożenia i test tekstowy.

Algorytm do zadania dotyczącego testu z tabliczki mnożenia utworzymy wykorzystując poznane już elementy. Oprócz zmiennej LICZ służącej do liczenia zadanych pytań użyjemy zmiennej DOBREODP do liczenia dobrych odpowiedzi. Założymy, że dysponujemy funkcją LOS dostarczającą liczb całkowitych losowanych z przedziału od 2 do 10 (z podobnej funkcji korzystałeś już w rozdz. 5.3.3). Wobec braku decyzji co do

liczby pytań w sformułowaniu zadania zrealizujemy wariant, w którym po każdej odpowiedzi użytkownik decyduje, czy zamierza test kontynuować czy nie.

Algorytm 12

Krok 1. Nadaj wartości zerowe zmiennym LICZ i DOBREODP.
Krok 2. Nadaj wartości losowe zmiennym CZYNNIK1 i CZYNNIK2. Pomnóż je przez siebie i wartość wyniku nadaj zmiennej ILOCZYN.
Krok 3. Odczytaj liczbę. Jeżeli jest równa wartości zmiennej ILOCZYN, to przejdź do kroku 5.
Krok 4. Napisz wynik cząstkowy, zawierający stwierdzenie, że odpowiedź jest błędna i podający wartość zmiennej ILOCZYN jako odpowiedź prawidłową. Przejdź do kroku 6.
Krok 5. Napisz wynik cząstkowy, zawierający potwierdzenie poprawności odpowiedzi danej przez użytkownika. Zwiększ wartość zmiennej DOBREODP o 1.
Krok 6. Zwiększ wartość zmiennej LICZ o 1. Odczytaj decyzję użytkownika. Jeżeli chce kontynuować test, to przejdź do kroku 2.
Krok 7. Podaj wynik końcowy: liczbę pytań (LICZ) i prawidłowych odpowiedzi (DOBREODP). Zakończ działanie.

Czy wiesz, jak zmienić algorytm, żeby liczba pytań była stała i równa na przykład 20?

Ćwiczenie 8-8

Sporządź sieć działań algorytmu 12.

Test tekstowy

Zadanie to różni się od zadania dotyczącego tabliczki mnożenia głównie rodzajem pytań i sposobem wyznaczania prawidłowych odpowiedzi. Tam odpowiedzi można było obliczyć, a pytanie było standardowe, natomiast tu zarówno pytania, jak odpowiedzi do wyboru oraz numery prawidłowych odpowiedzi muszą być przechowywane. Nowym elementem algorytmu, a nawet kilku algorytmów, jest tu sposób przechowywania danych i operowania na nich.

Każdemu pytaniu towarzyszy zestaw odpowiedzi. Zestawy te powinny być przechowywane tak jak pytania, żeby wybierając pytanie wybrać także właściwy zestaw odpowiedzi, spośród których uczeń ma wskazywać odpowiedź prawidłową.

Przyjmiemy, że odpowiedzi są ponumerowane i użytkownik wskazuje numer jednej z nich. Pytanie razem z odpowiedziami do wyboru możemy przedstawić jako jeden dany tekst (komunikat), a prawidłową odpowiedź — jako daną liczbę. Porównując wskazany przez użytkownika numer odpowiedzi na dane pytanie z numerem zapamiętanym możemy stwierdzić, czy odpowiedź jest prawidłowa.

Do przechowania danych wygodnie będzie posłużyć się tablicami podobnymi do stosowanych w bazach danych. Załóżmy zatem, że dane są umieszczone w wierszach dwóch tablic: pytania z zestawem odpowiedzi w tablicy PYTANIE, a numery prawidłowych odpowiedzi w tablicy ODPOWIEDŹ. Możesz też wyobrażać sobie jedną tablicę bazy danych zawierającą w każdym wierszu rekord z polem tekstowym PYTANIE i polem numerycznym ODPOWIEDŹ.

Zawartość wiersza n tablicy (pola rekordu) będziemy oznaczać dodając n w nawiasach po jej nazwie, tak jak oznacza się argument funkcji, na przykład PYTANIE(n), ODPOWIEDŹ(n).

Posłużymy się także zmienną NUMER oznaczającą numer kolejnego pytania i zarazem numer wiersza tablicy (lub rekordu).

Algorytm do tego zadania opracujemy w dwóch etapach. Najpierw uwzględnimy zadawanie nieokreślonej liczby pytań z większej listy, a w drugim etapie uwzględnimy ich losowe wybieranie.

