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-

gując się arkuszem kalkulacyjnym lub programem zarządzania bazami danych.

8.5.1. Deklaracje, instrukcje, komentarze

Operacje wykonywane w algorytmie przenosisz do programu posługując się **instrukcjami**. Do typowych należy instrukcja przypisania wartości (podstawienia). Są też instrukcje umożliwiające realizację „pętli" w algorytmie. Wykonanie instrukcji może być uzależnione od spełnienia warunku (instrukcje warunkowe).

Zazwyczaj używasz w programie zmiennych do przechowywania obliczanych (przetwarzanych) wielkości. Instrukcje przypisania odnoszą się właśnie do zmiennych, którym nadają nowe wartości.

Twoim obowiązkiem jest umieszczenie w programie informacji o tym, jakich zmiennych używasz i jakiego typu. Do tej pory informację stwierdzającą, że jakaś zmienna jest typu numerycznego lub tekstowego, zamieszczaliśmy w opisie słownym algorytmu, a więc niejako „obok" niego. Teraz informacja tego rodzaju powinna się pojawić w programie źródłowym.

Podobnie, musisz poinformować system programowania, że nazwa, jakiej zamierzasz używać w programie odnosi się do tablicy, a nie do pojedynczej zmiennej, i podać rozmiary tej tablicy. Tego typu informacja jest określana mianem **deklaracji**.

Deklarowanie w programie rodzaju zmiennych jest konieczne, żeby program tłumaczący wiedział, ile miejsca w pamięci ma przydzielić na poszczególne zmienne i w jaki sposób ma te zmienne zapisywać i odczytywać.

Niektóre systemy programowania umożliwiają pominięcie typowych deklaracji, nadając nie zadeklarowanym przez użytkownika zmiennym typ standardowy (np. traktując je jako liczby rzeczywiste).

Języki programowania przewidują możliwość korzystania z różnych typów zmiennych (liczb, tekstów, tablic, rekordów, plików), a niekiedy nawet umożliwiają definiowanie samemu nowych typów. Część programu zawierająca deklaracje i definicje może więc być bardzo rozbudowana.

W systemach programowania użytkownik może korzystać z gotowych **funkcji** i **procedur**. Służą do tego odpowiednie instrukcje, zawierające przede wszystkim nazwę procedury, a zazwyczaj także jej parametry (argumenty).

Stosowałeś już funkcje matematyczne w arkuszach kalkulacyjnych; w programach czynisz to podobnie. System oblicza wartość funkcji dla danych argumentów, wykonując pomocniczy podprogram. Wartość tę możesz nadać zmiennej albo użyć w obliczanym wyrażeniu. Funkcje mogą mieć takie typy wartości, jak zmienne (liczby, napisy i in.).

Nowe dla Ciebie pojęcie procedury mogły Ci przybliżyć przykłady podane podczas tworzenia algorytmów do zadań 8-4, 8-6, 8-7 i 8-8. Procedura jest pomocniczym programem, którego działanie może zależeć od wartości parametrów. Ważną grupę procedur stanowią procedury wejścia, dotyczące odczytywania danych z klawiatury albo z pliku, oraz procedury wyjścia, służące do wysyłania ich na ekran (w trybie znakowym lub graficznym), do pliku lub bezpośrednio na drukarkę. Są też procedury graficzne, służące do tworzenia wykresów na ekranie, procedury przerywania programu i in.

Uwaga. Efekt stosowania niektórych gotowych procedur, np. dotyczących zmiany koloru ekranu (ogólnie parametrów środowiska programowania), może trwać także po zakończeniu programu. Do dobrych zwyczajów — zwłaszcza przy korzystaniu z komputerów wspólnie z innymi użytkownikami — należy zamieszczenie w programie instrukcji wywołania procedur, które przed zakończeniem działania przywrócą pierwotne wartości zmienianych parametrów. Wartości te warto zapisać, ponieważ program nie uruchomiony w pełni może zdążyć parametry zmienić, natomiast nie zdążyć ich przywrócić. Warto mieć odrębny program służący tylko do przywracania standardowych parametrów (i uruchamiać go pod koniec zajęć).

Możesz nie tylko używać gotowych funkcji i procedur w Twoich programach, lecz także je tworzyć.

Oprócz deklaracji i instrukcji, a także funkcji i procedur, Twój program może zawierać **komentarze**. Są to teksty, których program tłumaczący nie bierze pod uwagę. Są one w tym celu obejmowane odpowiednimi znakami. Z komentarzami zetknąłeś się już w pliku WSAD1.BAT (rozdz. 2.5.2).

