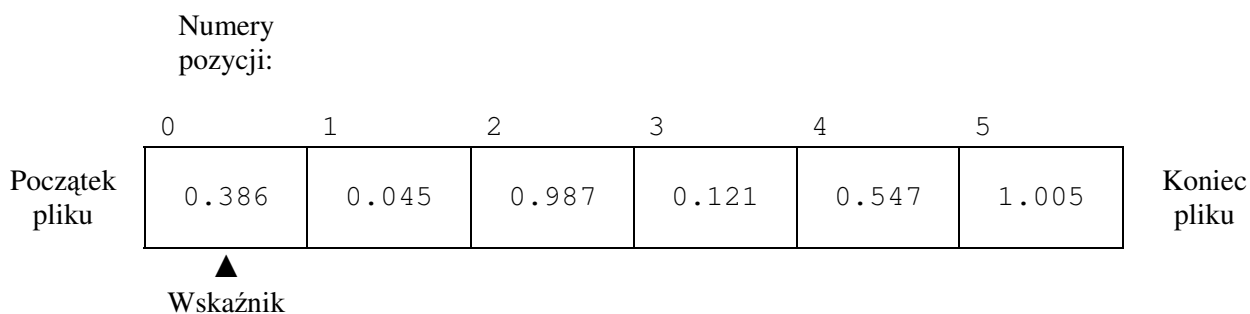


14. OPERACJE NA PLIKACH ELEMENTOWYCH

14.1. Struktura pliku fizycznego elementowego

Plik fizyczny jest to zbiór danych w pamięci zewnętrznej, stanowiący pewną całość i posiadający określoną strukturę. Struktura pliku *elementowego* (zwanego również *zdefiniowanym*) ma postać sekwencji elementów jednakowego typu, co ilustruje rysunek 14.1. Typem elementów może być każdy typ Turbo Pascala, zarówno prosty jak strukturalny, z wyjątkiem typu obiektowego. Liczba elementów pliku jest dowolna; ogranicza ją jedynie rozmiar dostępnej pamięci.



Rys. 14.1. Plik zawierający sześć elementów typu *Real*

Liczba elementów pliku nazywa się *rozmiarem* pliku. Rozmiaru pliku elementowego nie należy mylić z wielkością zajmowanego przez niego obszaru pamięci. Na przykład rysunek 14.1 przedstawia plik elementowy o rozmiarze 6, który zajmuje 36 bajtów pamięci, ponieważ do przechowania każdego elementu typu *Real* potrzeba sześciu bajtów..

Bardzo ważnym pojęciem jest *wskaźnik pliku*. Jego bieżąca pozycja wyznacza ten element, na którym zostanie dokonana kolejna operacja we/wy, to znaczy operacja zapisu danej do elementu pliku, lub odczytu danej, pamiętanej w elemencie pliku. Wskaźnik pliku zachowuje się podobnie, jak kursor na ekranie monitora,. Jak pamiętamy, kolejny znak jest pisany na ekranie w miejscu bieżącego położenia kursora. Po napisaniu znaku kursor samoczynnie przechodzi na następną pozycję ekranu. Podobnie, wskaźnik pliku po każdej operacji we/wy przesuwa się na następną pozycję pliku.

14.2. Ścieżka dostępu do pliku fizycznego

Plik fizyczny ma nazwę i umieszczony jest za pośrednictwem wybranego napędu dyskowego na dysku twardym lub innym nośniku danych. Dostępny na danym dysku obszar pamięci ze względów praktycznych dzielimy na katalogi i podkatalogi, podobnie jak książkę dzieli się na rozdziały i podrozdziały. Adres pliku ma postać łańcucha znaków zwanego *ścieżką dostępu*, w którym wyróżniamy trzy części: 1/ nazwę napędu, którą jest jedna z początkowych liter alfabetu, po której pisze się dwukropek, 2/ ścieżkę w drzewie katalogów i podkatalogów, prowadzącą do danego zbioru, 3/ nazwę zbioru. W Turbo Pascalu ścieżka dostępu może mieć nie więcej niż 79 znaków. Można ją zapamiętać w zmiennej typu *string*, lub użyć zmiennej specjalnie zdefiniowanego do tego celu standardowego typu Turbo Pascala o nazwie *Pathstr*. W tym ostatnim przypadku należy zadeklarować użycie modułu *Dos*, w którym ten typ zdefiniowano. Przykładem pliku fizycznego może być zapisany w pamięci dyskowej program wykonawczy Turbo Pascala, który nosi nazwę *Bp.exe*. W komputerze, na którym jest pisany ten tekst, ścieżka dostępu do tego pliku ma postać:

```
'C:\BP\BIN\BP.EXE'
```

Informację zawartą w tej ścieżce dostępu odczytujemy, jak następuje: Na dysku *C:* znajduje się katalog *Bp*. Z kolei w katalogu *Bp* występuje podkatalog *Bin*, w którym zapisano plik *Bp.exe*.

14.3. Zmienna plikowa

Zarządzanie pracą pamięci dyskowych, w których są umieszczone pliki fizyczne, jest jednym z zadań systemu operacyjnego komputera. Program napisany w Turbo Pascalu, w celu zapisywania danych w pliku fizycznym lub pobierania danych z pliku fizycznego, musi komunikować się z systemem operacyjnym. Program przesyła do systemu operacyjnego lub od niego otrzymuje dane, potrzebne do sterowania procesami tworzenia pliku i modyfikacji pliku. Te dane to na przykład ścieżka dostępu, rodzaj operacji na pliku, aktualny rozmiar pliku, stan buforów transmisji danych itp. W wymianie danych pomiędzy systemem operacyjnym a programem pośredniczy specjalna zmienna rekordowa zwana *zmienną plikową*. Jest to zmienna rekordowa o sześciu polach, z których każde pamięta inny rodzaj danych. Zmienna plikowa zajmuje w sumie 128 bajtów pamięci. Definicja typu takiej zmiennej ma ogólną postać:

```
type Nazwa_typu_plikowego = file of Typ_elementów_pliku;
```

Jak widać, w definicji tej stosuje się dowolnie wybrana nazwę pliku, a po słowach kluczowych *file of* pisze się nazwę typu, jaki reprezentują elementy pliku. Typ elementów musi być zdefiniowany wcześniej, o ile nie jest jednym z typów standardowych.