Załóżymy, że liczba pytań w tablicy jest równa (co najmniej) 40 i użytkownik decyduje po każdym pytaniu, czy chce odpowiedzieć na kolejne.

Algorytm 13

Krok 1. Nadaj zmiennej DOBREODP wartość 0, a zmiennej NUMER wartość 1.

Krok 2. Napisz komunikat zawierający PYTANIE(NUMER) — czyli tekst z wiersza NUMER tablicy PYTANIE.

Krok 3. Odczytaj liczbę podaną przez użytkownika. Jeżeli jest równa wartości zmiennej ODPOWIEDŹ(NUMER) — czyli liczbie z wiersza NUMER tablicy ODPOWIEDŹ — to zwiększ zmienną DOBREODP o 1.

Krok 4. Jeżeli zmienna NUMER jest równa 40 (lub większa), to przejdź do kroku 6.

Krok 5. Odczytaj decyzję użytkownika, czy chce odpowiedzieć na następne pytanie. Jeżeli jest pozytywna, to zwiększ zmienną NUMER o 1 i przejdź do kroku 2.

Krok 6. Podaj wynik końcowy, zawierający liczbę pytań (NUMER) i liczbę prawidłowych odpowiedzi (DOBREODP).

Do uwzględnienia losowego wyboru pytań trzeba posłużyć się losowo wybieranym numerem pytania. Załóżymy zatem, że dysponujemy funkcją LOS40, której wartością jest liczba całkowita losowana za każdym razem spośród liczb od 1 do 40. Jej wartość będziemy zapamiętywać w zmiennej NUMERLOS i zamiast pytania o numerze NUMER będziemy używać pytania o numerze NUMERLOS.

Algorytm 14

Krok 1. Nadaj zmiennej DOBREODP wartość 0, a zmiennej NUMER wartość 1.

Krok 2. Wylosuj liczbę z przedziału od 1 do 40 i przypisz zmiennej NUMERLOS jej wartość. Napisz komunikat zawierający PYTANIE(NUMERLOS) — czyli tekst z wiersza NUMERLOS tablicy PYTANIE.

Krok 3. Odczytaj liczbę podaną przez użytkownika. Jeżeli jest równa wartości zmiennej ODPOWIEDŹ(NUMERLOS) — czyli liczbie z wiersza NUMERLOS tablicy ODPOWIEDŹ — to zwiększ zmienną DOBREODP o 1.

Krok 4. Jeżeli zmienna NUMER jest równa 40 (lub większa), to przejdź do kroku 6.

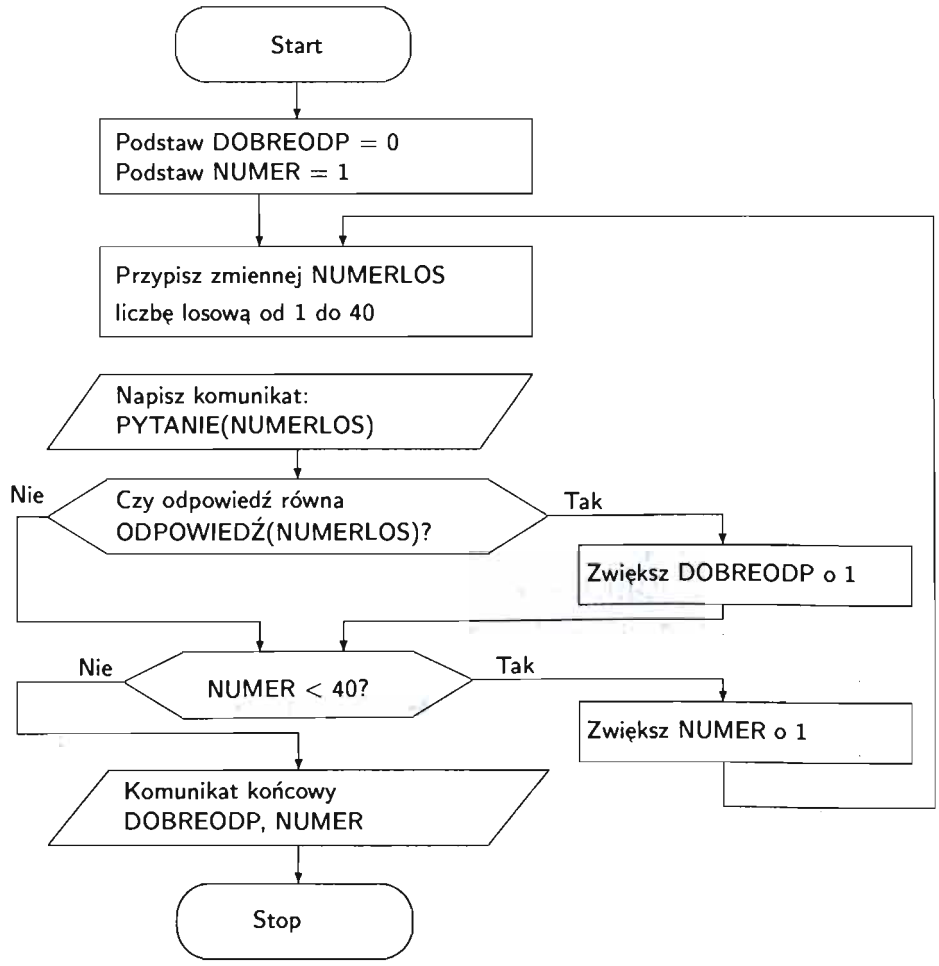
Krok 5. Odczytaj decyzję użytkownika, czy chce odpowiedzieć na następne pytanie. Jeżeli jest pozytywna, to zwiększ zmienną NUMER o 1 i przejdź do kroku 2.

