Tablice

Tablice jednowymiarowe

Jeżeli nasz zestaw danych składa się z wielu drobnych elementów tego samego rodzaju, jego najbardziej naturalnym ekwiwalentem w programowaniu będzie tablica.

Tablica (ang. `array`) to zespół równorzędnych zmiennych, posiadających wspólną nazwę. Jego poszczególne elementy są rozróżnianie poprzez przypisane im liczby - tak zwane indeksy.
Każdy element tablicy jest więc zmienną należącą do tego samego typu. Nie ma tutaj żadnych ograniczeń: może to być liczba (w matematyce takie tablice nazywamy wektorami), łańcuch znaków (np. lista uczniów lub pracowników), pojedynczy znak, wartość logiczna czy inny typ danych z wyjątkiem typu referencyjnego lub `void`. W szczególności, elementem tablicy może być także inna tablica!

Zadeklarowanie tablicy przypomina analogiczną operację dla zwykłych (skalarnych) zmiennych. Może zatem wyglądać na przykład tak:

```
int tablica[5];
```

Najpierw piszemy nazwę wybranego typu danych, a później oznaczenie samej zmiennej (w tym przypadku tablicy - to także jest zmienna). Nowością jest tu para nawiasów kwadratowych, umieszczona na końcu deklaracji. Wewnątrz niej wpisujemy rozmiar tablicy, czyli ilość elementów, jaką ma ona zawierać. Tutaj jest to 5, a zatem z tylu właśnie liczb (każdej typu `int`) będzie składała się nasza świeżo zadeklarowana tablica.

Nadajmy teraz jakieś wartości kolejnym elementom zadeklarowanej tablicy:

```
tablica[0] = 1;
tablica[1] = 2;
tablica[2] = 3;
tablica[3] = 4;
tablica[4] = 5;
```

Tym razem także korzystamy z nawiasów kwadratowych. Teraz jednak używamy ich, aby uzyskać dostęp do konkretnego elementu tablicy, identyfikowanego przez odpowiedni indeks. W C++ pierwszy element tablicy ma indeks równy zero (patrz przykład wyżej). Na postawie tego przykładu zatem możemy sformułować bardziej ogólną zasadę:

Tablica mieszcząca n elementów jest indeksowana wartościami 0, 1, 2, …, n - 2, n - 1. Z regułą tą wiąże się też bardzo ważne ostrzeżenie:

W tablicy n-elementowej nie istnieje element o indeksie równym n. Próba dostępu do niego jest bardzo częstym błędem, zwanym przekroczeniem indeksów (ang. subscript out of bounds). Poniższa linijka kodu spowodowałaby zatem błąd podczas działania programu i jego awaryjne zakończenie:

```
tablica[5] = 6; // BŁĄD!!!
```

Krytyczne spojrzenie na zaprezentowany kilka akapitów wyżej kawałek kodu może prowadzić do wniosku, że idea tablic nie ma większego sensu. Przecież równie dobrze można byłoby zadeklarować 5 zmiennych i zająć się każdą z nich osobno - podobnie jak czynimy to teraz z elementami tablicy.

```
int liczba1, liczba2, liczba3, liczba4, liczba5;
liczba1 = 1;
liczba2 = 2;
// itd.
```

Takie rozumowanie jest pozornie słuszne… ale na szczęście, tylko pozornie! Użycie pięciu instrukcji - po jednej dla każdego elementu tablicy - nie było bowiem najlepszym rozwiązaniem. O wiele bardziej naturalnym jest odpowiednia pętla for:

```
for (int i = 0; i < 5; ++i) // drugim warunkiem może być też i <= 4
      tablica[i] = i + 1;
```

Jej zalety są oczywiste: niezależnie od tego, czy nasza tablica składa się z pięciu, pięciuset czy pięciu tysięcy elementów, przytoczona pętla jest w każdym przypadku niemal identyczna!
Tajemnica tego faktu tkwi rzecz jasna w indeksowaniu tablicy licznikiem pętli, i. Przyjmuje on odpowiednie wartości (od zera do rozmiaru tablicy minus jeden), które pozwalają zająć się całością tablicy przy pomocy jednej tylko instrukcji! Taki manewr nie byłby możliwy, gdybyśmy używali tutaj pięciu zmiennych, zastępujących tablice. Ich „indeksy" musiałyby być bowiem stałymi wartościami, wpisanymi bezpośrednio do kodu. Nie dałoby się zatem skorzystać z pętli for w podobny sposób, jak to uczyniliśmy w przypadku tablic.

