1 Łańcuchy w Delphi i operacje na nich

1.1 Typ łańcuchowy

Typ łańcuchowy w Delphi opisuje ciąg znaków, zwany **łańcuchem** (ang. **String**). Definiuje się go następująco:

```
var zmienna : string;
```

Możemy też wyspecyfikować długość łańcucha, i wtedy definiujemy go następująco:

```
var zmienna : string[20];
```

1.2 Ważniejsze operacje na zmiennych typu łańcuchowego

- concat(string1, string2) - konkatenacja, łączy ze sobą dwa dane łańcuchy

Przykład użycia:

```
var s1,s2 : string;
begin
  s1:='Jan';
  s2:='Kowalski';
  s1:=concat(s1,s2); //'JanKowalski'
  s1:=concat('Tomasz ',s1); //'Tomasz JanKowalski'
  s1:=concat(s1,' ma 40 lat'); //Tomasz JanKowalski ma 40 lat'
end.
```

- copy(string,indeks,liczba) - kopiuje do danego łańcucha daną liczbę znaków,zaczynając od znaku będącego w danym łańcuchu na pozycji wskazanej przez indeks

Przykład użycia:

```
var s1,s2:string;
begin
  s1:='Jan Kowalski';
  s2:=copy(s1,3,5); //s2 jest równe 'n Kow'
end.
```

- delete(string, indeks, liczba) - usuwa z danego łańcucha podciąg długości danej przez liczbę, zaczynając od znaku wskazanego przez indeks

Przykład użycia:

```
var s1 : string;
begin
  s1:='Jan Kowalski';
  delete(s1,4,6); //zostaje 'Janski'
end.
```

- **insert(string1,string2,indeks)** - wstawia do łańcucha string2 łańcuch string1, umieszczając jego początek na pozycji wskazanej przez indeks

Przykład użycia:

```
var s1,s2 : string;
begin
  s1:='Jan Kowalski';
  s2:='ttt';
  insert(s2,s1,6); // 'Jan Kotttwalski'
end.
```

- **length(string1)** - podaje długość danego łańcucha

Przykład użycia:

```
var s1 : string;
    i  : integer;
begin
  s1:='Jan Kowalski';
  i:=length(s1); //i jest równe 12
end.
```

- **pos(substring,string)** - podaje pozycję (numer indeksu znaku) pierwszego wystąpienia danego podciągu w danym ciągu. Jeżeli dany podciąg nie wystepuje ani razu w danym ciągu to funkcja pos zwraca wynik 0.

Przykład użycia:

```
var s1 : string;
     i : integer;
begin
  s1:='Jan Kowalski';
  i:=pos('a',s1); //i jest równe 2
  i:=pos('A',s1); //i jest teraz równe 0
  i:=pos('n Kow',s1); // i jest teraz równe 3
end.
```

- **str(liczba)** - zamienia liczbę na łańcuch

Przykład użycia:

```
var i : integer;
    s1 : string;
begin
  i:=23456;
  str(i,s1); // s1='23456'. Jest to łańcuch znaków
end.
```

- val(string, liczba, kod) - zamiana łańcucha znaków na liczbę, kod kontroluje poprawność zamiany. Kod określa numer pierwszego znaku nie dającego się zamienić na liczbę. Jeśli kod=0 to wszystko jest w porządku.

Przykład użycia:

```
var s1 : string;
    i, kod : integer;
begin
  s1:='23456';
  val(s1,i,kod); //kod=0, zamiana przebiegła pomyślnie
  s1:='234m6';
  val(s1,i,kod); //kod=4, błąd
end.
```

2 Typ rekordowy w Delphi

Rekordem w Delphi nazywamy złożoną strukturę danych składającą się z **pól**, z których każde jest zmienną określonego przez nas typu. Pola rekordu mogą także same być złożonymi strukturami. Zmienną typu rekordowego definiujemy w deklaracji typów następująco (poniższy przykład pokazuje typ rekordowy opisujące dane o pracowniku firmy):

```
type pracownik = record
  imie : string[20];
  nazwisko : string[40];
  wiek : byte;
  plec : boolean; //np.: 0 - kobieta, 1 - mezczyzna
  stanowisko : string[50];
  pensja : real;
end;
```

Mając zdefiniowany typ rekordowy **pracownik** możemy teraz tworzyć zmienne tegoż typu i dokonywac na nich rozmaitych operacji, np.:

```
var p1 : pracownik;
begin
  p1.imie:='Jan';
  p1.nazwisko:='Kowalski';
  p1.wiek:=45;
  p1.plec:=1;
  p1.stanowisko:='Dyrektor d/s sprzedazy';
  p1.pensja:=2345.67;
end;
  // Możemy też prościej:
  with p1 do begin
    imie:='Jan';

    ...
```

```
  end;
end.
```

Jak pokazuje powyższy przykład, do poszczególnych pól zmiennej typu rekordowego odwołujemy się pisząc **nazwa_zmienej.nazwa_pola**.

