TABLICE W JĘZYKU C/C++

Ogólna postać definicji tablicy:

typ_elementu nazwa_tablicy [wymiar_1][wymiar_2] . . . [wymiar_N] ;

np.

```
int  tablica [ 10 ];          // 10-cio elementowa tablica liczb całkowitych
char tekst [ 255 ];           // 255-cio elementowa tablica znaków
float macierz [ 5 ] [ 2 ];    // dwuwymiarowa tablica: 5 wierszy po 2 kolumny,
```

UWAGA:

⇒ w języku C tablice są zawsze indeksowane od zera

np. pierwszym elementem tablicy «macierz» jest: **macierz[0][0]**

a ostatnim elementem jest:**macierz[wymiar_1 – 1][wymiar_2 – 1]**

tzn. **macierz[4][1]**

⇒ w języku C nie jest sprawdzana zgodność indeksu z wymiarami tablicy !!!
często jest to przyczyną trudnych do wykrycia błędów.
np. odwołanie: **macierz[1][2]** zwróci w rzeczywistości wartość pierwszego elementu z trzeciego wiersza tzn. **macierz[2][0]**

0 , 0	**0 , 1**
1 , 0	**1 , 1**
2 , 0	**2 , 1**
3 , 0	**3 , 1**
4 , 0	**4 , 1**

1 , 2

reprezentacja tej macierzy ↓ w pamięci komputera

0 , 0	**0 , 1**	**1 , 0**	**1 , 1**	**2 , 0**	**2 , 1**	**3 , 0**	**3 , 1**	**4 , 0**	**4 , 1**

↑

macierz[1][2]

⇒ Obszar pamięci zajmowany przez tablicę musi być mniejszy od 64 kB

W implementacji C++ firmy Borland to ograniczenie można obejść używając przy definicji tablicy słowo kluczowe **huge** .

np. definicja: **long double huge** tab[20000];

jest poprawna, chociaż tablica «tab» zajmuje 200 000 bajtów ≈ 196 kB

Definicję tablicy można połączyć z inicjacją jej zawartości:

```
int   tab[ 10 ];     // ← sama definicja bez inicjacji
int   tab_inicjowana[ 10 ]  =  { 20, -3, 12, 1, 0, 7, -5, 100, 2, 5 };
char  tab_znakow[ 5 ]  =  { 'a', 'B', '\n', '1', '\0' };
float macierz_A[ 3 ][ 2 ]  =  { {1,1}, {3.5,7.0}, {-15,100} };
float macierz_B[ 3 ][ 2 ]  =  { 1, 1, 3.5, 7.0, -15, 100 };
```

⇒ Kolejne „inicjatory" zawsze wstawiane są do kolejnych „komórek" tablicy (w związku z tym można pominąć wewnętrzne nawiasy klamrowe).

⇒ Jeżeli lista inicjatorów jest krótsza niż ilość elementów tablicy to pozostałe elementy są uzupełniane zerami lub wskaźnikami NULL

np. definicja:

```
int  tab[ 10 ] = { 20, -3, 12, 1 };
```

jest równoważna:

```
int  tab[ 10 ] = { 20, -3, 12, 1, 0, 0, 0, 0, 0, 0 };
```

a definicja:

```
float  macierz[ 3 ][ 2 ] = { {1}, {3.5,7.0} };
```

jest równoważna:

```
float  macierz[ 3 ][ 2 ] = { {1,0}, {3.5,7.0}, {0,0} };
```

lub:

```
float  macierz[ 3 ][ 2 ] = { 1, 0, 3.5, 7.0, 0, 0 };
```

⇒ W języku C inicjatorami muszą być stałe, natomiast w języku C++ inicjatorami mogą być zarówno stałe jak i zmienne.

Wykorzystanie stałych do definiowania ilości elementów tablicy:

```
int   tablica [ 100 ] ;                  // rozmiar zadany bezpośrednio

#define ROZMIAR  100                     // definicja stałej w stylu języka C
int   tablica [ ROZMIAR ] ;

const   ROZMIAR_2 = 100 ;                // definicja stałej w stylu języka C++
int   tablica_2 [ ROZMIAR_2 ] ;

for( int i=0 ; i < ROZMIAR ; i++ )       // przykład dalszego wykorzystania stałej
   printf ( "%d" , tablica[ i ] );
```

Przypisywanie / odczytywanie wartości elementów tablicy

```
void main( )
{
   const  ROZM = 4 ;
   int      Tab [ ROZM ] ;

   // bezpośrednie przypisanie wartości
   Tab[ 0 ] = 0 ;
   Tab[ 1 ] = 10 ;
   Tab[ 2 ] = - 20 ;
   Tab[ 3 ] = 3 ;

   // wczytanie zawartości z klawiatury
   scanf( "%d" , &Tab[ 0 ] ) ;
   scanf( "%d %d" , &Tab[ 1 ] , &Tab[ 2 ] ) ;
   printf( " Podaj 4 element tablicy = " );
   scanf( "%d" , &Tab[ 3 ] ) ;

   // wykorzystywanie i wyświetlanie zawartości elementów tablicy
   long suma = Tab[0] + Tab[1] + Tab[2] + Tab[3] ;
   printf( " Tab[1] = %5d " , Tab[0] );
   printf( " Tab[2] = %5d " , Tab[1] );
   printf( " Tab[3] = %5d " , Tab[2] );
   printf( " Tab[4] = %5d " , Tab[3] );

   // pośrednie zadawanie wartości indeksu za pomocą zmiennej pomocniczej
   int  i = 2 ;
   Tab[ i ] = 10;          // równoważne poleceniu:   Tab[ 2 ] = 10;

   // zadawanie indeksu elementu z klawiatury
   printf( " Podaj indeks elementu którego wartość chcesz wczytać " );
   scanf( "%d" , &i );

   printf( " Podaj nową wartość  Tab[ %d ] = " , i );
   scanf( "%d" , &Tab[ i ] );

   printf( " Nowa wartość Tab[ %d ] wynosi %d " , i , Tab[ i ] );

}
```