Krok 6. Podaj wynik końcowy, zawierający liczbę pytań (NUMER) i liczbę prawidłowych odpowiedzi (DOBREODP).

Zmienna NUMER nadal określa liczbę zadanych pytań, ale numer pytania w tablicy jest określony przez zmienną NUMERLOS, losowaną od nowa przy każdym kolejnym zadawaniu pytania.

Algorytm 14 można przedstawić również w postaci sieci działań. Na rys. 8.6 przedstawiono algorytm różniący się od algorytmu 14 jedynie sformułowaniem pytania odpowiadającego krokowi 4.

Zastanów się, czy przy takim sposobie losowania numerów pytań, jaki został zastosowany w algorytmie 14, zadawane pytania nie mogą się powtórzyć. Rzeczywiście tak, ponieważ można powtórnie wylosować tę samą liczbę. Takie zjawisko w teście jest niepożądane. Można się przed nim zabezpieczyć na dwa sposoby. Pierwszy dotyczy użycia takiej metody losowania, w której liczby losowane się nie powtórzą. Wtedy nie trzeba modyfikować algorytmu. W drugim trzeba wprowadzić zmienne przechowujące informację, czy pytanie było już zadane. Spróbujmy zmodyfikować algorytm 14 stosownie do drugiego sposobu.



Rys. 8.6. Sieć działań Algorytmu 14

Wprowadzimy trzecią tablicę (pole rekordu) o nazwie ZADANE zawierającą zmienne logiczne. Zmienne logiczne, jak pamiętasz, przyjmują wartości Prawda i Fałsz.

Algorytm 15

Krok 1. Nadaj zmiennej DOBREODP wartość 0, a zmiennym NUMER oraz NUMERPOM wartość 1.

Krok 2. Nadaj zmiennej ZADANE(NUMERPOM) wartość Fałsz oraz zwiększ wartość zmiennej NUMERPOM o 1. Jeżeli NUMERPOM nie przekroczył liczby 40, to powtórz krok 2.

Krok 3. Wylosuj liczbę z przedziału od 1 do 40 i przypisz zmiennej NUMERLOS jej wartość. Jeżeli ZADANE(NUMERLOS) ma wartość Prawda (tzn. pytanie o wylosowanym numerze NUMERLOS zostało już zadane), to powtórz krok 3.

Krok 4. Napisz komunikat zawierający PYTANIE(NUMERLOS) i podstaw do zmiennej ZADANE(NUMERLOS) wartość Prawda.

Krok 5. Odczytaj liczbę podaną przez użytkownika. Jeżeli jest równa wartości zmiennej ODPOWIEDŹ(NUMERLOS), to zwiększ zmienną DOBREODP o 1.

Krok 6. Jeżeli zmienna NUMER jest równa 20 (lub większa), to przejdź do kroku 8.

Krok 7. Napisz komunikat: „Czy chcesz odpowiedzieć na następne pytanie? Tak/Nie". Odczytaj wpisaną przez użytkownika odpowiedź. Jeżeli jest równa „Tak", to zwiększ zmienną NUMER o 1 i przejdź do kroku 3.

Krok 8. Napisz komunikat końcowy — najpierw tekst: „Udzieliłeś", potem liczbę równą wartości zmiennej DOBREODP, następnie drugi tekst: „prawidłowych odpowiedzi na", potem liczbę równą wartości zmiennej NUMER i na końcu tekst: „pytań".