3 Wskaźniki i struktury dynamiczne w Delphi

Zmienne dynamiczne, w odróżnieniu od dotychczas poznanych zmiennych statycznych, charakteryzują się tym, że nie muszą one istnieć w pamięci przez cały czas działania programu. Możemy w trakcie działania programu rezerwowac pamięć dla nowych zmiennych, jak i zwalniać pamięć przeznaczoną dla już niepotrzebnych nam zmiennych. W tym celu będzie nam potrzebne odwoływanie się do konkretnych adresów pamięci, w których są zapisane interesujące nas zmienne. Robi się to za pomocą **wskaźników**.

3.1 Typ wskaźnikowy

Wskaźnik w języku Delphi definiujemy następująco:

```
type typ_wskaznikowy = ^typ_zmiennej;

// przykłady:

type wskint = ^integer;
     wskprac = ^pracownik;
     pracownik = record

     ...
end;
```

3.2 Tablice dynamiczne

Wcześniej poznane **tablice statyczne** charakteryzują się tym, iż trzeba z góry (deklarujac zmienną typu tablicowego) podać liczbę elementów, z których składa się dana tablica, np.:

```
var tab : array[1..20] of integer;
```

Tablice dynamiczne zaś pozwalają nam w dowolnej chwili zmieniać ich długość na taką, jaka jest nam w danej chwili potrzebna, nie tracąc pamięci na już zbędne nam elementy. Robi się to za pomocą funkcji **setlength(tablica, ile);**.

Pokażemy to na przykładzie poniżej:

```
var tab1 : array of integer; // bez określonej długości - tablica
                             // dynamiczna
    tab2 : array of array of integer; // dwuwymiarowa tablica
                                      // dynamiczna

setlength(tab1,5); // ustawiamy długość tablicy tab1 na 5
```

```
setlength(tab2,10,20); // macierz 10*20

// Możemy także każdej z kolumn nadać osobną licbę wierszy, np.:

setlength(tab2,10); // 10 kolumn
setlength(tab2[1],3);
setlength(tab2[2],5);

// i tak dalej.
```

3.2.1 Różnice pomiędzy tablicami dynamicznymi a statycznymi

Uwaga. Niektóre instrukcje działają inaczej dla tablic statycznych, a inaczej dla dynamiczych. Np. operacja przypisania dla tablic dynamicznych powoduje, ustawienie wskaźnika na miejsce w pamięci, gdzie zapisana jest przypisywana tablica, i wszystkie operacje dokonane później na przypisywanej tablicy działają też na tablicę przypisującą, tak jak pokazano na poniższym przykładzie:

```
var A,B : array of string;
begin
  setlength(A,1);
  setlength(B,1);
  A[0]='Adam';
  B[0]='Bartosz';
  A:=B; //ustawia wskaźnik do tablicy A tam, gdzie jest zapisana w
        //pamięci tablica B
  B[0]:='Tomasz';
  // teraz A[0] jest równe 'Tomasz'. Jeśli A i B byłyby tablicami
  // statycznymi to A[0] byłoby nadal równe 'Adam'.
end.
```

Jeśli chcemy skopiować tablicę (lub jej fragment) dynamiczną, to musimy użyć instrukcji **copy**, tak jak na przykładzie poniżej:

```
var A,B : array of string;
begin
  setlength(A,1);
  setlength(B,1);
  A[0]:='Adam';
  B[0]:='Bartosz';
  A:=copy[B]; //kopiowanie tablicy do innej tablicy, bez przesuwania
              //wskaźnika do niej
  B[0]:='Tomasz';
  // teraz A[0] jest równe 'Bartosz'
end.
```

4 Podstawy zmiennych i struktur dynamicznych w Delphi

4.1 Deklaracja zmiennych dynamicznych

Mając daną zmienną będącą wskaźnikiem na interesujący nas typ danych (przypomnijmy):

```
  type wskt = ^typ_zmiennej;

...

var nowy : wskt;
```

Pamięć na dany wskaźnik do zmiennej rezerwujemy za pomocą instrukcji **new(wskaznik)**, a zwalniamy pamięć instrukcją **dispose(wskaznik)**.

Czyli do powyższego przykładu dopisalibyśmy:

```
begin
  new(nowy);

  ... // ciąg interesujących nas instrukcji

  dispose(nowy);
end.
```

4.2 Stosy

Stos (ang. stack) jako dynamiczna struktura danych charakteryzuje się tym, że nowy element można dodać tylko na wierzch stosu, a także zdjąć (i odczytać) mozna tylko element leżący na wierzchu stosu. Nie możemy odczytać ani dokonać zadnej operacji na elemencie leżącym w głębi stosu, dopóki nie zdejmiemy wszystkich elementów leżących powyżej niego.