Po zdefiniowaniu typu plikowego, można zadeklarować jedną lub więcej zmiennych plikowych, w zależności od liczby plików, z którymi ma współpracować pisany przez nas program, na przykład:

```
var F1, F2: Nazwa_typu_plikowego;
```

Dla pliku z rysunku 14.1 odpowiedni zapis miałby postać:

```
type Tplik = file of Real;  
var F:Tplik;
```

Oczywiście nazwa *Tplik* została przyjęta arbitralnie.

14.4. Tworzenie nowego pliku fizycznego

Tworzenie nowego pliku przebiega w czterech kolejnych krokach, którymi są:

- skojarzenie zmiennej plikowej z plikiem fizycznym
- otwarcie pliku do zapisu
- zapis danych do pliku
- zamknięcie pliku.

Poniżej pokażemy, jak te kroki są wykonywane w Turbo Pascalu.

a/ Kojarzenie zmiennej plikowej z plikiem fizycznym

Celem tego działania jest przekazanie do systemu operacyjnego informacji o położeniu i nazwie pliku. Służy do tego standardowa procedura *Assign*, wywoływana, jak następuje:

```
Assign(F, S);
```

Pierwszym z jej argumentów jest zmienna plikowa *F*. Drugi argument *S*, to ścieżka dostępu do pliku fizycznego, który chcemy utworzyć, Ścieżka ta określa położenie i nazwę katalogu, w którym umieścimy nowy plik. Na końcu ścieżki dostępu umieszczamy nazwę nowotworzonego pliku. Należy pamiętać, że ścieżka dostępu nie może zawierać nazwy katalogu, który nie istnieje. Odpowiedni katalog należy utworzyć wcześniej.

Ten sam program może równocześnie pracować z wieloma różnymi plikami, kojarząc je z różnymi zmiennymi plikowymi. Jednak można równocześnie otworzyć nie więcej, niż piętnastie plików.

b/ Otwarcie pliku do pisania (lub ponownego pisania) danych

Przed przystąpieniem do zapisu elementów tworzonego lub na nowo zapełnianego danymi pliku, należy przeprowadzić operację zwaną *otwarcie pliku do zapisu*. Jest to zadaniem procedury *Rewrite*, której argumentem jest zmienna plikowa, wcześniej skojarzona z otwieranym plikiem. Odpowiednia instrukcja otwarcia pliku do zapisu ma postać:

```
Rewrite(F);
```

Jeżeli otwierany przez procedurę *Rewrite* plik już istnieje, to jego aktualna zawartość zostanie usunięta. W każdym przypadku użycia instrukcji *Rewrite* otwarty plik nie zawiera żadnego elementu, a jego wskaźnik zostaje ustawiony na początkowej pozycji o numerze 0.

Pamiętajmy, że można otworzyć tylko taki plik, który został uprzednio skojarzony z odpowiednią zmienną plikową za pomocą instrukcji *Assign*.. Jeżeli skojarzenia nie było, wystąpi błąd wykonania programu z komunikatem:

```
Error 102: File not assigned.
```

Pamiętajmy również, że nie wolno powtórnie kojarzyć pliku, który jest otwarty.

c/ Zapis danych do pliku

Po skojarzeniu nowotworzonego (lub *na nowo* tworzonego) pliku ze zmienną plikową, można do niego zapisać dane. Dane, które chcemy zapisać jako element pliku, umieszczane są czasowo w specjalnej zmiennej, którą tutaj nazwiemy *Bufor*. Typ tej zmiennej musi być taki sam, jak typ elementu pliku.

Do przeniesienia danej ze zmiennej *Bufor* do pliku fizycznego służy instrukcja:

```
Write(F, Bufor);
```

Jest to ta sama instrukcja *Write*, której używa się do wyprowadzania danych na ekran, ale w tym przypadku posiada ona dodatkowy argument, którym jest zmienna plikowa *F*. Zwróćmy uwagę, że przy pominięciu tego argumentu, wyjście danych zostanie domyślnie kojarzone z urządzeniem wyjściowym (monitorem) i dane będą wyprowadzane na ekran.

Na miejsce argumentu *Bufor* nie można podstawiać wyrażenia, ani stałej. Spowodowałoby to błąd kompilacji z komunikatem:

```
Error 20: Variable identifier expected.
```

Jak widać, dołączenie jednego elementu do pliku z reguły wymaga wykonania pary instrukcji, z których jedna nadaje wartość zmiennej *Bufor*, a druga przenosi tę wartość ze zmiennej do pliku.

d/ Zamknięcie pliku

Po wykonaniu operacji na pliku należy go zamknąć za pomocą instrukcji:

```
Close(F);
```

gdzie *F* jest, jak poprzednio, zmienna plikowa skojarzona z danym plikiem fizycznym. Oczywiście nie należy zamykać pliku, który nie jest otwarty, bo spowoduje to błąd wejścia/wyjścia z komunikatem:

```
Error 103: File not open.
```

W przykładzie 14.1 pokazano dwie kompletne procedury, tworzące pliki fizyczne o różnym typie elementów. Pierwsza z nich o nazwie *Pierwiastki_do_pliku* tworzy plik fizyczny o ścieżce dostępu 'A:\SQRTS' złożony ze stu pierwiastków kwadratowych kolejnych liczb naturalnych 1..100. Druga procedura nazwana *Tablica_do_pliku* początkowo wypełnia 1000-elementową tablicę liczbami losowymi z zakresu *Byte*. Następnie cała ta tablica, jako jedyny element typu tablicowego, zostaje zapamiętana w pliku fizycznym o ścieżce dostępu 'A:\TABLICA'. Zapamiętane dane będą

mogły być wykorzystane w dowolnym czasie przez każdy program, który za pośrednictwem swojej zmiennej plikowej skojarzy się z tym plikiem fizycznym.