Na początku algorytmu wszystkim elementom tablicy ZADANE nadajemy wartość Fałsz na oznaczenie faktu, że żadne z pytań nie zostało zadane. Czynimy to w pomocniczej pętli w kroku 2; liczbę powtórzeń tej pętli kontrolujemy za pomocą zmiennej NUMERPOM.

Zadając kolejne pytanie zmieniamy wartość zmiennej ZADANE odpowiadającej numerowi wybranego pytania z Fałsz na Prawda. Losowanie numeru pytania powtarzamy natomiast aż do trafienia na takie pytanie, które nie zostało jeszcze zadane (krok 3). Liczbę zadawanych pytań ograniczyliśmy przy tym do 20; w ten sposób pytania będą wybierane z większej liczby, zgodnie z wymaganiem zadania 6.

Możesz mieć wątpliwości, czy taki sposób losowania „aż do skutku" jest dopuszczalny, jeżeli algorytm ma zakończyć się w skończonej liczbie kroków. Teoretycznie mogłoby się zdarzyć, że czekanie na właściwy numer pytania trwałoby „w nieskończoność". Jednak mechanizmy losowania stosowane w językach programowania mają takie cechy, że można mieć pewność wylosowania odpowiedniej liczby w skończonej liczbie kroków.

Jeżeli to tłumaczenie nie przekonuje Cię, to rozważ inny sposób losowania: zamiast losować za każdym razem liczbę od 1 do 40 losuj liczbę m z takiego przedziału, który odpowiada liczbie wolnych pytań (w pierwszej iteracji 40, w drugiej 39, w n-iteracji 41-n), a następnie przeszukaj wolne pytania w tablicy i zliczaj je aż do pytania numer m, które wybierzesz.

Uwaga. Numer m nie jest numerem pozycji w tablicy, lecz numerem kolejnym znalezionego wolnego pytania (np. po wykorzystaniu pytania 1 i 4 trzecim wolnym byłoby pytanie zapisane w piątym wierszu tablicy).

Ćwiczenie 8-9

Zapisz zaproponowany sposób losowania pytań w postaci algorytmu. Załóż, że dysponujesz funkcją losową LOS(n), losującą liczby z przedziału od 1 do n, gdzie wartość parametru n możesz określać.

Zadanie 8-12

Przyjmiemy, że plik dany do „przetłumaczenia" nazywa się DANE.TXT, a plik tworzony WYNIK.TXT. Dane są kody znaków, które mamy zastąpić innymi w nowym pliku (jak pamiętasz, każdy znak ma kod w postaci liczby całkowitej od 0 do 255 — rozdz. 4.4.4).

Przy zmianie kodu polskich liter należałoby uwzględnić 18 znaków, tu jednak dla uproszczenia ograniczymy się do czterech. Zamienimy zatem z kodu Mazovia na ISO Latin 2 litery: ą, ć, ę oraz ł. Odpowiada temu zmiana kodów: 134 na 177, 141 na 230, 145 na 234 i 146 na 179 (tabela 4.1).

Pliki można odczytywać i zapisywać znak po znaku; można także rozpoznać koniec pliku odczytywanego. Możemy z tego skorzystać i zapisywać plik WYNIK.TXT równocześnie z odczytywaniem pliku DANE.TXT. Dzięki temu nie musimy zapamiętywać wszystkich znaków jednego lub obu plików i rozważać, w jaki sposób mielibyśmy to uczynić. Algorytm tłumaczenia pliku sprowadza się dzięki temu do algorytmu zamiany pojedynczych znaków.

Odczytywanie i zapisywanie plików, a także ich otwieranie i zamykanie (znasz te operacje z programów użytkowych) należy właściwie do programu, a nie do algorytmu.

Po odczytaniu każdego znaku trzeba sprawdzić, czy nie powinien zostać zamieniony innym, a w razie odpowiedzi pozytywnej zastąpić go jego odpowiednikiem, po czym znak (zmieniony lub nie zmieniony) zapisać do pliku WYNIK.TXT. Postępowanie należy zakończyć, gdy w pliku DANE.TXT nie będzie więcej znaków.

Algorytm 16

Krok 1. Sprawdź, czy w pliku DANE.TXT jest jeszcze nie odczytany znak lub znaki. Jeżeli nie (z powodu dojścia do końca pliku), to zakończ działanie.

Krok 2. Odczytaj (kolejny) znak z pliku DANE.TXT i nadaj zmiennej ODCZYT wartość równą kodowi odczytanego znaku.