Inicjalizacja tablicy

Kiedy w tak szczegółowy i szczególny sposób zajmujemy się tablicami, łatwo możemy zapomnieć, iż w gruncie rzeczy są to takie same zmienne, jak każde inne. Owszem, składają się z wielu pojedynczych elementów („podzmiennych"), ale nie przeszkadza to w wykonywaniu nań większości znanych nam operacji. Jedną z nich jest inicjalizacja.
Dzięki niej możemy chociażby deklarować tablice będące stałymi. Tablicę możemy zainicjalizować w bardzo prosty sposób, unikając przy tym wielokrotnych przypisań (po jednym dla każdego elementu):

```
int tablica[5] = { 1, 2, 3, 4, 5 };
```

Kolejne wartości wpisujemy w nawiasie klamrowym, oddzielając je przecinkami. Zostaną one umieszczone w następujących po sobie elementach tablicy, poczynając od początku. Tak więc `tablica[0]` będzie miał wartość 1, `tablica[1]` -2, itd. Uzyskamy identyczny efekt, jak w przypadku poprzednich pięciu przypisań.
Interesującą nowością w inicjalizacji tablic jest możliwość pominięcia ich rozmiaru:

```
char systemyOperacyjne[] = {'W','i','l','k'};
```

W takiej sytuacji kompilator „domyśli się" prawidłowej wielkości tablicy na podstawie ilości elementów, jaką wpisaliśmy wewnątrz nawiasów klamrowych (w tzw. inicjalizatorze). Tutaj będą to oczywiście cztery litery.

Inicjalizacja jest więc całkiem dobrym sposobem na wstępne ustawienie wartości kolejnych elementów tablicy - szczególnie wtedy, gdy nie jest ich zbyt wiele i nie są one ze sobą jakoś związane. Dla dużych tablic nie jest to jednak efektywna metoda; w takich wypadkach lepiej użyć odpowiedniej pętli `for`.

Zadanie 1.
Stworzyć plik `tablice1.cpp` i wpisać w nim następujący kod:

```
#include <iostream>
using namespace std;
int main()
{
   int tablica[5];
   tablica[0] = 1;
   tablica[1] = 2;
   tablica[2] = 3;
   tablica[3] = 4;
   tablica[4] = 5;



   for(int i = 0; i <= 4; i++)
```

```
   {
      cout << "Element["<< i <<"]: " << tablica[i] << endl;
   }
   cout << "\n";
   int tablica1[5] = {1,2,3,4,5};
   for(int i = 0; i <= 4; i++)
   {
      cout << "Element["<< i <<"]: " << tablica1[i] << endl;
   }
   system("pause");
   return 0;
}
```

następnie skompilować plik oraz uruchomić program.

W praktyce inicjalizacja tablicy „na sztywno” jest mało przydatna. Chcemy np. obliczyć sumę wszystkich elementów ciągu liczbowego, ale nie dla jednego konkretnego ciągu tylko za każdym razem dla innego. Wygodnie jest wtedy wczytywać elementy takiego ciągu z klawiatury, co obrazuje poniższy program.

Zadanie 2.
Stworzyć plik `tablice2.cpp` i wpisać w nim następujący kod:

```
#include <iostream>
using namespace std;

int main()
{
   const int rozmiar=5;
   int tablica[rozmiar];
   cout << "Podaj elementy tablicy: " << endl;
   for(int i = 0; i <= 4; i++)
   {
      cout << "Element["<< i <<"]: ";
      cin >> tablica[i];
   }
   cout << "\nWprowadzone elementy tablicy to: " << endl;
   for(int i = 0; i <= 4; i++)
   {
      cout << tablica[i] << "  ";
   }
   cout << endl;
   system("pause");
   return 0;
}
```

następnie skompilować plik oraz uruchomić program.

Zadanie 3.
Stworzyć plik `tablice3.cpp` i wpisać w nim następujący kod (największy element w tablicy):

```
#include <iostream>
using namespace std;
int main()
{
   int tablica[5];
   int max;
   cout << "Podaj elementy tablicy:  " << endl;
   for(int i = 0; i <= 4; i++)
   {
      cout << "Element["<< i <<"]: ";
      cin >> tablica[i];
   }
   max = tablica[0];
   for(int i = 0; i <= 4; i++)
   {
      if(tablica[i] > max)
         max = tablica[i];
   }
   cout << "\nNajwiekszy element w tablicy to: " << max << endl;
   system("pause");
   return 0;
}
```

następnie skompilować plik oraz uruchomić program.

Zadanie 4.
Napisać program, który wczytuje tablicę 8 liczb rzeczywistych i wypisuje jej elementy na ekranie monitora. Następnie program szuka elementu najmniejszego tablicy oraz miejsca (indeksu), na którym ten element się znajduje oraz wypisuje je na ekranie.

Zadanie 5.
Napisać program, który wczytuje tablicę 8 liczb rzeczywistych i wypisuje jej elementy na ekranie monitora. Następnie program oblicza sumę tych elementów tablicy, które należą do przedziału [-5,10) oraz wypisuje ją na ekranie.