Przykład 14.1. Procedury tworzące pliki fizyczne

```
program Ex14_1;
const C1='A:\SQRTS';
      C2='A:\TABLICA';
type Tab = array[1..100] of Byte;
      Tplik_1 = file of Real;
      Tplik_2 = file of Tab;

procedure Pierwiastki_do_pliku(S:string);
var I:Integer;
    F:Tplik_1;
    Bufor:Real;
begin
  Assign(F,S);           {Skojarzenie}
  Rewrite(F);           {Otwarcie do zapisu}
  for I:=1 to 100 do begin {Wpisywanie danych}
    Bufor:=Sqrt(I);
    Write(F,Bufor);
  end;
  Close(F);             {Zamknięcie}
end;

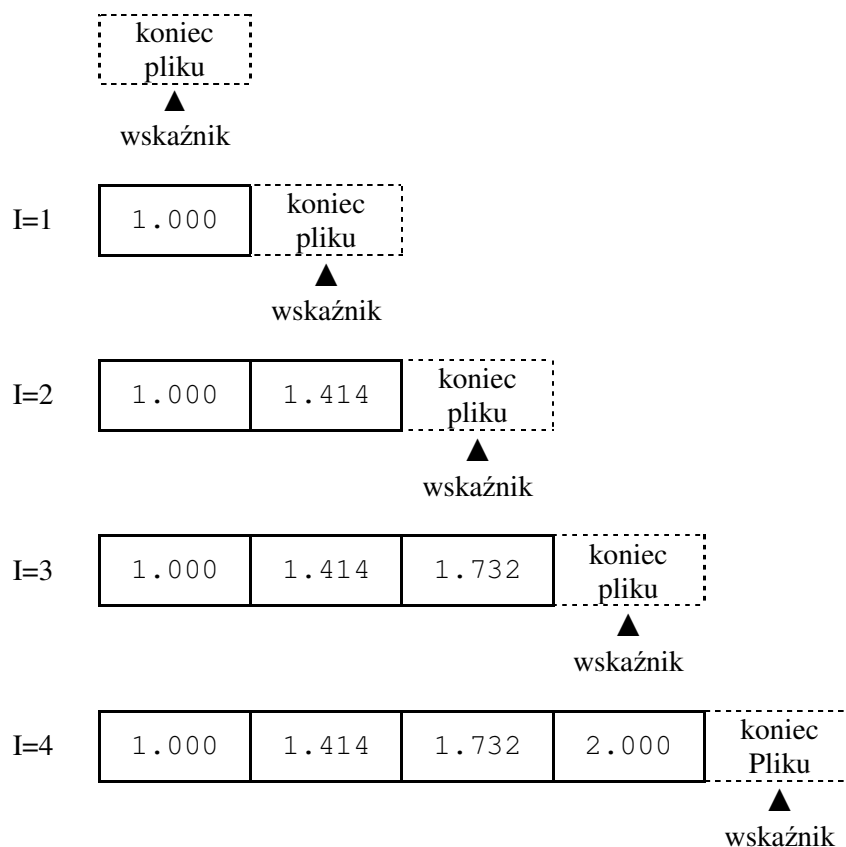
procedure Tablica_do_pliku(S:string);
var I:Integer;
    F:Tplik_2;
    Bufor:Tab;
begin
  Assign(F,S);           {Skojarzenie}
  Rewrite(F);           {Otwarcie do zapisu}
  Randomize;
  for I:=1 to 100 do     {Zapełnianie tablicy}
    Bufor[I]:=Random(256);
  Write(F,Bufor);       {Zapis tablicy do pliku}
  Close(F);             {Zamknięcie}
end;

begin
  Pierwiastki_do_pliku(C1);
  Tablica_do_pliku(C2);
end.
```

Przebieg procesu tworzenia pliku przez procedurę *Pierwiastki_do _pliku* ilustruje rysunek 14.2. Pokazano na nim stan pliku po czterech pierwszych krokach instrukcji

14.5. Odczyt zawartości pliku

Proces odczytu zawartości pliku obejmuje następujące etapy: a/ skojarzenie zmiennej plikowej z odpowiednim plikiem fizycznym, b/ otwarcie pliku do odczytu, c/ pobieranie kolejnych elementów do bufora (z ewentualnym zapamiętaniem lub przetwarzaniem odczytywanych z pliku danych), d/ zamknięcie pliku.



Rys. 14.2. Dołączanie kolejnych elementów do pliku

Etapy kojarzenia i zamykania pliku wykonuje się identycznie, jak przy tworzeniu nowego pliku, za pomocą procedur *Assign* i *Close* Natomiast do otwarcia pliku, który już istnieje, należy użyć instrukcji:

```
Reset (F) ;
```

gdzie *F* jest zmienna plikową, uprzednio skojarzoną z otwieranym plikiem. Instrukcja *Reset* otwiera plik zarówno do odczytu jak do zapisu. Oznacza to, że z tak otwartego pliku można odczytywać dane, lub wpisywać nowe wartości na wybrane pozycje pliku. Oczywiście, poprzednio pamiętane tam wartości zostaną utracone, jeżeli przedtem nie skopiowano ich do pomocniczych zmiennych programu. Podobnie, jak w przypadku instrukcji *Rewrite*, po otwarciu pliku przez instrukcję *Reset* wskaźnik pliku znajduje się na zerowej pozycji.

