**Лабораторна робота №1**

**З дисципліни «Алгоритми та структури даних»**

***«Дослідження структур даних зв’язний список та динамічний масив»***

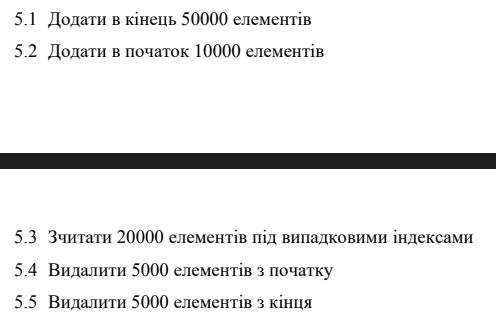
*Виконав  
студент I курсу  
групи ДА-12  
Ніколаєв Роман*

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.

*Варіант 19*

**Хід виконання завдання:**



**Код програми:**

#include <iostream>

#include <cmath>

#include <ctime>

struct University

{

char name;

int students\_amount;

bool turnstile;

University()

{

name = (char)(rand() % 26 + 65);

students\_amount = rand() % 100001;

turnstile = rand() % 2;

};

};

struct DynamicArray

{

int volume;

University\* data;

int size;

const int alpha = 2;

DynamicArray()

{

volume = 1;

size = 0;

data = new University[volume];

}

void Print()

{

for(int j = 0; j < size; j++)

std::cout << data[j].name << " " << data[j].students\_amount << " " << data[j].turnstile << std::endl;

}

void PushBack(University newRecord)

{

if (size == volume) {

volume \*= alpha;

University\* newArray = new University[volume];

for(int j = 0; j < size; j++)

newArray[j] = data[j];

delete[] data;

data = newArray;

}

data[size] = newRecord;

size++;

}

void PushFront(University newRecord)

{

if (size == volume) {

volume \*= alpha;

University\* newArray = new University[volume];

for(int j = 0; j < size; j++)

newArray[j] = data[j];

delete[] data;

data = newArray;

}

for (int l = size; l > 0; l--)

data[l] = data[l - 1];

data[0] = newRecord;

size++;

}

University PopBack()

{

if (size > 0) {

size--;

return data[size];

}

}

University Get(int number)

{

return data[number];

}

int Size()

{

return size;

}

University PopFront()

{

if (size > 0) {

University saveFirst = data[0];

for (int j = 0; j < size; j++)

data[j] = data[j + 1];

size--;

return saveFirst;

}

}

};

struct Node

{

University data;

Node\* next;

};

struct LinkedList

{

Node\* head;

Node\* tail;

int size;

LinkedList()

{

head = NULL;

tail = NULL;

size = 0;

}

void Print()

{

Node\* node = head;

while (node->next != NULL) {

std::cout << node->data.name << " " << node->data.students\_amount << " " << node->data.turnstile << std::endl;

node = node->next;

}

std::cout << node->data.name << " " << node->data.students\_amount << " " << node->data.turnstile << std::endl;

}

void PushFront(University newRecord)

{

Node\* newNode = new Node();

newNode->data = newRecord;

newNode->next = NULL;

if (size == 0)

tail = newNode;

newNode->next = head;

head = newNode;

size++;

}

int Size()

{

return size;

}

void PushBack(University newRecord)

{

Node\* newNode = new Node();

newNode->data = newRecord;

newNode->next = NULL;

if (size == 0)

head = newNode;

else

tail->next = newNode;

tail = newNode;

size++;

}

University Get(int number)

{

if (size > 0) {

Node\* newNode = head;

for (int j = 2; j <= number; j++)

newNode = newNode->next;

return newNode->data;

}

}

University PopFront()

{

if (size > 0) {

University saveRecord = head->data;

head = head->next;

size--;

return saveRecord;

}

}

University PopBack()

{

if (size > 0) {

University saveRecord = tail->data;

Node\* node = head;

while (node->next != tail)

node = node->next;

tail = node;

node->next = NULL;

size--;

return saveRecord;

}

}

};

template <typename T>

T test(T& data){

clock\_t startAll = clock();

clock\_t start1 = clock();

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

University record;

data.PushBack(record);

}

clock\_t finish1 = clock();

clock\_t start2 = clock();

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

University record;

data.PushFront(record);

}

clock\_t finish2 = clock();

clock\_t start3 = clock();