Krok 3. Jeżeli ODCZYT=134, to nadaj zmiennej ZAPIS wartość 177 i przejdź do kroku 8.
Krok 4. Jeżeli ODCZYT=141, to nadaj zmiennej ZAPIS wartość 230 i przejdź do kroku 8.
Krok 5. Jeżeli ODCZYT=145, to nadaj zmiennej ZAPIS wartość 234 i przejdź do kroku 8.
Krok 6. Jeżeli ODCZYT=146, to nadaj zmiennej ZAPIS wartość 179 i przejdź do kroku 8.
Krok 7. Nadaj zmiennej ZAPIS wartość zmiennej ODCZYT.
Krok 8. Dopisz do pliku WYNIK.TXT znak o kodzie ZAPIS. Przejdź do kroku 1.

Algorytm wykonuje obliczenia w pętli utworzonej przez przechodzenie z kroku 8 do kroku 1. Każda iteracja (powtórzenie obiegu pętli) obejmuje odczytanie jednego (kolejnego) znaku z pliku DANE.TXT, ewentualną zamianę na inny i zapisanie jednego znaku do pliku WYNIK.TXT. Jeżeli w pliku DANE.TXT nie ma już kolejnych znaków do odczytania, to obliczenia w pętli nie są ponawiane (następuje zakończenie obliczeń). Liczba iteracji zależy od liczby znaków w odczytywanym pliku DANE.TXT, nie jest zatem znana z góry, ale jest skończona.

Nowym elementem struktury w porównaniu z poprzednimi algorytmami jest fragment obejmujący kroki od 3 do 7. Wartość jednej zmiennej (w tym wypadku kod odczytanego znaku umieszczony w zmiennej ODCZYT) decyduje o **wyborze jednej z wielu dróg** obliczeń w algorytmie, a nie tylko jednej z dwóch. W naszym przykładzie mamy do czynienia z pięcioma drogami: cztery odpowiadają poszczególnym wartościom kodu i związane są ze zmianą kodu znaku zapisywanego, piąta zaś, realizowana w kroku 7, odpowiada pozostawieniu odczytanego znaku (wartości kodu) bez zmian. Tego typu strukturę można z łatwością rozbudować.

Ćwiczenie 8-10

1. Przedstaw algorytm 16 w postaci sieci działań.
2. Rozważ, jak zmodyfikowałbyś algorytm, żeby objąć zamianą wszystkie znaki polskie kodu Mazovia zawarte w pliku DANE.TXT.

„Szyfrowanie"

Do szyfrowania tekstu metodą zamiany kodów pojedynczych znaków można posłużyć się bardziej rozbudowaną wersją rozpatrzonego algorytmu. Mniej pracochłonnym rozwiązaniem jest użycie funkcji wyznaczającej wartość kodu zapisywanego znaku.

Funkcja powinna być tak dobrana, żeby możliwe było „odszyfrowanie" informacji (powinna być różnowartościowa).

Rozpatrzymy prosty algorytm „szyfrowania" polegający na zmniejszeniu kodów większych od 32 o 1 i przesunięcie 32 na koniec przez nadanie mu wartości 255. Przedstawimy go jako modyfikację algorytmu 16.

Algorytm 17

Kroki 1–2. Jak w algorytmie 16.

Krok 3. Jeżeli ODCZYT > 32, to zmiennej ZAPIS nadaj wartość ODCZYT – 1 i przejdź do kroku 8.

Krok 4. Jeżeli ODCZYT = 32, to zmiennej ZAPIS nadaj wartość 255 i przejdź do kroku 8.

Kroki 5–6. Pomiń (puste).

Kroki 7–8. Jak w algorytmie 16.

Kiedy sporządzisz program realizujący ten algorytm, możesz zmodyfikować algorytm, tak by „szyfrowanie" było nieco bardziej wymyślne. Możecie też zabawić się w grupie w „odszyfrowywanie" tekstów, które przygotowali inni.

Ćwiczenie 8-11

1. Utwórz algorytm do „odszyfrowania" informacji przetworzonej za pomocą algorytmu 17.
2. Utwórz algorytm uniwersalny, umożliwiający zarówno „szyfrowanie", jak i „odszyfrowywanie" informacji.