Należy pamiętać, że jeżeli zamiast instrukcji *Reset* użyjemy do otwarcia istniejącego pliku instrukcji *Rewrite*, to wszystkie przechowane w pliku dane zostaną utracone!

Przy próbie otwarcia pliku, którego nie ma na dysku pod adresem wskazanym przez ścieżkę dostępu, nastąpi błąd wykonania z komunikatem:

```
Error 2: File not found.
```

Odczyt danych z elementu pliku jest wykonywany za pomocą instrukcji:

```
Read (F, Bufor) ;
```

gdzie *F* jest zmienną plikową, z którą uprzednio skojarzono plik, a *Bufor* jest to zmienna o takim samym typie, jak typ elementu pliku. Instrukcja *Read* kopiuje wartość elementu, na którym aktualnie spoczywa wskaźnik pliku, do zmiennej buforowej *Bufor*. Po każdym odczycie elementu wskaźnik pliku przesuwają się automatycznie na kolejną pozycję, więc przy następnym wywołaniu instrukcji *Read* odczytany będzie następny w kolejności element pliku.

Przy odczycie danych z pliku elementowego trzeba posługiwać się instrukcją *Read*, a nie *Readln*. Jest to ta sama instrukcja, która służy do odczytu danych z klawiatury, tutaj jednak jej pierwszym argumentem jest zmienna plikowa. W przypadku pominięcia zmiennej plikowej instrukcja *Read* domyślnie skojarzy wejście danych z klawiaturą.

W przykładzie 14.2 pokazano dwie procedury, które odczytują zawartość plików utworzonych w programie z przykładu 14.1. Typowy sposób odczytywania kolejnych elementów pliku przedstawia procedura *Pierwiastki_z_pliku*. Sposób ten polega na zastosowaniu pętli *while*, kontrolowanej przez standardową funkcję *Eof*. Nazwa tej funkcji jest utworzona z pierwszych liter angielskich słów *end of file* (po polsku: koniec pliku). Argumentem funkcji jest zmienna plikowa. Funkcja *Eof* reaguje na położenie wskaźnika pliku. Jeżeli wskaźnik ustawi się na końcu pliku, lub plik jest pusty, to *Eof(F)=True*; w przeciwnym przypadku prawdziwa jest relacja *Eof(F)=False*. Chcąc odczytać wszystkie elementy pliku, wielokrotnie powtarzamy wywołanie instrukcji czytania elementu: *Read(F,Bufor)* tak długo, dopóki wskaźnik pliku, przesuwający się po każdym odczycie

Przykład 14.2. Odczytywanie danych z plików

```

program Ex14_2;
uses Crt;
const C1='A:\SQRTS'; C2='A:\TABLICA';
type Tab=array[1..100] of Byte;
      Tplik_1=file of Real;
      Tplik_2=file of Tab;

procedure Pierwiastki_z_pliku(S:string);
  var I:Integer; F:Tplik_1;
      Bufor:Real;
begin
  Assign(F,S);                               {Skojarzenie}
  Reset(F);                                   {Otwarcie do odczytu}
  while not Eof(F) do begin                 {Wpisywanie danych}
    Read(F,Bufor);
    Write(Bufor:8:4);
  end;
  Writeln;
  Close(F);                                   {Zamknięcie}
end;

procedure Tablica_z_pliku(S:string);
  var I:Integer; F:Tplik_2;
      Bufor:Tab;
begin
  Assign(F,S);                               {Skojarzenie}
  Reset(F);                                   {Otwarcie do zapisu}
  Read(F,Bufor);                             {Kopiowanie tablicy z pliku do bufora}
  for I:=Low(Bufor)to High(Bufor) do
    Write(Bufor[I]:4);                       {Wyprowadzenie na ekran}
  Writeln;
  Close(F);                                   {Zamknięcie}
end;

begin
  Clrscr;
  Pierwiastki_z_pliku(C1);
  Tablica_z_pliku(C2);
  Readln;
end.

```

o jedną pozycję, nie dojdzie do końca pliku. Wtedy działanie pętli powinno się zakończyć. Warunkiem tego jest przyjęcie przez wyrażenie po słowie *while* wartości *False*. Ponieważ jednak funkcja *Eof(F)* na końcu pliku przyjmuje wartość *True*, a nie *False*, trzeba tę wartość zanegować, stosując operator negacji *not*. Kompletna postać pętli *while*, przeznaczonej do kolejnego odczytu wszystkich elementów pliku i ich wyprowadzenia na ekran, ma postać:

```
while not Eof(F) do begin
  Read(F,Bufor);
  Write(Bufor);
end;
```

Druga z procedur o nazwie *Tablica_z_pliku*, jest prostsza, ponieważ założono, że plik przechowuje tylko pojedynczą tablicę. W tym przypadku, do odczytu danych nie trzeba stosować pętli – wystarczy pojedyncza instrukcja *Read*, która kopiuje tablicę pamiętaną w pliku do zmiennej tablicowej *Bufor*. Dopiero potem pętla *for* wyprowadza elementy tablicy *Bufor* na ekran. Zauważmy, że program pracuje z dwoma różnymi plikami i dlatego stosuje dwie różne zmienne plikowe, które są różnych typów (plik wartości *Real* oraz plik tablic *Tab*).

