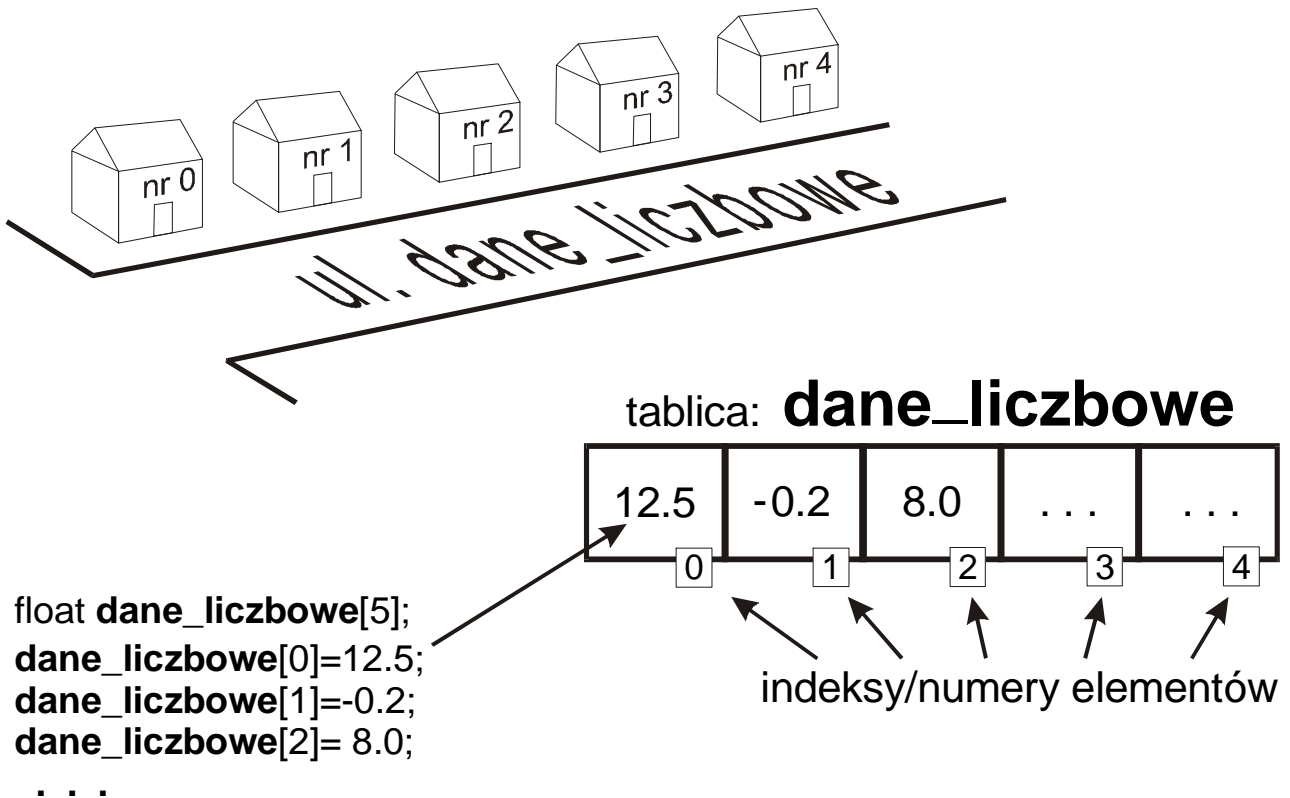


TABLICE W JĘZYKU C/C++



Tablica → jest reprezentacją umożliwiającą zgrupowanie kilku danych **tego samego typu** i odwoływanie się do nich za pomocą wspólnej nazwy. Jest to jeden z najczęściej wykorzystywanych typów danych.

Ogólna postać definicji tablicy:

```
typ_elementu nazwa_tablicy [ liczba_elementów ] ;
```

np.

```
float dane_liczbowe[ 5 ]; // 5-cio elementowa tablica liczb rzeczywistych
int tab[ 10 ]; // 10-cio elementowa tablica liczb całkowitych
char tekst[ 255 ]; // 255-cio elementowa tablica znaków
```

```
double (*funkcje[ 20 ]) (double,double); // tablica 20 wskaźników na funkcje
```

Uwagi o tablicach:

- Ważną cechą tablic jest reprezentacja w postaci **spójnego obszaru pamięci** oraz równomierne rozmieszczenie kolejnych elementów bezpośrednio jeden po drugim.

Dzięki takiej reprezentacji możliwe jest **szybkie wyliczenie położenia** danego elementu w pamięci operacyjnej (na podstawie jego numeru porządkowego - indeksu) oraz znaczne **skrócenie kodu** przetwarzającego duże tablice poprzez zastosowanie instrukcji pętlowych.