Zadanie 8-13

Zadanie to dotyczy bardzo specyficznej zmiany informacji, ponieważ chodzi raczej o jej formę niż o treść, skoro plik wyjściowy ma zawierać dokładnie te same liczby co plik wejściowy, tylko mają być zapisane po dwie w wierszu, a nie po jednej, jak w podanym przykładzie.

```
Plik wejściowy          Plik wyjściowy

123                     123;261,8
261,8                   2455;356,62
2455                    24;529
356,62
24
529
```

Zadanie takie można rozwiązywać dwiema metodami:

1. Odczytywać plik znak po znaku wyszukując znaki oznaczające koniec linii i numerując na tej podstawie linie. Jako znak końca linii występują zazwyczaj pary znaków o kodach: 13 (powrót karetki) i 10 (nowa linia). Takie pary znaków, które kończą linie o numerach nieparzystych, należy zastępować znakiem średnika, a pozostałych nie zmieniać.
2. Odczytywać liczby znajdujące się w poszczególnych wierszach. Po odczytaniu dwóch liczb należy je zapisać w jednej linii tworzonego pliku, oddzielając od siebie znakiem średnika.

Nowym elementem jest rozróżnienie działania w zależności od tego, czy numer odczytanej liczby jest parzysty czy nieparzysty. Zauważ, że wystarczy utworzyć pomocniczy licznik zliczający do dwóch, np. przyjmujący na przemian wartości 0 i 1 po napotkaniu kolejnego znaku końca linii (wersja 1) lub odczytaniu kolejnej liczby (wersja 2), i uzależniać postępowanie od stanu tego licznika.

Pierwsza z wymienionych metod nadaje się do przekształcania dowolnych plików tekstowych, mogących zawierać w poszczególnych wierszach nawet po kilka liczb, a także tekst, natomiast druga jedynie do plików zawierających po jednej liczbie w linii.

Zadanie 8-14

Zadanie to omówimy tu w zarysie, i uzupełnimy w dalszej części rozdziału po przedstawieniu typowych operacji na tekstach.

Koncepcja rozwiązania zakłada odczytywanie całych wierszy z pliku utworzonego poleceniem DIR w sposób opisany w rozdz. 2.4.7. Użyta opcja /S zapewnia przedstawienie informacji o dysku, plikach i katalogach w strukturze widocznej na podanym przykładzie (wiersze puste zostały tu pominięte; możesz utworzyć podobny plik przedstawiający zawartość załączonej dyskietki i porównać).

```
 Volume in drive B is ELEM_INF
Directory of B:\
TEKSTY       <DIR>          01-23-95  11:05a
CZYTAJ                 31 04-19-94  12:35a
       2 file(s)              31 bytes
Directory of B:\TEKSTY
.            <DIR>          01-23-95  11:05a
..           <DIR>          01-23-95  11:05a
MACIEK       <DIR>          01-23-95  11:05a
```

```
MOJPLIK              202 11-02-93   11:15p
MOJPLIK2 TXT          17 11-02-93   11:15p
        5 file(s)            219 bytes
Directory of B:\TEKSTY\MACIEK
.            <DIR>          01-23-95   11:05a
..           <DIR>          01-23-95   11:05a
        2 file(s)              0 bytes
Total files listed:
        9 file(s)            250 bytes
                       1 455 104 bytes free
```

Poszczególne wiersze różnią się od siebie na tyle, że można zidentyfikować rodzaj wiersza i odczytywać informację tylko z odpowiednich wierszy. Można więc wyróżnić: wiersz z etykietą (uwaga, dyskietka może nie mieć etykiety i wtedy w wierszu tym jest tekst: `Volume in drive B has no label`), wiersz z nazwą katalogu (podkatalogu), wiersz z nazwą podkatalogu, wiersz z nazwą pliku, wiersz ze standardowym odsyłaczem do katalogów na innych poziomach (z „nazwą" w postaci jednej lub dwóch kropek), wiersz podsumowujący dany katalog, wiersze końcowe podsumowujące całą dyskietkę, no i wiersze puste.

Gdy przyjrzysz się uważnie poszczególnym wierszom, to wskażesz sposoby rozróżnienia, o jaki wiersz chodzi. Na przykład wiersze zawierające nazwy plików lub katalogów można rozpoznać go po obecności daty i godziny (a więc np. po dwukropku na 40 miejscu w wierszu); można je rozróżnić między sobą albo po nazwie DIR występującej tylko przy katalogach (zajmującej miejsca od 15 do 17), albo po liczbie bajtów podawanej tylko przy plikach (zajmującej na pewno miejsce 26 w wierszu).