Warto przedyskutować sposób wyboru argumentów procedur, operujących na plikach. Oczywistym argumentem wejściowym jest ścieżka dostępu *S*. Dzięki wprowadzeniu tego argumentu, procedura staje się podprogramem uniwersalnym, zdolnym współpracować z plikami o różnych nazwach i rozmaitej lokalizacji. Procedury i funkcje plikowe używają często jako argumentu zmiennej plikowej. Natomiast w obu procedurach z przykładów 14.1, 14.2, a także w procedurze z przykładu 14.3, zmienna plikowa nie jest argumentem, lecz została zadeklarowana jako *zmienna lokalna*. Taką deklarację zastosowano z tego powodu, że proces pracy z plikiem, realizowany w ciele tych procedur, jest *kompletny* – występują wszystkie jego etapy: skojarzenie, otwarcie, przetwarzanie danych, zamknięcie. Wobec tego nie ma potrzeby przekazywania danych ze zmiennej plikowej do programu głównego i argument plikowy staje się zbędny. W innych przypadkach, gdy procedura realizuje tylko jeden lub niektóre etapy pracy z plikiem, konieczne jest przekazywanie zawartości zmiennej plikowej do programu głównego. Wtedy kolejno wywoływane podprogramy plikowe muszą posiadać argumenty plikowe, aby zmieniać i wzajemnie sobie przekazywać dane, pamiętane w globalnej zmiennej plikowej. Oczywiście, argument plikowy ma wtedy charakter argumentu *wyjściowego* i musi być poprzedzony słowem *var*. Przykład procedury z argumentem plikowym pokażemy w podrozdziale 14.7.

14.6. Sporządzanie kopii pliku

Poniżej w przykładzie 14.3 pokazano program z procedurą własną *Kopiuj_plik*. Procedura ta tworzy kopię pliku, do którego prowadzi ścieżka dostępu *S1*, w miejscu określonym przez ścieżkę *S2*. Obie ścieżki są argumentami wejściowymi procedury. W czasie kopiowania są otwarte oba pliki: pierwotny, z którego odczytujemy dane i wtórny, nowo tworzony, do którego przepisujemy dane z pliku pierwotnego. Dlatego trzeba użyć dwóch zmiennych plikowych *F1*, *F2*. Każdą z nich kojarzymy z innym plikiem fizycznym za pomocą dwóch instrukcji *Assign*. Następnie otwieramy plik pierwotny do odczytu za pomocą instrukcji *Reset(F1)*. Do otwarcia pliku wtórnego, który dopiero powstaje, trzeba użyć instrukcji *Rewrite(F2)*; Potem rozpoczyna się pętla *while*. W każdym cyklu jej działania następuje odczyt jednego elementu z pliku pierwotnego (instrukcja *Read*) i jego zapis do pliku wtórnego (instrukcja *Write*), w czym pośredniczy zmienna buforowa *Bufor*. Po przekopiowaniu wszystkich elementów pętla *while* kończy swoje działanie. Wtedy oba pliki zostają zamknięte za pomocą dwóch instrukcji *Close*.

Omówiona procedura jest przykładem jednoczesnej pracy z dwoma plikami. Zauważmy, że zmienne plikowe *F1*, *F2* są tutaj zmiennymi lokalnymi, a nie argumentami procedury. Wynika to z faktu, że proces przetwarzania obu plików w ciele procedury jest kompletny – wobec tego nie trzeba przekazywać danych zawartych w zmiennych plikowych do programu głównego.

Przykład 14.3. Sporządzanie kopii pliku

```

program Ex14_3;
const C1='A:\SQRTS';
        C2='C:\MOJE_PROGRAMY\SQRTS_1';
type Tplik=file of Real;
procedure Kopiuj_plik(S1,S2:string);
    var F1,F2:Tplik;
        Bufor:Real;
begin
    Assign(F1,S1);
    Assign(F2,S2);
    Reset(F1);
    Rewrite(F2);
    while not Eof(F1) do begin
        Read(F1,Bufor);
        Write(F2,Bufor);
    end;
    Close(F1);
    Close(F2);
end;
begin
    Kopiuj_plik(C1,C2);
end.

```

14.7. Programowa kontrola błędów wejścia/wyjścia

Otwarcie pliku, zarówno do zapisu, jak do odczytu, może nie udać się, co zostaje ujawnione w postaci błędu wykonania programu. Tego rodzaju błąd zalicza się do grupy błędów wejścia/wyjścia. System operacyjny przerywa wówczas nieodwracalnie proces wykonania programu i na ekranie pojawia się odpowiedni komunikat. Błąd powstaje na przykład wtedy, gdy nie skojarzono pliku ze zmienną plikową, lub gdy w napędzie nie ma dysku, lub gdy dysk jest zabezpieczony przed zapisem, albo też gdy nie ma na dysku pliku o podanej nazwie i lokalizacji. Po identyfikacji przyczyny błędu przerwany program trzeba uruchamiać od początku.

Przejmując w programie kontrolę nad błędami we/wy, można uniknąć konieczności rozpoczynania pracy programu od nowa. Ponadto można zapewnić użytkownikowi taką treść lub formę komunikatu o rodzaju błędu, jaka jest w danej sytuacji najwygodniejsza. Można na przykład wyprowadzać komunikat w języku polskim. Przejęcie kontroli wymaga przede wszystkim wyłączenia systemowej kontroli we/wy. W tym celu przed odpowiednim fragmentem programu piszemy dyrektywę $\{I-\}$. Po tej części programu, która realizuje własną kontrolę błędów, należy przywrócić kontrolę systemową, stosując dyrektywę $\{I+\}$.

Aby stwierdzić, czy operacja na pliku przebiegła poprawnie, wykorzystujemy standardową funkcję o nazwie *IOresult* (skrót od ang. *Input Output Result*). Funkcja ta zwraca liczbę całkowitą w formacie *Word*, która jest równa 0, gdy operacja przebiegła bezbłędnie. W przeciwnym przypadku funkcja zwraca numer błędu. Na przykład przy braku dyskietki w napędzie zwróci wartość 152.