Komentarze możesz pisać, choć nie musisz. Dobrze uczynisz, jeżeli przyjmiesz zasadę stosowania komentarzy w Twoich programach do:

- informowania, kto jest autorem programu lub jego modyfikacji;
- określenia celu działania i sposobu używania programu;
- określenia roli poszczególnych zmiennych, procedur, funkcji.

Dwie pierwsze informacje umieszczaj na początku tekstu programu, trzecia może być rozproszona: częściowo przy deklaracjach zmiennych, częściowo przy instrukcjach. Z tego względu dobrze jest deklarować wszystkie zmienne, nawet gdy nie wymaga tego system programowania (przydzielając zmiennej nie zadeklarowanej założony typ standardowy).

8.5.2. Typy danych

W językach programowania wyróżnia się odmienne typy danych, na których się operuje. Podstawowe z nich są podobne do stosowanych w arkuszach

kalkulacyjnych i bazach danych, ale występują tu różnice, na które należy zwrócić uwagę.

Liczby

W arkuszu kalkulacyjnym deklarowałeś, w jaki sposób liczba ma być prezentowana na ekranie, nie mając wpływu na jej postać w programie, natomiast w języku programowania decydujesz przede wszystkim o tym, jak ma być pamiętana w komputerze i co ma wyrażać.

Dwa podstawowe typy, to **liczby całkowite** (*integer*) i **liczby rzeczywiste** (*real*). Dla każdego typu określony jest **zakres liczb**, tzn. ich wartości maksymalne i minimalne. Na przykład dla liczb całkowitych przedstawianych za pomocą dwóch bajtów typowy zakres wynosi od –32768 do 32767.

Ogólnie biorąc dobrze jest stosować w programie liczby całkowite, jeżeli charakter danych i obliczeń na to pozwala. Dane zajmują wówczas mniej miejsca w pamięci (w niewielkich programach argument ten może mieć znaczenie drugorzędne), obliczenia są szybsze, a co może być szczególnie ważne — są wykonywane dokładnie. Możesz być zaskoczony tym ostatnim argumentem, jeżeli myślałeś, że komputer liczy dokładnie i nigdy się nie myli. Nawet jeżeli komputer bardzo rzadko „myli się" z powodu nieprawności technicznej, to jednak otrzymywane przy jego użyciu wyniki obliczeń mogą być niedokładne, a niekiedy nawet całkiem błędne. Występowanie błędów jest związane z niedokładnym przedstawianiem w komputerze liczb rzeczywistych i podobnie niedokładnym wykonywaniem na nich działań.

Dane będące liczbami rzeczywistymi przedstawia się w programie za pomocą liczb typu rzeczywistego, przedstawienie to jednak może być niedokładne nawet wtedy, gdy dana liczba mieści się w zakresie. Nie powinno Cię to dziwić, ponieważ wiesz, że za pomocą ułamka dziesiętnego o ograniczonej długości nie każdą liczbę możesz przedstawić dokładnie; np. nie przedstawisz dokładnie liczb 1/3 i $\sqrt{2}$ ani przy trzech cyfrach po przecinku, ani przy dziesięciu, ani przy stu, co najwyżej przy dłuższych liczbach błąd będzie mniejszy. Wprawdzie liczby w komputerach są zapisywane za pomocą cyfr dwójkowych, a nie dziesiętnych, ale długość zapisu liczb (liczba bajtów) jest też skończona, i niektórych liczb w systemie dwójkowym nie da się przedstawić dokładnie.

Niedokładności wiążą się także z wykonywaniem obliczeń. Na przykład mnożąc dwa ułamki dziesiętne z trzema cyframi po przecinku otrzymujesz ułamek z sześcioma cyframi po przecinku; gdy musisz go zapisać ograniczając się do trzech miejsc po przecinku, popełniasz pewien błąd. Podobnie nie-

które obliczenia wykonywane na liczbach zapisanych w systemie dwójkowym mogą być obarczone błędem. Błędy mogą być wnoszone podczas wykonywania operacji arytmetycznych na liczbach rzeczywistych, obliczania wartości funkcji (np. funkcji trygonometrycznych).

Występowanie błędów związanych ze sposobem przedstawiania w komputerze liczb rzeczywistych i wykonywaniem na nich działań jest nie do uniknięcia, można natomiast zmniejszyć wpływ tych błędów na ostateczny wynik.

Zamiast podstawowych typów zmiennych możesz zazwyczaj użyć typów podobnych, lecz zapisywanych za pomocą większej ilości bajtów i mających w związku z tym większy zakres lub precyzję, np. „długich" liczb całkowitych (*longinteger*) oraz liczb rzeczywistych umożliwiających „podwójną" precyzję obliczeń (*double precision*). Licz się z tym, że obliczenia na liczbach zapisywanych za pomocą większej liczby bajtów trwają dłużej niż na liczbach o standardowej długości.

