Podstawy programowania: Laboratorium nr 4

Tablice dynamicznie i statyczne.

2017-2018

mgr inż. Przemysław Walkowiak
dr inż. Michał Ciesielczyk

Instrukcja

W czasie pisania programu pamiętaj o:

1. dbaniu o czytelność kodu (odpowiednie formatowanie kodu, nazewnictwo zmiennych adekwatne do ich znaczenia, komentarze),
2. dbaniu o czytelność interfejsu z użytkownikiem (w sposób jawny pytaj użytkownika jakie dane ma podać oraz opisuj wyniki, które zwracasz),
3. przed fragmentem implementującym poszczególne zadania umieść komentarz: `/*Zadanie X */` oraz wypisz na ekranie analogiczny komunikat (X jest numerem zadania): `std::cout << "Zadanie X"<< std::endl;`,
4. umieszczeniu wszystkich rozwiązań w jednym pliku, chyba, że w poleceniu napisano inaczej.
5. w zadaniach wymagających udzielenia komentarza bądź odpowiedzi, należy umieścić go w kodzie programu (np. w postaci komentarza albo wydrukować na ekranie).

Wprowadzenie

Tablice pozwalają na przechowywanie uporządkowanych danych takiego samego typu. W tego typu kontenerach, poszczególne elementy dostępne są za pomocą indeksu. Indeks ten powinien przyjmować wartości numeryczne od 0 do $n-1$, gdzie n jest rozmiarem tablicy.

Rozmiar tablicy jest albo ustalony z góry (tablice statyczne), albo może się zmieniać w trakcie wykonywania programu (tablice dynamiczne).

Tablice statyczne – std::array

Odpowiednikiem tablic statycznych w C++ jest `std::array`. Rozmiar tablicy statycznej musi być znany w trakcie kompilacji. Aby móc skorzystać z tablic `std::array` w C++ należy załączyć bibliotekę `<array>`, tj. dodać do programu dyrektywę: `#include <array>`.

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

/* ... */

// inicjalizacja 3-elementowej tablicy liczb
// całkowitych wartościami [1,2,3]
array<int, 3> arr {1,2,3};

// inicjalizacja 5-elementowej tablicy liczb zmiennoprzecinkowych
array<float, 5> arr2;
```

```cpp
// pobieranie i-tego elementu z tablicy (licząc od '0')
int i = 1;
int x = arr[i]; // x = 2

// odczytywanie wielkości kolekcji (liczby elementów)
unsigned int n = arr.size(); // n = 3

// ustawianie wartości i-tego elementu tablicy
arr[i] = -1;
```

Tablice dynamiczne – std::vector

Odpowiednikiem tablic dynamicznych w C++ jest std::vector. Aby móc skorzystać z tablic dynamicznych w C++ należy załączyć bibliotekę `<vector>`.

```cpp
#include <vector>
using namespace std;

/* ... */

// inicjalizacja pustego wektora liczb całkowitych
vector<int> v;

// inicjalizacja wektora liczb całkowitych o rozmiarze 10
auto v2 = vector<int>(10);
vector<int> v3(10);

// dodawanie nowej wartości na końcu wektora
v.push_back(8);
int x = 2;
v.push_back(x);

// pobieranie i-tego elementu z wektora (licząc od '0')
int y = v[i];

// odczytywanie wielkości kolekcji (liczby elementów)
unsigned int n = v.size()

// ustawianie wartości i-tego elementu tablicy
v[i] = -1;

// usuwanie i-tego elementu (licząc od '0')
v.erase(v.begin() + i);

// czyszczenie wektora (usuwanie wszystkich elementów)
v.clear();
```

```
// zmiana/ustalenie rozmiaru wektora
// oraz wypełnienie wartościami domyślnymi nowych elementów
v.resize(20);

// zmiana/ustalenie rozmiaru wektora
// oraz wypełnienie wartościami 1337 nowych elementów
v.resize(20, 1337);
```

Pętla range-based for

Pętla range-based for pozwala iterować po wszystkich elementach kolekcji. Przykładowo:

