

Алгоритми та структури даних

**Лабораторна робота №5**

**“Дослідження структур даних зв’язний список та динамічний масив”**

Робота : Лещинського Б.Д. группа КА-07

21.02.2021

Варіант 14

**Мета роботи:**

Ознайомитись і дослідити структури даних зв’язний список та динамічний масив. Поглибити розуміння роботи вказівників, познайомитись з використанням функцій всередині структур. Набути навичок реалізації однозв’язного списку та динамічного масиву мовою програмування C++, порівняти час роботи основних операцій цих структур даних та дослідити їх асимптотичну складність.

**Хід виконання роботи:**

**Завдання:**

1. Створити структуру для зберігання різнотипних даних відповідно до свого варіанту.

2. Реалізувати динамічний масив (з саморозширюванням):

2.1 Створити структуру DynamicArray для зберігання структур свого варіанту та реалізувати в ній основні функції динамічного масиву:

• push\_back() додавання елементу в кінець

• pop\_back() зчитування та видалення елементу з кінця

• get() зчитування n-го елементу

• size() знаходження кількості елементів

• print() виведення всіх елементів з даними, що зберігаються

2.2 Реалізувати додаткові функції для динамічного масиву:

• push\_front() додавання елементу в початок

• pop\_front() зчитування та видалення елементу з початку

3. Реалізувати однозв’язний список:

3.1 Створити структуру Node для базового елементу однозв’язного списку, вона буде містити в собі структуру відповідно до варіанту.

3.2 Створити структуру LinkedList та реалізувати в ній основні функції однозв’язного списку:

• push\_front() додавання елементу в початок

• push\_back() додавання елементу в кінець

• get() зчитування n-го елементу

• pop\_front() зчитування та видалення елементу з початку

• pop\_back() зчитування та видалення елементу з кінця

• size() знаходження кількості елементів

• print() виведення всіх елементів з даними, що зберігаються

4. Продумати реалізацію стеку та черги на базі зв’язного списку, вміти обгрунтовано пояснити та показувати в коді. Реалізовувати не потрібно.

5. Провести порівняння роботи однозв’язного списку та динамічного масиву (кожен крок виконати для обох структур, заміряти та порівняти час кожного кроку та всіх кроків разом):

5.1 Додати в кінець 50000 елементів

5.2 Додати в початок 10000 елементів

5.3 Зчитати 20000 елементів під випадковими індексами

5.4 Видалити 5000 елементів з початку

5.5 Видалити 5000 елементів з кінця

**Код програми :**

#include <iostream>

#include <cstdlib>

#include <ctime>

#include <string>

#include <cmath>

#include <ctime>

using namespace std;

string Names[3] = { "KPIvanlav","KNU Schevchenka","Mogilyanka" };

struct University {

string Name;

int numberStudents;

bool Turnikety;

};

University generateUni() {

University randomUni = { Names[rand() % 3],rand() % 10001,(bool)(rand() % 2) };

return randomUni;

}

struct DynamicArray {

University \*arr = new University[1];

int entities = 0;

int Size = 1;

void push\_back(University uni) {

arr[entities] = uni;

entities++;

if (entities == Size) {

University \*newArr = new University[Size \* 2];

for (int i = 0; i < entities; i++) {

newArr[i] = arr[i];

}

delete[] arr;

arr = newArr;

Size \*= 2;

}

}

University pop\_back() {

entities--;

return arr[entities];

}

void push\_front(University uni) {

for (int i = entities - 1; i >= 0; i--) {

arr[i+1] = arr[i];

}

arr[0] = uni;

entities++;

if (entities == Size) {

University\* newArr = new University[Size \* 2];

for (int i = 0; i < entities; i++) {

newArr[i] = arr[i];

}

delete[] arr;

arr = newArr;

Size \*= 2;

}

}

University pop\_front() {

University front\_poped = arr[0];

for (int i = 0; i < entities - 1; i++) {

arr[i] = arr[i + 1];

}

entities--;

return front\_poped;

}

University get(int n) {

if (n>entities) {

throw "Invalid index\n";