4.2.1 Implementacja stosu w Delphi

Każdy element stosu w Delphi przedstawiony jest jako struktura składająca się z pola wartości, będącego polem interesującego nas typu zmiennej, opisującej wartość danego elementu stosu, oraz wskaźnika, który wskazuje na element leżący pod nim. Pusty wskaźnik (nie wskazujący na żaden element) oznaczamy w języku Delphi symbolem **nil**. Dla najgłębiej położonego elementu stosu wskaźnik będzie miał wartość nil, ponieważ pod tym elementem już nic nie leży.

```
type wsk = ^elstosu;
     elstosu = record
       dane = integer; // lub dowolny inny interesujący nas typ
                       //danych. Dla przykładu niech wartości
                       //elementów stosu będą liczbami całkowitymi.
       nastepnik = wsk;//wskazuje na adres pamięci elementu leżącego
                       //POD danym elementem
     end;
```

```
var wierzch = wsk; //musimy zaznaczyć, gdzie jest przechowywany w
                   //pamięci wierzchołek stosu.
```

Dodawanie danego elementu do stosu

```
procedure dodaj_na_stos(wartosc : integer)
var nowy : wsk;
begin
  new(nowy); // rezerwujemy pamięć na nowy element
  nowy^.dane:=wartosc;

  //Uwaga!
  // nowy - adres pamięci, gdzie przechowywana jest zmienna 'nowy'
  // nowy^ - to, co jest zapisane pod adresem wskazanym przez
  // wskaznik 'nowy' - czyli sama zmienna, tutaj typu rekordowego

  nowy^.nastepnik=wierzch; // nowo dodany element ma wskazywać na to,
                           //co poprzednio było wierzchem stosu
  wierzch=nowy; // teraz na wierzchu stosu leży nasz nowy element

end;
```

Zdejmowanie elementu ze stosu

```
procedure zdejmij_ze_stosu;
// Uwaga. W odróżnieniu od dodawania na stos procedura zdejmowania
// elementu ze stosu nie potrzebuje parametru. Zastanówcie się,
// dlaczego?

var punkt : wsk; // pomocnicza zmienna
begin
  if (wierzch=nil) then writeln('Stos jest pusty, nie ma czego
                                zdejmować')
  else begin
    punkt:=wierzch^.wsk;
    writeln('Odczytujemy wartość zdejmowanego elementu:
            ',wierzch^.wartosc);
    dispose(wierzch);
    wierzch:=punkt;
  end;


end;
```

4.3 Kolejki

Kolejka (ang.queue) jest dynamiczną strukturą danych charakteryzującą się tym, że w dowolnej chwili odczytać (i zdjąć z kolejki) możemy tylko element będący na przodzie kolejki, natomiast dodać nowy element możemy tylko z tyłu kolejki.

4.3.1 Implementacja kolejek w Delphi

Definiując kolejkę w Delphi musimy zaznaczyć, gdzie w pamięci przechowywany jest zarówno początek, jak i koniec kolejki, za pomocą odpowiednich wskaźników.

```
type wsk = ^elkolejki;
     elkolejki = record;
       dane : integer; // dla przykladu, elementy kolejko mogą mieć
                       // również wartości innych typów
       wskaznik : wsk;
     end;

var pocz,kon : wsk;
```

Dodawanie elementu do kolejki

```
procedure dodaj_do_kolejki(wartosc : integer);
var nowy : wsk;
begin
  if (pocz=nil) // lub if (kon=nil), tutaj wszystko jedno,
                // należy zauważyć, że pocz=nil wtedy i tylko
                // wtedy gdy kon=nil, tj. gdy kolejka aktualnie
                // jest pusta
  then begin
    new(nowy);
    nowy^.dane=wartosc;
    nowy^.wskaznik=nil;
    pocz:=nowy;
    kon:=nowy;
  end
  else begin // gdy kolejka nie była pusta, dodać możemy tylko na
             // koniec kolejki
    new(nowy);
    nowy^.dane=wartosc;
    nowy^.wskaznik=nil;
    kon^.wskaznik=nowy;
    kon=nowy;
  end;
end;
```

Dodając do kolejki nowy element musimy elementowi będącemu poprzednio końcem kolejki ustawić wskaźnik na nasz nowy element, a sam nasz nowy element będzie teraz nowym końcem i będzie miał wskźnik ustawiony na **nil** - bo jest ostatni w kolejce i jego następnik jest pusty.

Zdejmowanie elementu z kolejki

```
procedure zdejmij_z_kolejki;
var punkt : wsk; // pomocnicza zmienna;
begin
  if (pocz=nil) then writeln('Kolejka pusta, nie ma czego zdejmowac')
  else begin
    punkt:=pocz^.wskaznik;
    writeln('Wartosc usuwanego elementu: ',pocz^.dane);
    dispose(pocz);
    pocz:=punkt;
  end;
end;
```

Z kolei usunąć element z kolejki możemy tylko z początku, więc zapamiętujemy (w pomocniczej zmiennej nazwanej **punkt**) miejsce, na które wskazywał początkowy element kolejki (zanim go zdejmiemy). Następnie usuwamy dany element z pamięci (instrukcją **dispose**), a nowym początkiem kolejki staje się teraz to, co wskazuje wskaźnik pomocniczy **punkt**.