Zastosowanie instrukcji repetycyjnej ”for” do operacji na tablicach

```
#include <stdio.h>
void main( void )
  {
    #define  ROZMIAR  10
    float  tablica[ ROZMIAR ];              // definicja tablicy liczb rzeczywistych
    int  i ;

    // inicjowanie zawartości tablicy liczbami parzystymi: 0, 2, 4, 6, ...
    for( i = 0 ; i < ROZMIAR ; i++ )
      tablica[ i ] = 2*i ;

    // wczytanie zawartości elementów tablicy z klawiatury
    for( i = 0 ; i < ROZMIAR ; i++ )
      {
        printf( " Podaj  Tab[%2d] = ", i+1 );
        scanf( " %f " , &tablica[ i ] );
      }

    // wyświetlenie zawartości elementów tablicy
    for( i = 0 ; i < ROZMIAR ; i++ )
      printf( " Tab[%2d] = %10.3f ", i+1 , tablica[ i ] );

    // zsumowanie wartości elementów tablicy
    float suma = 0 ;
    for( i = 0 ; i < ROZMIAR ; i++ )
      suma = suma + tablica[ i ];           //   suma += tablica[ i ];
    printf( "Suma wartości elementów tablicy wynosi: %.2f " , suma );

    // zliczenie ilości elementów o dodatnich wartościach
    int ilosc = 0 ;
    for( i = 0 ; i < ROZMIAR ; i++ )
      if( tablica[ i ] > 0 )
        ilosc = ilosc + 1 ;                 //  ilość += 1;     lub    ilość++;
    if( ilość>0 )
      printf( "Ilość dodatnich elementów = %d " , ilosc );
    else
      printf( "W tablicy nie ma ani jednego dodatniego elementu " );
}
```

Przykłady prostych algorytmów „tablicowych”

```
#include <stdio.h>
void main( void )
  {
    #define  ROZMIAR  10
    int  tab[ ROZMIAR ];
    int i, ilosc, max, poz;
    long suma;
    double srednia;
    for( i = 0 ; i < ROZMIAR ; i++ )          //--------- wczytanie liczb z klawiatury
      {
        printf( ”Tab[%2d] = ”, i+1 );
        scanf( ”%d” , &tablica[ i ] );
      }
    i=0;                    //-------------------------- zliczenie elementów niezerowych
    ilosc=0;
    while( ROZMIAR – i )
      if( tab[i++] )
        ilosc++;
    suma=0;          //----------- wyznaczenie średniej z elementów niezerowych
    ilosc=0;
    i=0;
    do
      if( tab[ i ] != 0 )
        {
          suma += tab[ i ];
          ilosc++;
        }
    while( ++i < ROZMIAR );
    if( ilosc )
      {
        srednia = (double)suma / ilosc;
        printf( "\nSrednia niezerowych = %.2f" , srednia );
      }
    else
      printf( "\nNie ma elementow niezerowych" );
    max=tab[0];              //----------- wyznaczenie wartości i pozycji maksimum
    poz=0;
    for( i=1; i<ROZMIAR ; i++ )
      if( max<tab[ i ] )
        {
          max = tab[ i ];
          poz = i ;
        }
    printf( "\nNajwieksza wartosc jest rowna %d" , max );
    printf( "i wystapila na pozycji %d" , poz+1 );
```

Przykłady funkcji operujących na tablicach

```
#include <stdio.h>
#include <conio.h>
#define  ROZMIAR  10


void WczytajTablice( double tablica[ ] )
{
   clrscr();
   printf( " Podaj wartości elementów tablicy \n" );
   for( int i = 0 ; i < ROZMIAR ; i++ )
      {
         printf( "Tab[%2d] = ", i+1 );
         scanf( "%lf" , &tablica[ i ] );
      }
} //---------------------------------------------------------- funkcja WczytajTablicę


void WyswietlTablice( double tablica[ ] )
{
   clrscr();
   printf( " Wartości elementów tablicy są równe: \n" );
   for( int i = 0 ; i < ROZMIAR ; i++ )
      printf( "Tab[%2d] = %f", i+1 , tablica[ i ] );
   printf( " Nacisnij dowolny klawisz" );
   getch();
} //---------------------------------------------------------- funkcja WyswietlTablicę


void DodajTablice( double wejscie_1[ ] , double wejscie_1[ ], double wynik [ ] )
{
   for( int i = 0 ; i < ROZMIAR ; i++ )
      wynik[ i ] = wejscie_1[ i ] + wejscie_2[ i ] ;
} //---------------------------------------------------------- funkcja DodajTablice


void main( void )
   {
      double  A[ ROZMIAR ] ;
      double  B[ ROZMIAR ],  C[ ROZMIAR ] ;

      WczytajTablice( A );
      WyswietlTablice( A );

      WczytajTablice( B );
      DodajTablice( A, B, C );
      WyswietlTablice( C );
}
```