Uwaga. Liczby bajtów mogą być podawane w amerykańskim formacie finansowym, tj. z przecinkami oddzielającymi tysiące, miliony (podana wyżej liczba wolnych bajtów na dyskietce byłaby zapisana w postaci 1,455,504; patrz też rozdz. 2.3.3).

W algorytmie będą stosowane takie działania, jak: odczytanie wiersza tekstu, rozpoznanie końca pliku odczytywanego, analiza typu wiersza, działania polegające na wycięciu z wiersza fragmentu (fragmentów) tekstu — w wariantach odpowiednich dla poszczególnych typów wierszy, złożenie z zapamiętanych wyciętych fragmentów tekstu oraz z przyjętych separatorów (np. cudzysłowów i przecinków) wiersza tekstu przeznaczonego do zapisania w tworzonym pliku oraz zapisanie wiersza w tym pliku. Powiązanie tych działań między sobą jest oparte na schemacie podobnym do już stosowanych.

Algorytm 18

Krok 1. Przygotuj plik PLIKDIR do odczytu od początku oraz plik WYNIK do zapisu.

Krok 2. Sprawdź, czy możesz odczytać jeszcze jeden wiersz z pliku PLIKDIR. Jeżeli nie, to przejdź do kroku 7.

Krok 3. Odczytaj wiersz. Rozpoznaj typ wiersza. Jeżeli nie jest to żaden z typów wymienionych w krokach 4, 5 lub 6, to przejdź do kroku 2.

Krok 4. Jeżeli wiersz opisuje etykietę, to ją odczytaj i zapamiętaj oraz przejdź do kroku 2.

Krok 5. Jeżeli wiersz zawiera nazwę katalogu, to ją odczytaj i zapamiętaj oraz przejdź do kroku 2.

Krok 6. Jeżeli wiersz zawiera opis pliku, to: odczytaj poszczególne składniki i zapamiętaj, utwórz z zapamiętanych informacji oraz z separatorów wiersz tekstu i zapisz go w pliku WYNIK. Przejdź do kroku 2.

Krok 7. Zamknij pliki PLIKDIR oraz WYNIK i zakończ działanie.

Sposób realizacji wymienionych działań łatwiej Ci będzie zaplanować po zapoznaniu się z typowymi operacjami na tekstach realizowanymi w językach (systemach) programowania. Samą koncepcję możesz już rozważyć, wyobrażając sobie, że poszczególne znaki każdego wiersza zapisujesz w tablicy i do każdego znaku masz dostęp, a z poszczególnych znaków odczytanych z tablicy możesz składać teksty i tymi tekstami operować. Wygodniej jednak będzie posłużyć się innymi typami zmiennych niż tablica znaków.

Zadanie 8-15

Tablica z nazwami plików może być polem większej tablicy bazy danych, zawierającej jeszcze inne dane o plikach, np. rozmiar, data, godzina, nazwa katalogu. Tu potraktujemy ją jednak jak tablicę zawierającą w każdym wierszu jeden element tekstowy — nazwę pliku. Nie musimy w tej chwili czynić odniesień do systemu programowania: czy byłby to system uniwersalny, jak Turbo Pascal, czy związany z konkretnym systemem obsługi baz danych w rodzaju dBase lub FoxPro. Tego typu decyzję można podjąć później. Teraz skupimy się na samym algorytmie. Załóżmy więc, że w tablicy jest N wierszy z nazwami plików i że nie są one ułożone po kolei. Jak możemy stwierdzić, które z nich występują wielokrotnie? Widać dwie drogi postępowania:

1. Ułożyć nazwy w porządku alfabetycznym, a później ograniczyć porównywanie nazw do bezpośrednich sąsiadów; znalezienie innej

nazwy w kolejnym wierszu uporządkowanej tablicy oznaczałoby zakończenie poszukiwań.

2. Dla każdego pliku przeszukać całą tablicę, zliczając wiersze z identycznymi nazwami.

Jeżeli możemy liczyć na uporządkowanie tablicy przez program obsługi baz danych, to wystarczy ograniczyć się do drugiej części pierwszego sposobu postępowania, tzn. sprawdzenia nazw sąsiednich. Jeżeli byłyby trzy pliki lub więcej o takiej samej nazwie, to każdy z nich zostałby znaleziony jako bezpośredni sąsiad poprzedniego.