Poniżej, w przykładzie 14.4, pokazano procedurę, która otwiera plik, skojarzony z jej argumentem plikowym. Przed słowem *begin*, od którego rozpoczyna się ciało procedury, wyłączamy kontrolę systemową, stosując dyrektywę $\{I-\}$. Kontrolę tę włączamy ponownie po słowie *end*, zamykającym ciało procedury. Plik może zostać otwarty do zapisu, gdy argument *R* ma wartość *True*, lub do odczytu, gdy wartość *R* jest równa *False*.

Po próbie otwarcia pliku zostaje wywołana funkcja *IOresult*, która przekazuje swoją wartość do zmiennej *B*. Jeżeli otwarcie przebiegło pomyślnie (tj. $B=0$), instrukcja *Exit* kończy działanie

procedury. W przypadku, gdy wystąpił błąd, instrukcja *case* wybiera właściwy komunikat, w zależności od wartości zmiennej *B*. Teraz procedura czeka na reakcję użytkownika. Dzięki zastosowaniu pętli *repeat*, po usunięciu przyczyny błędu (na przykład po włożeniu dysku do pustego napędu), można powtórzyć próbę otwarcia pliku, naciskając dowolny klawisz z wyjątkiem *<Esc>*. Jeżeli nie udaje się otworzyć pliku, można przerwać działanie programu, naciskając klawisz *<Esc>*, któremu odpowiada znak *#27*. Wtedy kończy się pętla *repeat* i instrukcja *Halt* powoduje przerwanie działania programu. Zwróćmy uwagę na różnicę pomiędzy procedurami standardowymi *Exit* i *Halt*. Pierwsza z nich powoduje zakończenie *procedury*, w której została wywołana, natomiast druga powoduje zakończenie całego *programu*.

Przykład 14.4. Procedura realizująca programowa kontrolę błędów we/wy

```

program Ex14_4;
type Tplik = file of Real;
var F:Tplik;

procedure Otwarcie_pliku(var F:Tplik; R:Boolean);
  var B:Word;
      Z:Char;
  {$I-}           {Wyłączenie systemowej kontroli we/wy}
begin
  repeat
    if R then Rewrite(F) else Reset(F);
    B:=IOResult;
    if B=0 then Exit;
    case B of
      2: Write('Nie ma takiego pliku. ');
      3: Write('Nie ma takiego adresu. ');
    102: Write('Plik nie skojarzony ');
    150: Write('Dyskietka zabezpieczona. ');
    152: Write('W napędzie brak dysku. ');
    else Write('Otwarcie pliku niemożliwe. ');
    end;
    Writeln;
    Writeln('<dowolny klawisz> - ponowienie proby');
    Writeln('<Esc> - wyjście z programu. ');
    Z:=Readkey;
  until Z=#27;
  Halt;
end;
  {$I+}           {Przywrócenie systemowej kontroli we/wy}
begin
  Assign(F, 'C:\Dupa');
  Otwarcie_pliku(F, False);
  { ----- }
end.

```

Ponieważ omówiona procedura *Otwarcie_pliku* wykonuje tylko jeden etap pracy z plikiem (jego otwarcie do zapisu lub odczytu), jej argumentem musi być zmienna plikowa *F*. Za pośrednictwem tego argumentu plikowego procedura otrzymuje informacje o pliku, z którym skojarzono zmienną plikową. Po wykonaniu swojego zadania, przez ten sam argument plikowy procedura przekazuje do programu głównego informację o stanie pliku.. Wynika z tego, że ten argument plikowy jest argumentem *wyjściowym* i dlatego został on w nagłówku definicji procedury poprzedzony słowem *var*.

14.8. Standardowe procedury i funkcje do pracy z plikami

Dotychczas w tym rozdziale omówiono procedury kojarzące, otwierające i zamykające plik: *Assign*, *Rewrite*, *Reset*, *Close*, procedury zapisu i odczytu danych *Write*, *Read*, a także funkcję *Eof*, która wykrywa końcowe położenie wskaźnika pliku. Poniżej znajduje się opis kilku innych podprogramów standardowych, pomocnych w realizacji działań na plikach elementowych. Podano nagłówki funkcji lub procedur, ponieważ to umożliwi zwięzły zapis rodzaju podprogramu i typów jego argumentów. Krótko opisano działanie poszczególnych podprogramów. Symbol *F* oznacza argument, będący zmienną plikową. Zmienna *F* musi być skojarzona z plikiem, na którym jest wykonywana operacja. Typu argumentu *F* nie podano, bo on może być różny, w zależności od typu elementów pliku.

```
function Filepos(var F):Longint;
```

Funkcja zwraca w formacie *Longint* numer pozycji pliku, na jakiej aktualnie znajduje się wskaźnik pliku. Funkcję wolno wywołać, gdy plik jest *otwarty*.

```
function Filesize(var F):Longint;.
```

Funkcja zwraca aktualny rozmiar (liczbę elementów) pliku. Nie jest to rozmiar pliku w bajtach. Funkcję można wywołać, gdy plik jest *otwarty*.

```
procedure Seek(var F; N:Longint);
```

Procedura przesuwa wskaźnik *otwartego* pliku na pozycję określoną przez argument *N*. Jeżeli w pliku nie ma pozycji *N*, to przy próbie zapisu lub odczytu z tej pozycji wystąpi błąd *we/wy*.

```
procedure Truncate(var F);
```

Procedura usuwa z pliku wszystkie elementy, położone między aktualną pozycją wskaźnika a końcem pliku.

```
procedure Rename(var F; S:string);
```

Procedura zmienia nazwę i ścieżkę dostępu do pliku zgodnie z wartością argumentu *S*. Można przenieść plik do innego katalogu, ale tylko na tym samym dysku. Procedurę można wywołać tylko wtedy, gdy plik jest *zamknięty*.

```
procedure Erase(var F);
```

Procedura usuwa z dysku plik, skojarzony z *F*. Usuwany plik musi istnieć i musi być zamknięty. Nie może mieć atrybutu „tylko do odczytu” (ang. *ReadOnly*).