```
std::vector<float> v;
// ...
for (auto e : v) {
    cout << e; // wyswietla kolejne elementy kolekcji
}

std::array<int, 3> a {1,2,3};
// ...
for (auto e : a) {
    cout << e; // wyswietla kolejne elementy kolekcji
}
```

Specyfikator auto pozwala na automatyczne wnioskowanie typu zmiennej na podstawie sposobu w jaki jest inicjalizowana. Jednakże zmienna nadal ma jeden, stały typ przez cały czas działania programu!

Aliasy typów

Definiując zmienne reprezentujące macierz można było zauważyć, że nazwy typów są długie i mogą być niewygodne przy wpisywaniu, zwłaszcza jeżeli taki typ występuje w wielu miejscach.

Aby ułatwić sobie korzystanie z takiego typu można nadać mu "nową" nazwę, alias. W tym celu można wykorzystać słowo kluczowe **using**. Np.:

```
using Matrix = std::vector<std::vector<int>>;
using Vector = std::vector<int>;
using MinMax = std::array<float, 2>;
```

Z takich aliasów korzysta się tak samo jak z każdego innego typu. Np.:

```
void printMatrix(const Matrix& mat) {
    for (int i = 0; i < mat.size(); ++i) {
        // ...
    }
}
```

```
const int N = 10, M = 10;
Matrix m1;
std::vector<std::vector<int>> m2;

resizeMatrix(m1, M, N);
resizeMatrix(m2, M, N);

printMatrix(m1);
printMatrix(m2);
```

Oprócz zwiększenia odporności na potencjalne błędy (literówki, pomyłki, zmiany typów), kod staje się dużo czytelniejszy przy obraniu odpowiedniej nazwy, gdyż wyraża intencje autora. Porównaj np.:

```
void print(const std::vector<std::vector<int>> &mat);
void print(const Matrix &mat);
void print(const Vector &vec);
```

Zadania

Zadanie 1

Zdefiniuj nową tablicę statyczną liczb całkowitych o rozmiarze 100. Umieść w niej kolejno liczby całkowite z przedziału $\langle 100, 199 \rangle$. Następnie, oblicz sumę wszystkich elementów w tablicy. Wyświetl na ekranie wynik działania programu.

Wskazówka Do sumowania wszystkich elementów tablicy możesz wykorzystać pętlę range-based for.

Zadanie 2

Zdefiniuj funkcję `void print(vector<int> v)` wypisującą na ekranie całą zawartość wektora liczb całkowitych w następujący sposób: $[v_0, v_1, ..., v_{n-1}]$, gdzie v_i to kolejne elementy wektora o rozmiarze n. Następnie, przetestuj swoją implementację wyświetlając zawartość przykładowego wektora, np.:

```
vector<int> v{ 1,2,3,4 };
print(v); // powinno wyswietlic na ekranie: [1,2,3,4]
```

Zadanie 3

Zdefiniuj następujące funkcje:

a) `float sum(vector<float> v)` – zwracającą sumę wszystkich elementów wektora,
b) `float average(vector<float> v)` – zwracającą średnią wszystkich elementów wektora,

c) `array<float,2> minmax(vector<float> v)` – zwracającą tablicę, w której pierwszy element to minimum podanego wektora, a drugi element to jego maksimum.

Następnie, wczytaj od użytkownika n liczb zmiennoprzecinkowych (wartość n również wczytaj od użytkownika). Wykorzystując zaimplementowane przez Ciebie funkcje wyświetl statystki wczytanego wektora (sumę, średnią, minimum oraz maksimum).

Zadanie 4

Zdefiniuj funkcję `vector<int> randomVector(unsigned int size, int min, int max)` zwracającą nowy wektor o rozmiarze `size`, którego elementy zostały zainicjalizowane liczbami losowymi z przedziału od `min` do `max`. Wykorzystaj funkcję z poprzedniego zadania do wyświetlenia przykładowego wyniku działania funkcji `randomVector`.

Wskazówka Do generowania liczb losowych możesz wykorzystać następującą funkcję:

