1. Typy rekordowe

W ogólności typem rekordowym nazywa się złożoną strukturę danych o różnych elementach składowych

1.1. Rekordy klasyczne

Rekord klasyczny to złożona struktura danych, której składowe, zwane polami, mogą należeć do różnych typów. Poszczególne pola same mogą być strukturami złożonymi, przy czym liczba pól rekordu jest ustalona. Definicja takiego rekordu jest następująca:

type *identyfikator-typu* = **record**

lista-deklaracji-pól

end;

Ostatnia deklaracja może być wariantowa. Składa się ona ze słowa kluczowego *case*, po którym występuje identyfikator oraz jego typ oraz słowa kluczowego *of*., po którym następuje wykaz wariantów. Deklaracje wariantowe stosuje się w przypadku gdy chcemy utworzyć jeden typ rekordowy zawierający różne typy danych (liczba pól zmienna) lub gdy zamierzamy w różny sposób interpretować dane zapisywane w części wariantowej rekordu.

Przykładowy typ rekordowy opisujący dane pracownika firmy:

```
type pracownik = record
        imie : string[20];
        nazwisko : string[40];
        wiek : byte;
        plec : boolean;
        stanowisko : string[50];
        pensja : real;
        //pole wariantowe, identyfikator dziecko
        case dziecko: boolean of
        true: (dzieci: array [1..5] of string[20]); //wykaz wariantów
        false: (znak: Char);
end;
```

Mając zdefiniowany typ opisujący pracownika możemy utworzyć zmienną tego typu i odwoływać się do poszczególnych pól rekordowych (odwołanie ma postać nazwa_zmiennej.nazwa_pola), np.:

```
var
    p1 : pracownik;
begin
    p1.imie := ‘Jan’;
    p1.nazwisko := ‘Kowalski’;
    p1.wiek := 50;
    p1.plec := true;
    p1.stanowisko := ‘Dyrektor ds. sprzedaży’;
    p1.pensja := 12344.34;
    p1.dziecko := true;
    for i := 1 to 5 do
        (* Funkcja Format umożliwia sformatowanie łańcucha na podstawie
        przekazanych parametrów. pierwszy parametr tej funkcji jest wzorcem.
        W miejsce oznaczone %d podstawione zostaną wartości przekazane w
        drugim parametrze, czyli wartość zmiennej p1.dzieci[2] będzie miała
        następującą postać: dziecko 2. *)
        p1.dzieci[i] := Format(‘dziecko %d’, [i]);
end;
```

To samo co powyżej można zapisać prościej:

```
var
    p1 : pracownik;
begin
    with p1 do
    begin
        imie := ‘Jan’;
        nazwisko := ‘Kowalski’;
        wiek := 50;
        plec := true;
        stanowisko := ‘Dyrektor ds. sprzedaży’;
        pensja := 12344.34;
        dziecko := true;
        for i := 1 to 5 do
            dzieci[i] := Format(‘dziecko %d’, [i]);
    end;
end;
```

df

1.2. Rekordy klasopodobne

Rekordy o strukturze przypominającej klasy. Oprócz elementów, które posiadają rekordy klasyczne mogą dodatkowo mieć: metody, konstruktory, własności, pola klasowe i typy zagnieżdżone. Od klas odróżnia je m.in. brak dziedziczenia, brak destruktorów oraz możliwość przeciążania operatorów. Definicja takiego rekordu jest następująca:

type *identyfikator-typu* = **record**
lista-elementów-rekordu-klasopodobnego
end;

1.3. Rekordy pomocnicze

Są to rekordy skojarzone z innymi rekordami, które definiują dla nich dodatkowe metody i własności. Typ rekordu pomocniczego rozszerza funkcjonalność danego rekordu i jest wykorzystywany głównie przy modyfikacji kodu źródłowego, po to, żeby nie zmieniać definicji rekordu oryginalnego. Definicja takiego rekordu jest następująca:

type *identyfikator-typu* = **record helper for** *identyfikator-typu-oryginalnego*
lista-elementów-rekordu-pomocniczego
end;

2. Typy plikowe

Plik jest pewną strukturą danych zapisaną na dysku i identyfikowaną za pomocą nazwy. Istnieje kilka rodzajów plików np. tekstowe, typowane, binarne etc.

Ogólny schemat operacji plikowej w Delphi obejmuje cztery etapy:

- skojarzenie zmiennej plikowej z odpowiednim plikiem (będącym na dysku lub nowo tworzonym),
- otwarcie pliku (przygotowujące do zapisywania lub odczytywania informacji),
- wykonanie operacji zapisu lub odczytu danych,

- zamknięcie pliku (przerwanie skojarzenia pomiędzy zmienną plikową i plikiem).

2.1. Pliki tekstowe

Pliki tekstowe przechowują dane w postaci wierszy tekstu zakończonych znakami końca wiersza. Są to pliki o dostępie sekwencyjnym, co oznacza, że aby dostać się do wybranego elementu pliku, należy przeczytać wszystkie elementy znajdujące się przed nim.