Zadanie 6.
Napisać program, który wczytuje tablicę 6 liczb całkowitych i wypisuje jej elementy na ekranie monitora. Następnie program oblicza iloczyn tych elementów tablicy, które są podzielne przez 3 i są dodatnie oraz wypisuje ten iloczyn na ekranie.

Zadanie 7.
Napisać program, który wczytuje tablicę 8 liczb całkowitych i wypisuje jej elementy na ekranie monitora. Następnie program oblicza sumę kwadratów tych elementów tablicy, które przy dzieleniu przez 4 dają resztę 2 lub są niedodatnie oraz wypisuje tę sumę na ekranie.

Zadanie 8.
Napisać program, który dla zadeklarowanej 10-elementowej tablicy liczb całkowitych wypisze sumę liczb parzystych i sumę liczb nieparzystych z tej tablicy.

Zadanie 9.
Napisać program, który wczytuje 10-elementową tablicę wartości całkowitych, a następnie wypisuje informacje:
1. czy te liczby tworzą ciąg arytmetyczny,
2. czy liczby tworzą ciąg naprzemienny.

Zadanie 10.
Napisać program, który dla zadeklarowanej 7-elementowej tablicy liczb całkowitych obliczy średnią arytmetyczną elementów nieujemnych tablicy.

Zadanie 11.
Napisać program, który posortuje metodą elementu największego 7-elementową tablicę liczb całkowitych i wypisze na ekranie tablicę posortowaną.

Tablice wielowymiarowe (dwuwymiarowe)

Oprócz tablic jednowymiarowych w C++ stosuje się także tablice wielowymiarowe. My ograniczymy się do tablic dwuwymiarowych. Z matematycznego punktu widzenia tablica jednowymiarowa (typu liczbowego) to nic innego jak ciąg liczb, natomiast dwuwymiarowa to macierz liczbowa.
Deklaracja takiej tablicy wygląda następująco:

```
typ nazwa[rozmiar1][rozmiar2];
```

Chcąc np. zadeklarować macierz A liczb całkowitych wymiaru 2 na 3 użyjemy zapisu

```
int A[2][3];
```

Z kolei inicjalizacja takiej macierzy może wyglądać następująco:

```
int A[2][3] = { {1, 2, 3}, {3, 4, 8} };
```

lub

```
int A[2][3] = { 1, 2, 3, 3, 4, 8 };
```

Wczytywanie i wypisywanie elementów tablicy dwuwymiarowej

```
#include <iostream>
using namespace std;


int main()
{
   int tablica[4][3];
   cout << "Podaj elementy tablicy: " << endl;
   for(int i = 0; i < 4; i++)
     {
      for(int j = 0; j < 3; j++)
       {
        cout << "Element["<< i <<"]["<<j<<"]: ";
        cin >> tablica[i][j];
```

```
        }
       }
   cout << "\nWprowadzone elementy tablicy to: " << endl;
    for(int i = 0; i < 4; i++)
     {
      for(int j = 0; j < 3; j++)
       {
        cout <<tablica[i][j]<<"  ";
       }
        cout << endl;
      }
   system("pause");
   return 0;
}
```

Zadanie 12.
Napisać program, który wczytuje macierz kwadratową liczb rzeczywistych A wymiaru 3 i wypisuje jej elementy na ekranie monitora. Następnie program oblicza ślad tej macierzy i wypisuje go na ekranie.

Zadanie 13.
Napisać program, który wczytuje macierz kwadratową liczb rzeczywistych A wymiaru n (n podaje użytkownik) i wypisuje jej elementy na ekranie monitora. Następnie program sprawdza, czy dana macierz
a) jest symetryczna czy nie,
b) jest dolnie trójkątna czy nie. Jeśli jest to obliczyć wyznacznik tej macierzy.
Następnie należy wypisać odpowiednią informację na ekranie.

Zadanie 14.
Napisać program, który wczytuje macierze kwadratowe liczb całkowitych A i B wymiaru n (n podaje użytkownik z klawiatury), wypisuje jej elementy na ekranie monitora, a następnie program oblicza sumę oraz iloczyn tych macierzy i wypisuje w ten sposób uzyskane macierze na ekranie.

Zadanie 15.
Napisać program, który wczytuje macierz kwadratową liczb całkowitych A wymiaru 3 i wypisuje jej elementy na ekranie monitora. Następnie program oblicza iloczyn tych elementów tej macierzy, które są podzielne przez 3 lub 4 i wypisuje obliczony iloczyn na ekranie.

Zadanie 16.
Napisać program, który zbada ilość rozwiązań układu dwóch równań z dwiema niewiadomymi metodą wyznaczników.