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

if (data.size > 0) {

int n = rand() % data.size + 1;

University result = data.Get(n);

}

}

clock\_t finish3 = clock();

clock\_t start4 = clock();

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

University result = data.PopBack();

}

clock\_t finish4 = clock();

clock\_t start5 = clock();

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

University dresult = data.PopFront();

}

clock\_t finish5 = clock();

clock\_t finishAll = clock();

float time1 = (float(finish1 - start1)) / CLOCKS\_PER\_SEC;

float time2 = (float(finish2 - start2)) / CLOCKS\_PER\_SEC;

float time3 = (float(finish3 - start3)) / CLOCKS\_PER\_SEC;

float time4 = (float(finish4 - start4)) / CLOCKS\_PER\_SEC;

float time5 = (float(finish5 - start5)) / CLOCKS\_PER\_SEC;

float timeAll = (float(finishAll - startAll)) / CLOCKS\_PER\_SEC;

std::cout << "PushBack time: " << time1 << "s" << std::endl;

std::cout << "PushFront time: " << time2 << "s" << std::endl;

std::cout << "Get time: " << time3 << "s" << std::endl;

std::cout << "PopBack time: " << time4 << "s" << std::endl;

std::cout << "PopFront time: " << time5 << "s" << std::endl;

std::cout << "Summary: " << timeAll << "s\n" << std::endl;

}

int main()

{

srand(time(NULL));

DynamicArray array;

LinkedList list;

std::cout << "Dynamic array testing:" << std::endl;

test(array);

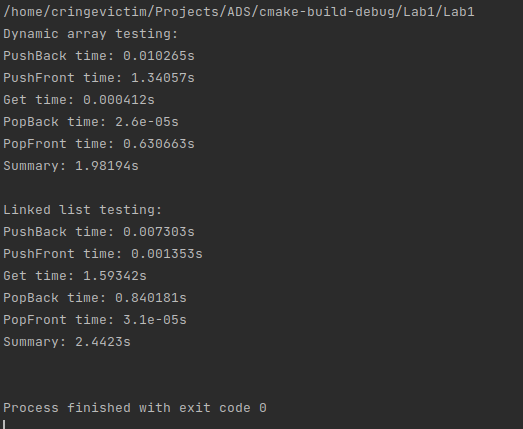
std::cout << "Linked list testing:" << std::endl;

test(list);

return 0;

}

**Результат виконання:**



**Висновки:**

*Виконуючи дану лабороторну роботу було набуто навичок використання динамічного виділення пам’яті для роботи з динамічним масивом та зв’язним списком.*

**Контрольні питання:**

1. *Динамічний масив корисний за необхідності миттєвого доступу до елементів, а однозв’язний список є кращим, коли важливий порядок елементів, а доступ за індексом не є необхідністю.*
2. *У разі, якщо виділена пам’ять заповнена, то для додавання нового елементу, необхідно перевиділити нову частину пам’яті, скопіювати туди попередні дані та додати новий елемент, що і займає незвично великий час.*
3. *Вказівник на хвіст необхідний для швидкого доступу на кінець списку для реалізації PushBack(). Даний метод можна реалізувати і без нього, але асимтотика виконання такої операції завжди буде О(n), де n – к-ть елементів списку. Зі вказівником на хвіст напроти – завжди О(1).*
4. *Пам’ять на стеку зберігає локальні змінні, а пам’ять на купі використовується для зберігання великих структур даних та динамічного виділення пам’яті. Сміття у пам’яті утворюється коли вже не потрібні дані залишаються в пам’яті і не видаляються, та заважають виділенню нової пам’яті (може спричинити так звану фрагментацію пам’яті). Також через це може статися витік пам’яті. Все це запобігається своєчасним звільненням виділеної пам’яті.*
5. *Черга зручно реалізується за допомогою списку через вже наявність основних її операцій – додавання елементів в кінець та видалення з початку за О(1). Така черга може бути не обмежена в розмірі, але буде займати більше пам’яті через зберігання вказівників. Стек також може бути реалізований за допомогою зв’язного списку через можливість видаляти да додавати елементи з початку за О(1), та також може мати необмежений розмір.*
6. *Двозв’язний список дозволяє швидко видаляти та додавати елементи з обох кінців, а також дозволяє зчитувати елементи у зворотньому напрямку, але займає більше пам’яті, а доступ до елементів за індексом все ще залишається довгим.*