- **Deklaracja zmiennej plikowej oraz skojarzenie zmiennej plikowej z plikiem**

  Program odwołuje się do plików poprzez *zmienne plikowe*, czyli złożone struktury danych reprezentujące fizyczne pliki zapisane na dysku. Przypisanie konkretnego pliku do zmiennej realizuje polecenie *AssignFile*. Po wykonaniu tej procedury wszystkie odwołania do zmiennej plikowej będą dotyczyły skojarzonego z nią pliku (o nazwie którego możemy zapomnieć).

```
var
        fp : TextFile;
begin
        AssignFile(fp, 'c:\plik.txt'); //skojarzenie pliku 'c:\plik.txt' ze zmienną fp
        {dalsze operacje na pliku}
end;
```

- **Otwarcie pliku**

  - **`procedure Rewrite(var F: File [; Recsize: Word ] );`** - umożliwia otwarcie pliku niezależnie od tego czy istniał on poprzednio. Jeśli nie istniał to tworzy ona nowy plik o danej nazwie, jeśli istniał to zeruje długość istniejącego pliku i ustawia wskaźnik plikowy na jego początek, z czym wiąże się utrata wszystkich danych w pliku. Otwiera plik wyłącznie do zapisu.

```
var
        fp : TextFile;
begin
        AssignFile(fp, 'c:\plik.txt');
        try
                Rewrite(fp);
                {dalsze operacje na pliku}
         finally
                CloseFile(fp);
```

```
end;
```

- **`procedure Reset(var F [: File; RecSize: Word ] );-`**

  umożliwia otwarcie istniejącego pliku i ustawienia *wskaźnika plikowego* na jego początku. W przypadku braku pliku procedura zakończy się błędem. Otwiera plik wyłącznie do odczytu.

```
var
        fp : TextFile;
begin
        AssignFile(fp, 'c:\plik.txt');
        try
                Reset(fp);
                {dalsze operacje na pliku}
         finally
                CloseFile(fp);
end;
```

  W celu sprawdzenia czy plik istnieje można użyć funkcji *FileExists(SciezkaDoPliku)* np.

```
If FileExists('c:\plik.txt')
        Reset(fp);
else Rewrite(fp);
```

- **`procedure Append ( var FileHandle : TextFile );-`**

  procedura ta otwiera plik do dopisywania. Otwiera go do zapisu nie niszcząc przy tym poprzedniej zawartości i ustawia wskaźnik plikowy na jego końcu.

```
var
        fp : TextFile;
begin
        AssignFile(fp, 'c:\plik.txt');
        try
                Append(fp);
                {dalsze operacje na pliku}
         finally
                CloseFile(fp);
end;
```

- **Odczyt danych z pliku**

  Do odczytu danych z pliku służą procedury *Read(zmienna-plikowa, lista-elementów)* oraz *Readln (zmienna-plikowa, lista-elementów)* (odczytuje dane wraz ze znakiem końca linii).

```
var
```

```
        fp : TextFile;
        Str: String;
begin
        AssignFile(fp, 'c:\plik.txt');
        try
                Reset(fp);

                {dopóki funkcja Eof nie natrafi na koniec pliku odczytywane są
                 kolejne wiersze pliku tekstowego}
                while not Eof(fp) do
                begin
                        {odczytanie pojedynczego wiersza z pliku i przypisanie
                         go do zmiennej Str}
                        Readln(fp, Str);

                        {kolejne wiersze z pliku dodawane są do komponentu
                        Memo i zostaną w nim wyświetlone}
                        Memo1.Lines.Add(Str);
                end;
         finally
                CloseFile(fp);
end;
```

- **Zapis danych do pliku**

  Do zapisu danych do pliku służą procedury *Write(zmienna-plikowa, lista-elementów)* oraz *Writeln (zmienna-plikowa, lista-elementów)* (zapisuje dane wraz ze znakiem końca linii).

```
var
        fp : TextFile;
        I: Integer;
begin
        AssignFile(fp, 'c:\plik.txt');
        try
                Rewrite(fp);

                {zapisz wszystkie wiersze z komponentu Memo do pliku }
                for I := 0  to Memo1.Lines.Count – 1 do
                begin
                        Writeln(fp, Memo1.Lines[I]);
                end;
         finally
                CloseFile(fp);
end;
```

2.2. Pliki typowane

Pliki typowane to pliki o regularnym układzie np. całe rekordy danych.

- **Deklaracja zmiennej plikowej dla pliku rekordów oraz skojarzenie zmiennej plikowej z plikiem**.