Nasz algorytm wyobraźmy sobie w postaci uniwersalnej, stosowanej wielokrotnie przy różnych położeniach początkowych k (numerach wiersza tablicy), od których dalsze porównywanie byłoby rozpoczynane. Wynikiem działania algorytmu jest liczba wierszy tablicy zawierających nazwy identyczne z umieszczoną w wierszu początkowym. Nazwy będziemy oznaczać jako NAZWA(m), gdzie m jest numerem wiersza tablicy. Do numerowania wiersza bieżącego użyjemy zmiennej NUMER, a wiersza początkowego — NUMERPOCZ; liczbę znalezionych identycznych wierszy oznaczymy jako IDENT.

Algorytm 19

Krok 1. Nadaj zmiennej NUMERPOCZ wartość k, zmiennej NUMER wartość $k + 1$, a zmiennej IDENT wartość 1.

Krok 2. Jeżeli NUMER $> N$, to zakończ działanie.

Krok 3. Jeżeli NAZWA(NUMER) różni się od NAZWA(k), zakończ działanie.

Krok 4. Zwiększ wartość zmiennych IDENT i NUMER o 1 i przejdź do kroku 2.

Zauważ, że najpierw sprawdza się w kroku 2, czy w ogóle istnieje kolejny wiersz tablicy NAZWA, a dopiero kiedy warunek ten jest spełniony, odczytuje się w kroku 3 jego zawartość i dokonuje porównania.

Jeżeli przedstawiony algorytm potraktować jako funkcję zależną od argumentu oznaczającego numer wiersza tablicy NAZWA, nazwijmy ją FIDENT(k), to algorytm przeszukiwania całej tablicy NAZWA miałby postać:

Algorytm 20

Krok 1. Nadaj zmiennej WIERSZ wartość 1.

Krok 2. Jeżeli WIERSZ $> N$, to zakończ działanie.

Krok 3. Oblicz funkcję FIDENT(WIERSZ) i jej wartość nadaj zmiennej ID.

Krok 4. Jeżeli ID > 1, to podaj komunikat zawierający nazwę pliku NAZWA(WIERSZ) oraz liczbę wystąpień ID.

Krok 5. Zwiększ wartość zmiennej WIERSZ o ID i przejdź do kroku 2.

Nowym elementem jest tu zwiększanie zmiennej WIERSZ, oznaczającej numer analizowanego wiersza tablicy NAZWA, nie o 1, lecz o ID, czyli o liczbę jednakowych nazw. W ten sposób nie tylko przyspiesza się działanie algorytmu, lecz również zabezpiecza się przed wielokrotnym analizowaniem tej samej nazwy (i wielokrotnym podawaniem komunikatu).

Druga droga nie wymaga uporządkowanego pliku. Do porównywania można stosować algorytm podobny do funkcji FIDENT, z modyfikacją polegającą na powracaniu z kroku 3 do kroku 2 bez kończenia działania (żeby uwzględnić wszystkie wiersze tablicy do ostatniego). Trudniej jest też uniknąć powtarzania tych samych porównań. Trzeba by w tym celu gromadzić informację o tych nazwach, dla których została już przeprowadzona analiza danych w tablicy. Można np. utworzyć pomocniczą tablicę ANAL o tej samej liczbie wierszy N co tablica NAZWA, zawierającą w swoich wierszach zmienne logiczne informujące, czy dana nazwa była już analizowana (na początku do wszystkich wierszy trzeba by wpisać Fałsz). Wpisanie do danego wiersza tablicy ANAL wartości Prawda mogłoby następować w czwartym kroku zmodyfikowanego algorytmu FIDENT. Na końcu pierwszego kroku tego algorytmu należałoby dopisać warunek:
„Jeżeli ANAL(NUMER) = Prawda, to zakończ działanie". Zauważ, że przy takiej zmianie funkcji FIDENT zmiany w algorytmie głównym sprowadzą się do wpisania wartości Fałsz do wszystkich wierszy tablicy ANAL. W praktyce różnica między algorytmami może okazać się znaczna pod względem czasu działania, gdy mamy do czynienia z dużymi tablicami. Widzisz, że „uporządkowanie" danych w tablicy, które przy zapoznawaniu się z programami obsługi baz danych mogłeś traktować jako zajęcie o znaczeniu estetycznym, ma tu aspekt praktyczny.

8.5. Tworzenie programu źródłowego — uwagi wstępne

Języki programowania wysokiego poziomu umożliwiają niemal bezpośrednie przeniesienie opracowanych algorytmów do programów źródłowych. Są jednak pewne kwestie, na które warto zwrócić uwagę, zanim przystąpisz do tworzenia i uruchamiania programów.

Ogólnie biorąc powinieneś liczyć się z tym, że musisz dostarczyć w programie źródłowym znacznie więcej informacji niż musiałbyś posłu-