Zmienne typu łańcuchowego

- struktura danych do przechowywania napisów (ciągu − łańcucha znaków),
- liczba znaków (*długość* łańcucha) może być różna,
- maksymalną przewidywaną długość łańcucha podaje się w definicji typu,
- najmniejsza długość wynosi 0 znaków (taki łańcuch jest *pusty*),
- największa długość to 255 znaków.

Definicje/deklaracje typu łańcuchowego

```
type
   lancuch = string[ 10 ];
   dlugi_lancuch_1 = string[ 255 ];
   dlugi_lancuch_2 = string;

var
   tekst :  lancuch;
   nazwa : string[ 30 ];
```

Format zmiennej łańcuchowej w pamięci

tekst := ’Pascal’ ;

tekst	**#6**	’**P**’	’**a**’	’**s**’	’**c**’	’**a**’	’**l**’	...	...	...	...
	0	1	2	3	4	5	6	7	8	9	10

Element zerowy − znak, którego numer jest równy *dynamicznej*
czyli aktualnej długości zmiennej łańcuchowej.

Stałe łańcuchowe:

’Turbo Pascal’

’’

’B:\Jezyki\TP\TURBO.EXE’

’Festiwal’’97’

#3’PepsiCola’ → ♥PepsiCola

’To sa trzy’#7#7#7’dzwieki’ → To sa trzy ♪♪♪ dzwieki

Definicje stałych łańcuchowych:

```
Const
   Autor = 'Jan Kowalski' ;
   Wersja = 'ver. 2.1' ;
```

Zmienne łańcuchowe inicjowane:

```
Const
   nazwa_domyslna : string = 'program.pas' ;
```

Instrukcja przypisania:

- przypisanie łańcucha dłuższego niż deklarowana maksymalna długość zmiennej spowoduje *obcięcie* końcowych znaków łańcucha.

```
var   M :  string [6];
M := 'TurboPascal';        →        M = 'TurboP'
```

Uwaga: Łańcuchowi wolno przypisać znak. (odwrotnie nie można!),

Odwoływanie się do elementów łańcucha:

- tak samo, jak do elementów jednowymiarowej tablicy znakowej.
- dyrektywa kompilatora {$R+} do kontroli zakresu

```
function NaDuze ( tekst : string );
var
   i : Byte;
begin
   for  i := 1 to Length( tekst ) do
      tekst[ i ] := UpCase( tekst[ i ] );
   NaDuze := tekst;
end;


function IleCyfr ( tekst : string ): byte ;
var
   i , licznik : Byte;
begin
   licznik := 0;
   for  i := 1 to Length( tekst ) do
      if  (tekst[ i ] >= '0') and (tekst[ i ] <= '9') then
         licznik := licznik + 1;
   IleCyfr := licznik;
end;
```

Operacja sklejania łańcuchów:

operator sklejania „+”

```
var
   S1, S2, S : string;
begin
   S1 := 'Error 1:';
   S2 := ' Out of memory';
   S : = S1 + S2 ;                    →      'Error 1: Out of memory'
end;
```

Porównywanie łańcuchów:

operatory: =, < >, <, >, <=, >=.

- Bada się numery porządkowe kodu ASCII kolejnych znaków,
- Porównywanie kończy się po dojściu do końca krótszego łańcucha,
- Łańcuch *większy* to ten, który na wcześniejszej pozycji ma znak o większym numerze porządkowym,
- Łańcuch jest większy od swojego podłańcucha, a łańcuch pisany małymi literami jest większy od swojego odpowiednika pisanego dużymi literami,
- Łańcuchy są *równe*, jeżeli mają tę samą liczbę znaków i jeżeli znaki na każdej ich pozycji są wzajemnie jednakowe.

```
function RowneTeksty ( tekst_1, tekst_2 : string ): boolean ;
var
   i : Byte;
Begin
   if length( tekst_1 ) <> length( tekst_2 )
   RowneTeksty := FALSE
   else
      begin
         RowneTeksty := TRUE;
         for  i := 1 to Length( tekst_1 ) do
            if  tekst_1[ i ] <> tekst_2[ i ] then
               RowneTeksty := FALSE;
      end;
End;
```

Funkcje i procedury standardowe przeznaczone do pracy z łańcuchami

function LENGTH (S:**string**) : Integer;

Zwraca aktualną długość łańcucha **S**. Length(S) → Ord(S[0])

function CONCAT(S1 [, S2, ... Sn] : **string**) : **string**;

Łączy kilka łańcuchów w jeden. L : = Concat(S1,S2); → L := S1 + S2;

function COPY(S: **string**; Poz: Integer; Dlugosc : Integer):**string**;

Zwraca wycinek (podłańcuch) łańcucha **S**.

function POS(S1,S2 : **string**) : Byte;

Sprawdza, czy w łańcuchu **S2** znajduje się podłańcuch **S1** i zwraca numer jego pozycji początkowej. Jeżeli w **S2** nie ma podłańcucha **S1**, to zwracane jest 0.

procedure DELETE(**var** S: **string**; Poz: Integer; Dlugosc: Integer);

Wycina podłańcuch z łańcucha **S**.

procedure INSERT(S1 : **string**; **var** S2: **string**; N: Integer);

Wstawia do łańcucha **S2** podłańcuch **S1**

procedure STR(X [: Dlugosc[: Miejsca_dziesiętne]]; **var** S: **string**);

Przekształca daną **X** dowolnego typu liczbowego na łańcuch.

procedure VAL(S: **string**; **var** X; **var** Kod: Integer);

Przekształca łańcuch tekstowy **S**, na liczbę **X**.

stary = ’ma’ → nowy = ’miala’
tekst = ’Ala ma kota’ → ’Ala miala kota’

```
function ZamienFragmenty( tekst, stary, nowy : string  ): string;
var
   pozycja : byte;
begin
   if stary<>nowy then
      repeat
         pozycja := Pos( stary, tekst );
         if pozycja > 0 then
            begin
               delete( tekst, pozycja, length(stary) );
               insert( nowy, tekst, pozycja );
            end;
      until pozycja = 0;
   ZamienFragmenty := tekst;
end;


function ObetnijTekst_1( tekst : string; nowa_dlugosc:byte  ): string;
begin
   if nowa_dlugosc < length( tekst ) then
      tekst[ 0 ] := nowa_dlugosc;
   ObetnijTekst_1 := tekst;
end;


function ObetnijTekst_2( tekst : string; nowa_dlugosc:byte  ): string;
begin
   ObetnijTekst_2 := Copy( tekst, 1, nowa_dlugosc );
end;


function ObetnijSpacje( tekst ): string;
var
   poczatek : byte;
begin
   poczatek := 1;
   while (tekst[ poczatek ] = #32) and (poczatek < length( tekst)) do
      Inc( poczatek );
   Delete( tekst, 1, poczatek−1 );
   ObetnijSpacje := tekst;
end;
```