  Pojedynczym elementem pliku typowanego jest tutaj nie 1 bajt, ale cały rekord. Do takiego pliku można jedynie zapisywać i odczytywać całe rekordy (zapisywanie oraz odczytywanie pojedynczych pól jest niewykonalne). Skojarzenie *zmiennej plikowej* z fizycznym plikiem znajdującym się na dysku zachodzi tak jak dla pliku tekstowego, za pomocą polecenia *AssignFile*.

```
type
        books = record
                key: Integer;
                nazwa: String[64];
        end;
var
        fp : file of books;
begin
        {skojarzenie pliku rekordów 'c:\plik.dat' ze zmienną fp}
        AssignFile(fp, 'c:\plik.dat');
        {dalsze operacje na pliku}
end;
```

- **Otwarcie pliku**

  Podobnie jak w przypadku plików tekstowych (punkt. 1.1), do otwarcia pliku służy procedura *Reset* lub *Rewrite*. W odróżnieniu od plików tekstowych, pliki typowane można odczytywać oraz zapisywać bez ograniczeń, bez względu na to czy zostały otwarte za pomocą procedury *Reset czy Rewrite* (w tym ostatnim przypadku trzeba najpierw zapisać do pliku jakieś dane).

```
var
        fp : file of books;
begin
        AssignFile(fp, 'c:\plik.dat');
        try
                Reset(fp);
                {dalsze operacje na pliku}
         finally
                CloseFile(fp);
end;
```

  W celu otwarcia pliku do dopisywania (bez niszczenia jego dotychczasowej zawartości), należy wykonać następujące czynności:

```
var
        fp : file of books;
begin
        AssignFile(fp, ‘c:\plik.dat’);
        try
                Reset(fp);
                {ustawi wskaźnik plikowy za ostatnim elementem pliku (czyli
                 przygotuje plik do dopisywania)}
                Seek(fp, FileSize(fp));
                {dalsze operacje na pliku}
         finally
                CloseFile(fp);
end;
```

- **Odczyt danych z pliku**

  Pliki typowane zawierają tylko dane określonego typu, nie mogą zawierać znaków końca wiersza, użycie więc procedury *Readln* jest w ich przypadku niedozwolone. Do odczytu danych z pliku rekordów służy **tylko procedura *Read*.**

```
var
        fp : file of books;
        I : integer;
        elem : books;
begin

        if not FileExists('c:\plik.dat') then Exit;

        AssignFile(fp, 'c:\plik.dat');
        try
                Reset(fp);
                ListBox1.Clear;

                {odczytaj wszystkie rekordy z pliku}
                for I := 0 to FileSize(fp) - 1 do
                begin
                        Read(fp, elem);

                        {zawartość pola nazwa każdego rekordu dodaj jako
                         osobny element  do komponentu ListBox}
                        ListBox1.Items.Add(elem.nazwa);
                end;
         finally
                CloseFile(fp);
        end;
end;
```

- **Zapis danych do pliku**

  Pliki typowane zawierają tylko dane określonego typu, nie mogą zawierać znaków końca wiersza, użycie więc procedury *Writeln* jest w ich przypadku

niedozwolone. Do zapisu danych do pliku rekordów służy **tylko procedura *Write*.**

```
var
        fp : file of books;
        I : integer;
        elem : books;
begin
        {wypełnienie rekordu danymi}
        elem.nazwa := 'Justynian';
        elem.key := 1;

        AssignFile(fp, 'c:\plik.dat');
        try
                Rewrite(fp);
                Write(fp, elem);
         finally
                CloseFile(fp);
        end;
end;
```

2.3. Inne operacje na plikach

2.3.1. Pliki tekstowe

- `function Eof(var F : file):Boolean;` - zwraca true gdy plik znajduje się w pozycji za ostatnim elementem lub nie zawiera żadnych elementów lub wartość false w przeciwnym przypadku.
- Funkcja *SeekEof* daje takie same wyniki co Eof, z tym, że ignoruje najbliższe spacje, znaki tabulacji oraz znaki CR i LF.
- Funkcja *Eoln* zwraca true gdy plik znajduje się w pozycji znaku końca wiersza lub gdy wartością funkcji *Eof* dla tej samej zmiennej plikowej jest true.
- Funkcja *SeekEoln* daje takie same wyniki co Eoln, z tym, że ignoruje najbliższe spacje oraz znaki tabulacji.

2.3.2. Pliki nie będące plikami tekstowymi

- `function FilePos(var F): Longint;` - zwraca aktualną wartość wskaźnika plikowego (aktualną pozycję w pliku – jeżeli wartość eof jest równa true to zwracany jest rozmiar pliku).

- `function FileSize(var F): Integer;` - zwraca rozmiar pliku dyskowego.
- `procedure Seek(var F; N: Longint);` - ustawia wskaźnik plikowy na określonym miejscu w pliku, definiowanym przez parametr *N*.
- `procedure Truncate(var F);` - służy do przycięcia pliku, od miejsca bieżącego (ustawione np. przez procedurę *Seek*) do końca pliku.

2.3.3. Wszystkie pliki

- *Chdir* – powoduje zmianę bieżącego katalogu na katalog i/lub napęd określony przez argument.
- *GetDir* - pozwala uzyskać informację o bieżącym katalogu w określonym napędzie.
- *MkDir* – pozwala na utoworzenie nowego podkatalogu.
- *RmDir* – usuwa pusty, nie zawierający żadnych plików katalog.
- *Erase* - usuwa plik dyskowy.

Przykład: Aplikacja wykorzystująca pliki