Uwaga. W systemie Turbo Pascal możliwość używania niektórych typów liczb zależy od wyposażenia komputera w koprocesor. Zależy od tego także dokładność wykonywania niektórych operacji. Są programy symulujące koprocesor, zwane emulatorami. Także system Turbo Pascal zawiera program tego typu i użytkownik może zdecydować o użyciu go (jednak program ten nie daje wszystkich możliwości dostępnych przy rzeczywistym koprocesorze).

Z wyborem typu liczby może się wiązać konieczność drobnych modyfikacji algorytmów. Warunek zakończenia obliczeń był czasem formułowany jako warunek równości: zmienna służąca do numerowania obiegów pętli musiała osiągnąć każdą kolejną wartość całkowitą. Jeżeli zmienną odgrywającą taką rolę zadeklarujesz jako zmienną całkowitą, to możesz bezpiecznie posługiwać się podobnymi warunkami (jeżeli tylko nie wykraczają poza zakres liczb całkowitych danego typu). Przy stosowaniu do obliczeń liczb rzeczywistych musisz się liczyć z możliwością niedokładności. Na przykład liczba, która powinna być równa 400, będzie równa 399,9999. Ta drobna niedokładność może sprawić, że warunek zakończenia obliczeń nie zadziała. Z tego względu lepiej jest warunek taki zmienić z równości na nierówność (patrz rozwiązanie zastosowane w algorytmie 11).

Uwaga. W językach programowania jest stosowany zapis ułamków dziesiętnych z kropką zamiast przecinka. Dotyczy to zapisu stałych w tekście programu, a także odczytywania liczb z urządzeń wejściowych i ich przedstawiania w urządzeniach wyjściowych.

Dane tekstowe — łańcuchy

Zazwyczaj dostępne są typy zmiennych służące do przedstawiania pojedynczych znaków (*character*), oraz do przedstawiania całych napisów, zwanych **łańcuchami** (*string*). Operacje wykonywane na łańcuchach umożliwiają m.in. wycięcie fragmentu napisu, połączenie napisów itp.

Teraz łatwiej wyobrazisz sobie wykonanie szczegółowych działań podczas realizacji algorytmu 18 (do zadania 8-14). Zawsze z odpowiedniego wiersza tekstu możesz wyciąć fragmenty zawierające poszczególne interesujące Cię informacje (etykieta dyskietki, nazwa pliku, rozszerzenie, rozmiar). Możesz też upewnić się, czy dany wiersz zawiera potrzebną informację. Wyciętą informację możesz zgromadzić i zapisywać w odpowiedniej kolejności, np. zamieszczać etykietę dysku na początku każdego wiersza tworzonego pliku; możesz też wstawiać właściwe separatory pomiędzy teksty odpowiadające poszczególnym polom bazy danych (np. cudzysłowy, przecinki).

Dane proste i złożone

W językach programowania operuje się często tablicami zawierającymi zmienne. Tablice takie są nazywane w matematyce macierzami (*array*). W przypadku tablic trzeba podać liczbę wymiarów (jeden, dwa, trzy lub więcej) oraz zakres indeksów. Elementami tablic mogą być liczby, teksty i inne zmienne.

Można też operować rekordami i tworzyć własne bazy danych.

8.5.3. Wprowadzanie danych i wyprowadzanie wyników

Dane do obliczeń mogą być umieszczone od razu w programie (mają wówczas charakter stałych), a mogą też być wczytywane. Podstawowym urządzeniem służącym do wprowadzania danych jest klawiatura; ponadto dane mogą być odczytywane z plików dyskowych. Obsługa innych urządzeń wejściowych jest trudniejsza albo nawet nie zawsze możliwa z danego języka programowania.

Odczytywanie danych z pliku dyskowego wymaga uważnego przygotowania programu i danych, żeby odczyt przebiegał prawidłowo, zgodnie z życzeniem autora. Wiesz już o możliwości odczytu plików znak po znaku, ale są też i inne tryby odczytywania danych.

Teksty można wczytywać do zmiennych łańcuchowych (trwa to krócej). Warto mieć wtedy pewność, że długość łańcucha znaków w pliku (ciągu znaków zakończonego znakiem końca linii) nie przekracza maksymalnej długości łańcucha w języku programowania (np. 255 znaków), inaczej część danych może zostać stracona.

Liczby można odczytywać jako liczby. Trzeba określić tzw. **format odczytu** liczb. Są formaty wymagające dokładnego pozycjonowania liczb w pliku (z dokładnością do jednej spacji), są też tzw. formaty swobodne umożliwiające odczytanie danych zapisanych w sposób nieregularny. Zwróć uwagę, że przy odczytywaniu liczb ważny jest także ich typ: całkowity lub rzeczywisty i podanie innego typu może prowadzić do błędów. Przy wprowadzaniu liczb rzeczywistych trzeba używać kropki dziesiętnej.