- Elementy tablicy są **indeksowane od zera !**
- W języku C i C++ **nie jest sprawdzana poprawność (zakres) indeksów!** Często jest to przyczyną trudnych do wykrycia błędów. Na przykład
przy definicji: `float dane_liczbowe[5];`
instrukcja: `dane_liczbowe[5]=10.5;`
niszczy / zamazuje zawartość pamięci zaraz po ostatnim elemencie tej tablicy.
- Nazwa tablicy jest jednocześnie adresem pierwszego elementu tej tablicy, tzn.
`nazwa_tablicy == &nazwa_tablicy[0]`
- Zwykła tablica nie przechowuje informacji o liczbie swoich elementów.
Uwaga! Polecenie: **`sizeof()`** nie zwraca rozmiaru w sensie liczby elementów.

Definicja tablicy wielowymiarowej:

```
typ_elementu nazwa_tablicy [wymiar_1] [wymiar_2] [wymiar_3] . . . ;
```

np.

```
char kostka_Rubika [ 3 ][ 3 ][ 3 ];
```

```
float macierz [ 5 ] [ 2 ]; // ← dwuwymiarowa tablica: 5 wierszy po 2 kolumny,
```

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

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]

Definicje tablicy można połączyć z inicjalizacją jej zawartości:

```
int tab[ 10 ]; // ← sama definicja bez inicjalizacji
int tab_inicjalizowana[ 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 „inicjalizatory” zawsze wstawiane są do kolejnych „komórek” tablicy (w związku z tym można pominąć wewnętrzne nawiasy klamrowe).

⇒ Jeżeli lista inicjalizatorów 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 inicjalizatorami muszą być stałe, natomiast w języku C++ inicjalizatorami 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 int 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 ] );
```

Podstawowe operacje na elementach tablicy

```
#include <stdio.h>
int 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 ] ) ;

    // prymitywne sumowanie wartości elementów tablicy
    long suma = Tab[0] + Tab[1] + Tab[2] + Tab[3] ;

    // wyświetlanie zawartości elementów tablicy
    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>
int main( void )
{
    #define ROZMIAR 10
    float tablica[ ROZMIAR ];           // definicja tablicy liczb rzeczywistych
    int i ;

    // inicjalizacja zawartości tablicy kolejnymi 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" );
}
}
```

cd. Przykłady algorytmów „tablicowych” (z pętlami typu „for”)

```
#include <iostream>
using namespace std;
int main( void )
{
    const int ROZMIAR = 10;
    int i , tablica[ ROZMIAR ];

    for( i = 0 ; i < ROZMIAR ; i++ ) //----- wczytanie liczb z klawiatury
    {
        cout << "Tab[" << (i+1) << "] = ";
        cin >> tablica[ i ];
    }

    int ilosc=0; //----- zliczanie elementów niezerowych
    for( i = 0 ; i < ROZMIAR ; i++ )
        if( tablica[ i ] )
            ilosc++;

    int suma=0; //----- wyznaczenie średniej z ujemnych wartości
    ilosc=0;
    for( i = 0 ; i < ROZMIAR ; i++ )
        if( tablica[ i ] < 0 )
        {
            suma += tablica[ i ];
            ilosc++;
        }

    if( ilosc )
    {
        srednia = (double)suma / ilosc;
        cout << endl << "Srednia ujemnych = " << srednia ;
    }
    else
        cout << endl << "Nie ma elementow o ujemnych wartościach" ;

    int max=tablica[0]; // wyznaczenie wartości i pozycji maksimum
    int poz=0;
    for( i=1; i<ROZMIAR ; i++ )
        if( max < tablica[ i ] )
        {
            max = tablica[ i ];
            poz = i ;
        }
    cout << endl << "Najwieksza wartosc jest rowna: " << max;
    cout << endl << "i wystapila na pozycji: " << (poz+1);
}
```

Negatywne przykłady programowania → z innymi pętlami niż „for”

```
#include <stdio.h>          // Uwaga: przetwarzanie tablic innymi pętlami niż „for”
int main( void )          // często pogarsza czytelność kodu
{
    #define ROZMIAR 10
    int i=0 , tab[ ROZMIAR ];

    while(i < ROZMIAR)      //----- wczytanie liczb z klawiatury
    {
        printf( "Tab[%2d] = ", i+1 );
        scanf( "%d" , &tab[ i ] );    i = i+1;
    }

    i=0;
    int ilosc=0;           //----- zliczanie elementów niezerowych
    while( ROZMIAR – i )
        if( tab[i++] )
            ilosc++;

    int suma=0;           //----- wyznaczenie średniej z ujemnych wartości
    ilosc=0;
    i=0;
    do
        if( tab[ i ] < 0 )
        {
            suma += tab[ i ];
            ilosc++;
        }
    while( ++i < ROZMIAR );
    if( ilosc )
    {
        srednia = (double)suma / ilosc;
        printf( "\nSrednia ujemnych = %.2f" , srednia );
    }
    else
        printf( "\nNie ma elementow o ujemnych wartościach" );

    int max=tab[0];       //----- wyznaczenie wartości i pozycji maksimum
    int poz=0;
    while( --i           // ← z poprzedniej pętli pozostało i==11
        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 WczytajTablice

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 WyswietlTablice

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

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

    WczytajTablice( A );
    WyswietlTablice( A );

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