}

else {

return arr[n];

}

}

int size() {

return entities;

}

void print() {

for (int i = 0; i < entities; i++) {

cout << "University #" << i + 1 << ": " << arr[i].Name << " " << "; Number of students: " << arr[i].numberStudents << "; Tyrnikety: " << arr[i].Turnikety << "\n";

}

}

};

struct Node {

University uni;

Node\* next;

};

struct LinkedList {

int entities = 0;

Node\* firstNode = nullptr;

Node\* lastNode = nullptr;

void push\_front(University uni) {

Node\* newFirst = new Node;

newFirst->uni = uni;

if (entities == 0) {

firstNode = newFirst;

lastNode = newFirst;

}

else {

newFirst->next = firstNode;

firstNode = newFirst;

}

entities++;

}

void push\_back(University uni) {

Node\* newLast = new Node;

newLast->uni = uni;

if (entities == 0) {

firstNode = newLast;

lastNode = newLast;

}

else {

lastNode->next = newLast;

lastNode = newLast;

}

entities++;

}

University pop\_front() {

University poped\_front = firstNode->uni;

Node\* newFirst = firstNode->next;

delete firstNode;

firstNode = newFirst;

entities--;

return poped\_front;

}

University pop\_back() {

University poped\_back = lastNode->uni;

Node\* currNode = firstNode;

while (currNode->next != lastNode) {

currNode = currNode->next;

}

delete lastNode;

currNode->next = nullptr;

lastNode = currNode;

entities--;

return poped\_back;

}

University get(int n) {

if (n > entities) throw "Invalid index!";

Node\* nNode = firstNode;

for (int i = 0; i < n; i++) { // nor sure with number of inerations needed

nNode = nNode->next;

}

return nNode->uni;

}

int size() {

return entities;

}

void print() {

Node\* currNode = firstNode;

for (int i = 0; i < entities; i++) {

cout << "University #" << i + 1 << ": " << currNode->uni.Name << " " << "; Number of students: " << currNode->uni.numberStudents << "; Tyrnikety: " << currNode->uni.Turnikety << "\n";

currNode = currNode->next;

}

}

};

int main(){

srand(time(NULL));

DynamicArray uniArr;

LinkedList uniList;

double sTime, fTime, resultTime = 0;

sTime = double(clock());

for (int i = 0; i < 50000; i++) {

uniArr.push\_back(generateUni());

}

fTime = double(clock());

resultTime += (fTime - sTime)/CLOCKS\_PER\_SEC;

cout << "Dynamic array push\_back 50000 element time: " << resultTime << endl;

sTime = double(clock());

for (int i = 0; i < 50000; i++) {

uniList.push\_back(generateUni());

}

fTime = double(clock());

resultTime = (fTime - sTime) / CLOCKS\_PER\_SEC;

cout << "Linked list push\_back 50000 element time: " << resultTime << endl;

cout << endl;

sTime = double(clock());

for (int i = 0; i < 10000; i++) {

uniArr.push\_front(generateUni());

}

fTime = double(clock());

resultTime = (fTime - sTime) / CLOCKS\_PER\_SEC;

cout << "Dynamic array push\_front 10000 element time: " << resultTime << endl;

sTime = double(clock());

for (int i = 0; i < 10000; i++) {

uniList.push\_front(generateUni());

}

fTime = double(clock());

resultTime = (fTime - sTime) / CLOCKS\_PER\_SEC;

cout << "Linked list push\_front 10000 element time: " << resultTime << endl;

cout << endl;

sTime = double(clock());

for (int i = 0; i < 20000; i++) {

uniArr.get(rand() % 60000);

}

fTime = double(clock());

resultTime = (fTime - sTime) / CLOCKS\_PER\_SEC;

cout << "Accessing 20000 random elements of dynamic array time ( get() ) : " << resultTime << endl;

sTime = double(clock());

for (int i = 0; i < 20000; i++) {

uniList.get(rand() % 60000);;

