Звіт з лабораторної роботи на тему:
“Списки”
студента групи ПС-23-1
Стеценка Данила

Завдання

Написати програму, яка порівнює швидкість роботи ArrayList та LinkedList
Для порівняння слід обрати наступні операції:
1. Заповнення списку даними
2. Доступ до елементу Random Access (доступ за індексом) та Sequential Access (доступ за ітератором)
4. Вставка елементу на початок списку
5. Вставка елементу в кінець списку
6. Вставка елементу в середину списку

Рекомендації:

- Список заповнюється випадковими числами від 0 до N (N - розмір списку)
- Кількість елементів в обох списках однакова (рекомендовано 100000)
- кількість елементів для вставки однакова для обох списків (рекомендовано 1000)
- код повинен бути виложений в Git
- Оцінити складність алгоритмів

Тест

```
1. Filling lists:
Vector: 0.12 ms
List  : 14.22 ms

2. Random Access (every 1000th element):
Vector: 0.00 ms
List  : 1.95 ms

3. Sequential Access (full iteration):
Vector: 0.11 ms
List  : 2.60 ms

4. Insert at front:
Vector: 6.86 ms
List  : 0.22 ms

5. Insert at back:
Vector: 0.04 ms
List  : 0.17 ms

6. Insert in middle:
Vector: 3.77 ms
List  : 0.36 ms
```

Складність алгоритмів:

Операція	std::vector (ArrayList)	std::list (LinkedList)
Заповнення	O(n)	O(n)
Random access	O(1)	O(n)
Ітерація	O(n)	O(n)
Вставка на початок	O(n)	O(1)
Вставка в кінець	O(1)	O(1)
Вставка в середину	O(n)	O(n)

Код программи

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

using namespace std;
using namespace chrono;

const int N = 100000;
const int INSERT_COUNT = 1000;

vector<int> generateRandom(int count) {
    vector<int> data(count);
    mt19937 gen(random_device{}());
    uniform_int_distribution<> dist(0, count);
    for (int& x : data) x = dist(gen);
    return data;
}

template <typename Func>
double measureTime(Func f) {
    auto start = high_resolution_clock::now();
    f();
    auto end = high_resolution_clock::now();
    auto duration = duration_cast<microseconds>(end - start).count();
    return duration / 1000.0; // milliseconds with fractional part
}
```

```
int main() {
    vector<int> data = generateRandom(N);
    vector<int> inserts = generateRandom(INSERT_COUNT);

    vector<int> vec;
    list<int> lst;

    cout << fixed << setprecision(2);

    cout << "1. Filling lists:\n";
    double timeVecFill = measureTime([&]() {
        vec = data;
        });
    double timeListFill = measureTime([&]() {
        lst.assign(data.begin(), data.end());
        });
    cout << "Vector: " << timeVecFill << " ms\n";
    cout << "List  : " << timeListFill << " ms\n\n";

    cout << "2. Random Access (every 1000th element):\n";
    volatile int temp;
    double timeVecRandom = measureTime([&]() {
        for (int i = 0; i < N; i += 1000)
            temp = vec[i];
        });
    double timeListRandom = measureTime([&]() {
        auto it = lst.begin();
        for (int i = 0; i < N; i += 1000) {
            advance(it, (i == 0) ? 0 : 1000);
            temp = *it;
        }
        });
    cout << "Vector: " << timeVecRandom << " ms\n";
    cout << "List  : " << timeListRandom << " ms\n\n";

    cout << "3. Sequential Access (full iteration):\n";
    double timeVecSeq = measureTime([&]() {
        for (auto x : vec) temp = x;
        });
    double timeListSeq = measureTime([&]() {
        for (auto x : lst) temp = x;
        });
    cout << "Vector: " << timeVecSeq << " ms\n";
```

```
cout << "List  : " << timeListSeq << " ms\n\n";

cout << "4. Insert at front:\n";
double timeVecInsertFront = measureTime([&]() {
    for (int x : inserts)
        vec.insert(vec.begin(), x);
    });
double timeListInsertFront = measureTime([&]() {
    for (int x : inserts)
        lst.push_front(x);
    });
cout << "Vector: " << timeVecInsertFront << " ms\n";
cout << "List  : " << timeListInsertFront << " ms\n\n";

cout << "5. Insert at back:\n";
double timeVecInsertBack = measureTime([&]() {
    for (int x : inserts)
        vec.push_back(x);
    });
double timeListInsertBack = measureTime([&]() {
    for (int x : inserts)
        lst.push_back(x);
    });
cout << "Vector: " << timeVecInsertBack << " ms\n";
cout << "List  : " << timeListInsertBack << " ms\n\n";

cout << "6. Insert in middle:\n";
double timeVecInsertMiddle = measureTime([&]() {
    for (int x : inserts) {
        auto midVec = vec.begin() + vec.size() / 2;
        vec.insert(midVec, x);
    }
    });

auto itMid = lst.begin();
advance(itMid, lst.size() / 2);
double timeListInsertMiddle = measureTime([&]() {
    for (int x : inserts)
        lst.insert(itMid, x);
    });

cout << "Vector: " << timeVecInsertMiddle << " ms\n";
cout << "List  : " << timeListInsertMiddle << " ms\n";
```

```
    return 0;
}
```