14.9. Zmiana wartości wybranych elementów pliku

W przykładzie 14.5 pokazemy, w jaki sposób można w pliku fizycznym zmienić wartości, spełniające określone kryterium. Pokazana procedura *Zeruj_ujemne* ma za zadanie nadanie wartości zerowej tym wszystkim elementom pliku, które są ujemne..

W ogólnym przypadku, chcąc wpisać nową wartość na wybraną pozycję *Poz* pliku, należy odpowiednio ustawić wskaźnik pliku, posługując się procedurą *Seek(F,Poz)*. Następnie trzeba na wskazywaną pozycję wpisać nową wartość za pomocą instrukcji *Write(F, Bufor)*. Nowo wpisywana wartość musi być przedtem zapamiętana w zmiennej *Bufor*.

W przypadku procedury *Zeruj_ujemne* chodzi o nadanie nowej wartości *0* wszystkim tym elementom pliku, które są ujemne. Trzeba w tym celu zorganizować pętlę, która odczytuje kolejno wszystkie elementy pliku, sprawdzając ich znak. Jeżeli element jest ujemny, trzeba wykonać dwa działania: 1/ cofnąć wskaźnik pliku o jedną pozycję w stosunku do aktualnej, ponieważ w wyniku odczytu przesunął się on samoczynnie o jedną pozycję do przodu, 2/ na wskazywaną teraz pozycję, za pośrednictwem zmiennej buforowej, wpisać zero. Aktualną pozycję wskaźnika zwraca funkcja *Filepos(F)*. Chcąc cofnąć wskaźnik o jedną pozycję, zastosujemy następujące instrukcje:

```
Poz:=Filepos(F)-1;  
Seek(F,Poz);
```

Przykład 14.5. Zmiana wartości elementów pliku

```
program Ex14_5;
uses Crt;
const S ='A:\INTEGERS';
type Tplik=file of Integer;
procedure Zeruj_ujemne(S:string);
  var F:Tplik;
      Bufor:Integer;
      Poz:Longint;
begin
  Assign(F,S);
  Reset(F);

  while not Eof(F) do begin
    Read(F,Bufor);
    if Bufor<0 then begin
      Poz:=Filepos(F)-1;

      Seek(F,Poz);
      Bufor:=0;
      Write(F,Bufor);
    end;
  end;
  Close(F);
end;
```

14.10. Odczyt wartości z określonej pozycji pliku

W przykładzie 14.6 pokazano, w jaki sposób można odczytać wartość, pamiętaną na danej pozycji pliku. Argumentami wejściowymi odpowiedniej funkcji o nazwie *Daj_element* są: *S* – ścieżka dostępu do pliku oraz *P* – numer pozycji, której wartość chcemy odczytać. Argument *P* jest typu *Longint*, ponieważ liczba elementów pliku może być bardzo duża. Do przekazania wyniku służy argument wyjściowy *E*. Funkcja jako całość zwraca wartość typu *Byte*, normalnie równą 0. Działanie funkcji zaczyna się od sprawdzenia, czy numer pozycji nie przekracza rozmiaru pliku. Służy do tego instrukcja *if* z warunkiem $P > \text{Filesize}(F) - 1$. Przy nieprawidłowej wartości *P* funkcja zwróci wartość 1 i zakończy działanie. Jeżeli natomiast wartość *P* pozostaje w dopuszczalnym przedziale wartości, to instrukcja *Seek* ustawia wskaźnik pliku na pozycji *P*, po czym instrukcja *Write* kopiuje daną z tej pozycji bezpośrednio na wyjście *E*. Jak widać, argument wyjściowy *E* jest tutaj jednocześnie buforem danej dla instrukcji *Write*.

Przykład 14_6. Kontrolowany odczyt danej z danej pozycji pliku

```
type Tplik=file of Integer;
function Daj_element(S:string; P:Longint; var E:Integer):Byte;
  var F:Tplik;
begin
  Assign(F,S);
  Reset(F);
  Daj_element:=0;
  if P>Filesize(F)-1 then Daj_element:=1
  else begin
    Seek(F,P);
    Read(F,E);
  end;
  Close(F);
end;
```

Dla wykorzystania możliwości kontrolnych funkcji *Daj_element*, należy ją wywołać wewnątrz instrukcji *if-then-else*, jak następuje:

```
if Daj_element(S,P,E)=0 then Write('Element =',E)
else Write('Nie ma takiej pozycji.');
```

14.11. Usuwanie elementu z określonej pozycji pliku

Jedną z metod usunięcia elementu z pliku jest przekopiowanie do nowego pomocniczego pliku wszystkich elementów, z wyjątkiem usuwanego. Zaletą tej metody jest jej szybkość oraz to, że nie wymaga przesuwania elementów. Jeżeli plik, z którego usuwamy element, nazwiemy pierwotnym, a tworzony plik pomocniczy – wtórnym, to etapy procesu usuwania elementu można zapisać, jak następuje:

1. Z pliku pierwotnego kopiujemy do pliku wtórnego wszystkie elementy poza tym, który ma zostać usunięty.
2. Usuwamy plik pierwotny.
3. Zmieniamy nazwę i położenie pliku pomocniczego w taki sposób, by miał tę samą nazwę i położenie, jak usunięty plik pierwotny.

Implementację opisaną metody stanowi program pokazany w przykładzie 14.7.