}

fTime = double(clock());

resultTime = (fTime - sTime) / CLOCKS\_PER\_SEC;

cout << "Accessing 20000 random elements of linked list time ( get() ) : " << resultTime << endl;

cout << endl;

sTime = double(clock());

for (int i = 0; i < 5000; i++) {

uniArr.pop\_front();

}

fTime = double(clock());

resultTime = (fTime - sTime) / CLOCKS\_PER\_SEC;

cout << "Deleting 5000 elements of dynamic array from the beginning of it ( pop\_front() ) :" << resultTime << endl;

sTime = double(clock());

for (int i = 0; i < 5000; i++) {

uniList.pop\_front();

}

fTime = double(clock());

resultTime = (fTime - sTime) / CLOCKS\_PER\_SEC;

cout << "Deleting 5000 elements of linked list from the beginning of it ( pop\_front() ) :" << resultTime << endl;

cout << endl;

sTime = double(clock());

for (int i = 0; i < 5000; i++) {

uniArr.pop\_back();

}

fTime = double(clock());

resultTime = (fTime - sTime) / CLOCKS\_PER\_SEC;

cout << "Deleting 5000 elements of dynamic array from the end of it ( pop\_back() ) :" << resultTime << endl;

sTime = double(clock());

for (int i = 0; i < 5000; i++) {

uniList.pop\_back();

}

fTime = double(clock());

resultTime = (fTime - sTime) / CLOCKS\_PER\_SEC;

cout << "Deleting 5000 elements of linked from the end of it ( pop\_back() ) :" << resultTime << endl;

return 0;

}

**Результати роботи програми:**

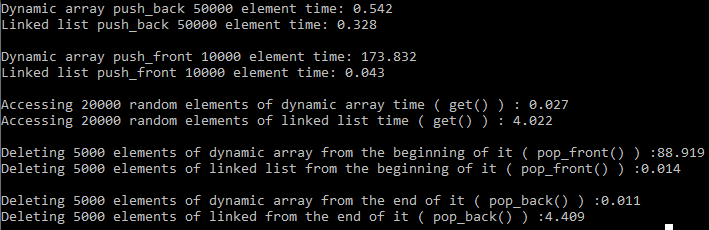


Рис.1 результат роботи програми

Висновки:

Я ознайомився і дослідив структури даних зв’язний список та динамічний масив. Поглибив розуміння роботи вказівників, познайомився з використанням функцій всередині структур. Набув навичок реалізації однозв’язного списку та динамічного масиву мовою програмування C++, порівняв час роботи основних операцій цих структур даних та дослідив їх асимптотичну складність.

При додаванні елемента на початок різниця у часі пояснюється тим, що при додаванні елементу у однозв'язний список на його початок це займає константний час виконання, а випадку динамічного масиву, коли кількість елементів стає рівною кількості пам'яті, доступної масиву на даний мамент, доводиться створювати новий масив з більшою кількістю доступної пам'яті та переписувати елементи, що вже були у минулому масиві. Це у свою чергу має асимптотичну складність О(n), що виражається у довшому часі виконання.

При додаванні елементу на початок, однозв'язний список також виграє у часі, порівняно з динамічним масивом, бо додавання у список, знову ж таки, має константний час виконання, а у масиві доводитсья переміщувати елементи на один у право, щоби додати елемент на початок, це, як і у минулому випадку, має асимптотичну складність О(n).

В зчитуванні елментів значно краще показує себе динамічний масив, маючи для цієї операції складність О(1), тобто константу. Зв'язний список затрачає на це О(n), оскільки доводиться проходити по черзі по адресах наступних елементів, поки не буде досягнутий шуканий.

Видалення елементів з початку виконується набагато швидше у однозв'язному списку та має складність О(1), а у динамічному масиві О(n), бо, знову ж таки, доводиться зміщувати елементи на один, але тепер уліво.

Видалення елементів з кінця, набагато швидше виконується у динамічному масиві (О(1)), у однозв’язному списку це займає О(n), оскільки треба проти майже весь список, аби дістати адресу передостаннього елемента та зробити, щоб останнім став саме він.