

Programowanie obiektowe C++

Podstawy STL, kontenery, iteratory

Uwagi

W C++ 11 pojawiło się słowo kluczowe **auto**. Jest to *prośba* do kompilatora, żeby na podstawie typu inicjalizatora automatycznie dobrał typ do przypisywanego obiektu. Często **auto** używa się z iteratorami. Zamiast np. `std::map<std::string, std::string>::iterator it = dict.begin();` można użyć: `auto it = dict.begin();`

W pętlach operujących na iteratorach zwykle stosuje się **preinkrementację** (`++it`). **Postinkrementacja** tworzy tymczasową kopię obiektu iteratora przed jego zwiększeniem, co jest niepotrzebnym kosztem pamięciowym i procesorowym (szczególnie w kontenerach takich jak `std::list` czy `std::map`). Preinkrementacja modyfikuje obiekt w miejscu. W przypadku nowoczesnych kompilatorów (GCC lub MSVC) i użyciu optymalizacji ta zasada traci znaczenie. Można sprawdzić czasy wykonania poszczególnych rodzajów inkrementacji uruchamiając przykład nr 1.

1. Przykład

```
#include <iostream>
#include <list>
#include <chrono>
#include <vector>

using namespace std;

int main() {

    const int elementCount = 100000000; // 100 milionow
    list<int> bigList(elementCount, 1);

    cout << "Starting performance test with " << elementCount << " elements..." << endl <<
        endl;

    auto startPre = chrono::high_resolution_clock::now();
    long long sumPre = 0;

    for(auto it = bigList.begin(); it != bigList.end(); ++it)
    {
        sumPre += *it;
    }

    auto endPre = chrono::high_resolution_clock::now();
    chrono::duration<double, milli> durationPre = endPre - startPre;

    auto startPost = chrono::high_resolution_clock::now();
    long long sumPost = 0;

    for(auto it = bigList.begin(); it != bigList.end(); it++)
    {
        sumPost += *it;
```

```
}

auto endPost = chrono::high_resolution_clock::now();
chrono::duration<double, milli> durationPost = endPost - startPost;

cout << "Pre-increment (++it): " << durationPre.count() << " ms" << endl;
cout << "Post-increment (it++): " << durationPost.count() << " ms" << endl;

double diff = (durationPost.count() - durationPre.count());

if(diff > 0)
    cout << endl << "Pre-increment was faster by: " << diff << " ms" << endl;
else
    cout << endl << "Post-increment was faster by: " << diff << " ms" << endl;

return 0;
}
```

Wprowadź program i przetestuj jego działanie.

Kontener sekwencyjny

2. Przykład

Poniższy przykład ilustruje pracę z kontenerem sekwencyjnym i iteratorem:

```
#include <iostream>
#include <vector>

using namespace std;

int main()
{
    vector<int> numbers = {10, 20, 30, 40};
    numbers.push_back(50);

    cout << "Elementy wektora: ";

    for(auto it = numbers.begin(); it != numbers.end(); it++)
        cout << *it << " ";

    cout << std::endl;

    return 0;
}
```

Wprowadź program i przetestuj jego działanie.

3. Zadanie

Opracuj program z obiektem `std::vector<std::string> guestList`, który będzie realizował następujące zadania:

- użytkownik będzie mógł podać wartość `n` i dodać `n` nazwisk,
- wyświetli listę z użyciem iteratora,

- usunie element listy zdefiniowany przez użytkownika (**guestList.erase()**),
- wyświetli ponownie listę z użyciem pętli *range-based* (**for(const auto &...)**).

4. Przykład

Poniższy przykład ilustruje pracę z kontenerem asocjacyjnym i iteratorem:

```
#include <iostream>
#include <string>
#include <map>

using namespace std;

int main()
{
    map<string, double> prices;

    prices["Jablko"] = 2.50;
    prices["Banan"] = 1.80;

    for(auto it = prices.begin(); it != prices.end(); it++)
        cout << it->first << ": " << it->second << " zł" << endl;

    return 0;
}
```

Wprowadź program i przetestuj jego działanie.

5. Zadanie

Opracuj program, który będzie implementacją słownika angielsko-polskiego przy pomocy kontenera **std::map**. Program powinien umożliwiać wykonywanie takich operacji jak: umieszczanie słowa angielskiego wraz z odpowiednim tłumaczeniem w słowniku, wyszukanie słowa angielskiego (**find()**) i wyświetlenie odpowiednika polskiego, wyświetlenie zawartości słownika na ekranie. Program powinien posiadać proste menu.

Zadania dodatkowe

1. Napisz program, który będzie implementował listę uczniów w klasie przy pomocy kontenera **std::set**. W szczególności program powinien umożliwiać: umieszczanie ucznia na liście, usuwanie ucznia z listy, wyświetlanie listy uczniów, przy użyciu prostego menu.
2. Użyj kontenera **std::vector<int> scores** i pobierz od użytkownika kilka wyników. Posortuj kontener (**sort()**). Przekopiuj dane do kontenera **std::list<int> scoresList**. Używając iteratora i pętli **while**, usuń z listy wszystkie wyniki mniejsze niż 50 (pamiętaj, że funkcja **erase()** automatycznie przesuwa iterator na następny element). Wyświetl przefiltrowaną listę malejąco (użyj **std::reverse** lub iteratorów wstecznych **rbegin()** i **rend()**).
3. Napisz program używający kontenera **std::map** o nazwie **inventory** (nazwa produktu, ilość sztuk), który zostanie zainicjowany kilkoma dowolnymi produktami. Wykorzystaj algorytm **std::max_element** z biblioteki **<algorithm>**, aby znaleźć produkt, którego jest najwięcej. Wskazówka: musisz użyć wyrażenia lambda do porównania wartości (**second**).