Przykład 14.7. Usuwanie elementu z określonej pozycji pliku

```
program Ex14_7;
uses Crt;
const C='A:\SQRTS';
type Tplik=file of Real;
var Poz:Longint;

procedure Usun_pozycje(S:string; Poz:Longint);
const S1='A:\POMOC';
var F,F1:Tplik;
    Bufor:Real;
begin
    Assign(F,S);
    Assign(F1,S1);
    Reset(F);
    Rewrite(F1);
    while not Eof(F) do begin
        Read(F,Bufor);
        if Filepos(F)-1<>Poz then Write(F1,Bufor);
    end;
    Close(F);
    Erase(F);
    Close(F1);
    Rename(F1,S);
end;

begin
    Clrscr;
    Write('Podaj pozycje: ');
    Readln(Poz);
    Usun_pozycje(C,Poz);
    Readln;
end.
```

14.12. Plik współpracujący z programem z przykładu 13.10

W rozdziale 13.11 pokazano program, wykonujący rozmaite operacje na tablicy rekordów, która spełniała funkcję małej bazy ocen studentów. Wspomniano tam, że program w przedstawionej postaci jest praktycznie bezużyteczny, bo wszystkie dane, gromadzone i przetwarzane w tablicy rekordów *Grupa*, są tracone po zakończeniu programu. Na końcu rozdziału 13 zapowiedziano uzupełnienie programu o procedury, umożliwiające przechowanie tablicy w pamięci zewnętrznej. Spełniając tę zapowiedź, pokażemy tutaj dwie procedury własne przeznaczone do współpracy z plikiem. Procedury te, włączone do programu *Ex13_10*, czynią go w pełni użytecznym. Przedstawiono je w przykładzie 14.8.

Dla obu procedur, typ zmiennej plikowej zdefiniowano jako *file of Tabstud*. Oznacza to, że elementem pliku jest tablica takiego samego typu, jak tablica *Grupa*. Plik zawiera tylko jeden element, ponieważ na początku programu kopiujemy do programu całą zawartość pamiętanej na dysku tablicy, a po zakończeniu programu przenosimy w całości do pliku nową zawartość tablicy, zmodyfikowaną w wyniku działania programu.

Procedura *Z_pliku* jest wywoływana na początku programu głównego. Jej przeznaczeniem jest przekazanie do programu zawartości tablicy, przechowywanej w postaci jednoelementowego pliku dyskowego, do którego prowadzi ścieżka dostępu *S*. Normalnie plik ten istnieje, więc do jego otwarcia stosuje się procedurę *Reset*. Następnie instrukcja *Read* kopiuje dane z pliku do tablicy *Grupa*. Może wystąpić przypadek szczególny, gdy plik jeszcze nie istnieje. Ma to miejsce przy pierwszym uruchomieniu programu. Chcąc kontrolować tę sytuację, za pomocą dyrektywy *{SI-}* wyłączono systemową kontrolę błędów *we/wy*. Po próbie otwarcia nieistniejącego pliku, funkcja kontrolna *IOResult* zwróci niezerową wartość, czego następstwem jest natychmiastowe zakończenie działania procedury przez wywołaną warunkowo instrukcję *Exit*.

Procedura *Na_plik* jest wywoływana na końcu programu głównego. Jej zadaniem jest przekazanie z powrotem do pliku zawartości tablicy *Grupa*, zmodyfikowanej w czasie działania programu. Ponieważ plik jest zapełniany na nowo, do jego otwarcia zastosowano instrukcję *Rewrite*. Zapis tablicy do pliku jest realizowany przez pojedynczą instrukcję *Write*. Następnie plik zostaje zamknięty.

Przykład 14.8. Procedury do odczytu tablicy z pliku i zapisu tablicy w pliku

```
{SI-}
procedure Z_pliku(var Grupa:Tabstud; S:string);
    var F:file of Tabstud;
begin
    Assign(F,S);
    Reset(F);
    if IOResult<>0 then Exit;
    Read(F,Grupa);
    Close(F);
end;
{SI+}

procedure Na_plik(var Grupa:Tabstud; S:string);
    var F:file of Tabstud;
begin
    Assign(F,S);
    Rewrite(F);
    Write(F,Grupa);
    Close(F);
end;
```

Poniżej pokazujemy zmodyfikowaną postać programu głównego z przykładu 13.10. Jak widać, na początku programu jest wywoływana procedura *Z_pliku*, a na jego końcu – procedura *Na_plik*. Uruchamiając program, mamy w tablicy *Grupa* dane w postaci uzyskanej w czasie ostatniej sesji pracy z programem. Kończąc pracę z programem, prześlemy z powrotem na dysk nową postać danych, uzyskaną w wyniku ich uzupełnienia lub modyfikacji.

Przykład 14.9. Zmodyfikowany program główny z rozdziału 13.10

```
{Program główny =====}
begin
  Z_pliku(Grupa, 'C:\MOJE\GRUPA'); {Odczyt tablicy z pliku}
  Clrscr;
  Belka_opcji;
  Window(10, 5, 70, 25);
  repeat
    case C of
      '1' : begin
              Dodaj(Grupa);
              Sort_alfabet(Grupa);
            end;
      '2' : Usun(Grupa);
      '3' : Wypisz_ocena(Grupa, 0);
      '4' : begin
              Sort_ocena(Grupa);
              Wypisz_ocena(Grupa, 0);
              Sort_alfabet(Grupa);
            end;
      '5' : begin
              Sort_ocena(Grupa);
              Wypisz_ocena(Grupa, Dane2);
              Sort_alfabet(Grupa);
            end;
      '6' : Wypisz_ocena(Grupa, Max_ocen(Grupa));
      '7' : Szukaj(Grupa);
    end;
    Komunikat(1);
    Decyzja(C);
    Clrscr;
  until C=#27;
  Window(1, 1, 80, 25);
  Clrscr;
  Na_plik(Grupa, 'C:\MOJE\GRUPA'); {Zapis tablicy do pliku}
end.
```