Wyniki działania programu są wyprowadzane przede wszystkim na ekran monitora. Oprócz trybu tekstowego używa się także trybu graficznego. Można wtedy samodzielnie tworzyć wykresy. Ponadto wyniki mogą być zapisywane do pliku dyskowego, a także wysyłane bezpośrednio na drukarkę (pracującą w trybie tekstowym). Obsługa innych urządzeń wyjściowych wymaga lepszej znajomości systemu komputerowego.

8.5.4. Struktura programów. Podprogramy

Wiesz już, że możesz tworzyć procedury i funkcje (a nie tylko korzystać z gotowych). Języki programowania umożliwiają wyodrębnienie fragmentów programu w postaci **podprogramów**, które zależnie od sposobu działania mają postać procedur (*subroutine, procedure*) albo funkcji (*function*). Wiesz również, że wyodrębnienie podprogramów jest korzystne ze względu na nakład pracy przy tworzeniu algorytmu i zapisywaniu programu, ale nie jest to jedyna korzyść. Obejmuje ona także uruchamianie programu. W razie stwierdzenia błędu poprawia się jeden podprogram. Ten jeden podprogram wszędzie zachowuje się tak samo, testowanie programu wymaga więc mniejszego nakładu pracy. Jest też inny wzgląd: przejrzystość programu głównego, który ograniczony do zasadniczych działań, jest bardziej czytelny (łatwiej więc jest analizować jego działanie i wyszukiwać ewentualne błędy podczas uruchamiania).

Zapamiętaj!

W programach o większym stopniu złożoności fragmenty programu stanowiące funkcjonalną całość zapisujemy w postaci podprogramów.

Podprogramy, aczkolwiek muszą być użyte w programie, gdzie są wywoływane, a nie samodzielnie, mają jednak charakter samodzielnych jednostek. Na przykład, tworzenie programów w ten sposób, że są dzielone na podprogramy, umożliwia też podzielenie pracy w zespole, co w przypadku

większych programów może być ważne. Podprogramami przygotowanymi przez Ciebie możesz podzielić się z innymi; możesz też użyć podprogramu udostępnionego Ci przez autora (zachowaj w nagłówku podprogramu komentarz dotyczący jego autorstwa).

8.5.5. Uruchamianie programów

W Twoich programach, zwłaszcza tych bardziej rozbudowanych, mogą być błędy. Często zdarza się, że bezpośrednio po napisaniu (zakodowaniu) program nie działa poprawnie. Systemy programowania są wyposażane w różne pomoce mające ułatwić użytkownikowi postępowanie z programem w tej fazie jego tworzenia. Pomoce te polegają na umożliwieniu:

- śledzenia biegu programu;
- wykonywania programu fragmentami, a nawet krokowo — instrukcja po instrukcji;
- odczytywania wartości wybranych zmiennych po każdym zatrzymaniu (z możliwością nadawania im nowych wartości i kontynuowania biegu programu).

Programy odpowiedzialne za te funkcje noszą ogólną nazwę **debugerów** (usuwacz błędów; w angielskim żargonie programistycznym błędy w programie określono mianem *bug*'ów, czyli pluskiew). Najważniejszą funkcją, z jakiej użytkownik może korzystać, jest wykonywanie programu instrukcja po instrukcji, z możliwością kontynuowania śledzenia biegu programu wewnątrz procedur i funkcji zdefiniowanych przez użytkownika. Jest też możliwość określania miejsc w programie, w których program ma się zatrzymać i umożliwić działania użytkownikowi.

Programy powinny być przetestowane, żeby mieć (większą) pewność co do ich poprawnego działania. Programy profesjonalne przechodzą całe cykle takich prób — najpierw u ich twórców, a potem u użytkowników — zanim będą skierowane na rynek, a i tak nie są wolne od usterek. Jeżeli miałbyś swój program komuś ofiarować do użytkowania, powinieneś go sprawdzić na różnorodnych danych.

Zapamiętaj!
Programy powinny być testowane.

Pytania

1. Jaki jest schemat rozwiązywania problemów w informatyce?
2. Czym jest programowanie w informatyce?
3. Czym się różni język programowania od systemu programowania?
4. Wymień etapy tworzenia programu komputerowego.
5. Co to jest algorytm?
6. Wymień znane Ci sposoby przedstawiania algorytmów.
7. Wymień typowe elementy sieci działań.
8. Podaj znane Ci typy danych w systemach programowania.
9. Jakie widzisz korzyści ze stosowania podprogramów?
10. Jakie znasz formy pomocy w uruchamianiu programów, oferowane przez systemy programowania?