```
#include <random>

int randomInt(int min, int max) {
    static default_random_engine e{};
    uniform_int_distribution<int> d(min, max);
    return d(e);
}
```

Zadanie 5

Zainicjalizuj dwa wektory o długości n wartościami losowymi z przedziału $\langle 3; 27\rangle$. Oblicz ich iloczyn skalarny oraz wyświetl całe działanie na ekranie. Wartość n wczytaj od użytkownika. Wykorzystaj funkcje zdefiniowane w poprzednich zadaniach.

Wskazówka 1 iloczyn skalarny:

$$a \cdot b = \sum_{i=1}^{n} a_i b_i, \tag{1}$$

gdzie $a, b \in \mathbb{R}^n, n = 1, 2, \ldots,$

Zadanie 6

Wczytaj od użytkownika dowolny napis. Następnie, wykonaj na nim poniższe operacje:

a) zamień wszystkie litery na wielkie,
b) zamień wszystkie litery na małe,
c) zamień wielkie litery na małe i małe na wielkie.

Przyjmij, że podane zostały wyłącznie znaki z tablicy ASCII.

Dodatkowe informacje:
- Tablica ASCII: `http://cplusplus.com/doc/ascii/`.
- Sprawdzanie długości napisu: string::size.

Wskazówka 1 Do wczytywania napisów wykorzystaj funkcję std::getline, np.:

```
string napis;
getline(cin, napis);
```

Wskazówka 2 Przejrzyj jakie użyteczne funkcje znajdują się w bibliotece standardowej: `http://en.cppreference.com/w/cpp/string/byte`. Wykorzystaj je do sprawdzenia czy dany znak jest literą oraz do sprawdzenia/zmiany jej wielkości.

Zadanie 7

Wczytaj od użytkownika napis o długości do 30 znaków, a następnie wyświetl go od końca (napisz funkcję, która odwraca napis). Sprawdź czy podany napis jest palindromem oraz określ jego parzystość/nieparzystość. Przy sprawdzaniu należy zignorować znaki białe.

Palindrom ciąg znaków, który czytany od początku i od końca ma taką samą postać i znaczenie. Np. *anna* (palindrom parzysty), *kajak* (palindrom nieparzysty), *'Nogawka jak wagon'*.

Zadanie 8

Zaimplementuj następujące funkcje do obsługi macierzy liczb całkowitych:

a) `vector<vector<int>> createMatrix(array<unsigned int,2> shape)` – tworzącą nową macierz o wymiarach $shape_0 \times shape_1$ (podanych w tablicy `shape`) oraz z wszystkimi komórkami równymi 0,

b) `vector<vector<int>> randomMatrix(array<unsigned int,2> shape,int min,int max)` – tworzącą nową macierz o wymiarach $shape_0 \times shape_1$ (podanych w tablicy `shape`) oraz z losowymi wartościami od `min` do `max`, oraz

c) `void print(vector<vector<int>> matrix)` – wyświetlającą podaną macierz na ekranie.

Zauważ, że macierz zaprezentowana jest tu jako wektor wektorów: `vector<vector<int>>`.

Przetestują swoją implementację, np.:

```
unsigned int m = 10, n = 5;
vector<vector<int>> A = createMatrix({m, n});
print(A);

vector<vector<int>> B = randomMatrix({m, n}, -10, 15);
print(B);
```

Wskazówka 1 Pamiętaj o odpowiednim zainicjalizowaniu każdego wektora w macierzy. W tym celu możesz wykorzystać funkcję `std::vector::resize`.

Wskazówka 2 Do generowania liczb losowych możesz wykorzystać funkcję `randomInt()` z zadania 4.

Zadanie 9

Zmień deklaracje przygotowanych w poprzednim zadaniu funkcji zamieniając nazwę typu macierzy `vector<vector<int>>` na jego alias `IntMatrix` (który musi być oczywiście wcześniej zdefiniowany – patrz sekcja Aliasy typów). Czy Twoim zdaniem stosowanie tego typu aliasów upraszcza czy utrudnia czytanie kodu źródłowego?

Na następne zajęcia

- Specyfikator **const** – `http://en.cppreference.com/w/cpp/language/cv`.
- Przekazywanie argumentów przez referencję: `http://www.cplusplus.com/doc/tutorial/functions/#reference`.
- Stałe referencje: `http://www.cplusplus.com/doc/tutorial/functions/#constref`.
- Obsługa plików tekstowych: `std::fstream`, `std::ofstream` – `http://www.cplusplus.com/reference/fstream/`.