МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ

Кафедра системного проектування

**ЗВІТ**про виконання лабораторної роботи №3

«Дослідження структур даних зв'язний список та динамічний масив»

з дисципліни «Алгоритми та структури даних»

Виконав:

cтудент І курсу

групи ДА-02

Гринчишин Д.Б.

Перевірили:

..

Київ – 2020

**Мета**. 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 завдань - 2**

#include <iostream>

#include <ctime>

#include <cmath>

#include <cstdlib>

using namespace std;

struct Hostel

{

int number;

int space;

bool heating;

};

struct DynamicArray

{

int capacity = 5;

const int koef = 2;

Hostel\* arr = new Hostel[capacity];

int size = 0;

void push\_back(Hostel new\_elem)

{

if (size == capacity)

{

capacity \*= koef;

Hostel\* temp\_arr = new Hostel[capacity];

for (size\_t i = 0; i <= size; i++) temp\_arr[i] = arr[i];

delete[] arr;

arr = temp\_arr;

}

arr[size] = new\_elem;

size ++;

}

Hostel pop\_back()

{

Hostel temp = arr[size];

size--;

return temp;

}

Hostel get(int n)

{

if(n >= 0 && n < size) return arr[n];

else

{

printf("no element with index %d\n", n);

exit(1);

}

}

int get\_size(){return size;}

void print()

{

if(size > 0)

{

for (int i = 0; i < size; i++) printf("№: %d, space: %d, heating: %d\n", arr[i].number, arr[i].space, arr[i].heating);

}

else cout << "no elements to print" << endl;

}

void push\_front(Hostel new\_elem)

{

if (size == capacity)

{

capacity \*= koef;

Hostel\* temp\_arr = new Hostel[capacity];

for (int i = 0; i <= size; i++) temp\_arr[i + 1] = arr[i];

delete[] arr;

arr = temp\_arr;

}

else

{

for (int i = size - 1; i >= 0; i = i - 1) arr[i + 1] = arr[i];

}

arr[0] = new\_elem;

size++;

}

Hostel pop\_front()

{

size--;

Hostel temp = arr[0];

for (int i = 0; i < size; i++) arr[i] = arr[i + 1];

return temp;

}

void clean()

{

delete [] arr;

}

};

struct Node

{

Node\* next;

Hostel hostel;

};

struct LinkedList

{

Node\* tail = NULL;

Node\* head = NULL;

void push\_back(Hostel new\_elem)

{

Node\* node = (Node\*)malloc(sizeof(Node));

node->hostel = new\_elem;

node->next = NULL;

if (head != NULL)

{

tail->next = node;

tail = node;

}

else

{

head = node;

tail = node;

}

}

void push\_front(Hostel new\_elem)

{

Node\* node = (Node\*)malloc(sizeof(Node));

node->hostel = new\_elem;

node->next = NULL;

if (head != NULL)

{

node->next = head;

head = node;

}

else

{

head = node;

tail = node;

}

}

int size()

{

if(head == NULL) return 0;

else

{

int counter = 1;

Node\* node = head;

while(node->next != NULL)

{

node = node->next;

counter++;

}

return counter;

}

}

void print()

{

if(head == NULL) cout << "no elements to print" << endl;

else

{

Node\* node = head;

while(1)

{

printf("№: %d, space: %d, heating: %d\n", node->hostel.number, node->hostel.space, node->hostel.heating);

if(node->next != NULL) node = node->next;

else break;

}

}

}

Hostel pop\_front()

{

if(head == NULL)

{

cout << "no elements to pop front" << endl;

exit(1);

}

else

{

Hostel hostel = head->hostel;

head = head->next;

return hostel;

}

}

Hostel pop\_back()

{

Hostel hostel;

if(head == NULL)

{

cout << "no elements to pop back" << endl;

exit(1);

}

else if(head == tail)

{

hostel = head->hostel;

head = NULL;

tail = NULL;

return hostel;

}

else

{

Node\* node = head;

hostel = tail->hostel;

while(1)

{

if(node->next == tail)

{

node->next = NULL;

tail = node;

return hostel;

}

else node = node->next;

}

}

}

Hostel get(int n)

{

if(head == NULL)

{

cout << "no elements in array" << endl;

exit(1);

}

else

{

Node\* node = head;

for(int i = 0;i < n;i++)

{

if(node->next != NULL) node = node->next;

else

{

printf("No element with index %d in array\n", n);

exit(1);

}

}

Hostel hostel = node->hostel;

return hostel;

}

}

void clean()

{

if (head != NULL)

{

Node\* node = head;

Node\* ptr;

while(1)

{

ptr = node;

if(node->next != NULL)

{

node = node->next;

delete ptr;

}

else break;

}

}

}

};

int main()

{

srand(time(0));

LinkedList linked;

DynamicArray dynamic;

Hostel h;

double start;

double d\_start = clock();

double times\_dynamic[6] = {0};

start = clock();

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

{

h = {i, 2, true};

dynamic.push\_back(h);

}

cout << "push back " << (clock() - start) / CLOCKS\_PER\_SEC << endl;

times\_dynamic[0] = clock() - start;

start = clock();

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

{

h = {i, 2, true};

dynamic.push\_front(h);

}

cout << "push front " << (clock() - start) / CLOCKS\_PER\_SEC << endl;

times\_dynamic[1] = clock() - start;

start = clock();

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

{

dynamic.get(rand() % 60000);

}

cout << "get " << (clock() - start) / CLOCKS\_PER\_SEC << endl;

times\_dynamic[2] = clock() - start;

start = clock();

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

{

dynamic.pop\_front();

}

cout << "pop front " << (clock() - start) / CLOCKS\_PER\_SEC << endl;

times\_dynamic[3] = clock() - start;

start = clock();

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

{

dynamic.pop\_back();

}

cout << "pop back " << (clock() - start) / CLOCKS\_PER\_SEC << endl;

times\_dynamic[4] = clock() - start;

cout << "dynamic all time " << (clock() - d\_start) / CLOCKS\_PER\_SEC << endl << endl;

d\_start = clock() - d\_start;

dynamic.clean();

double l\_start = clock();

double times\_linked[6];

start = clock();

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

{

h = {i, 2, true};

linked.push\_back(h);

}

cout << "push back " << (clock() - start) / CLOCKS\_PER\_SEC << endl;

times\_linked[0] = clock() - start;

start = clock();

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

{

h = {i, 2, true};

linked.push\_front(h);

}

cout << "push front " << (clock() - start) / CLOCKS\_PER\_SEC << endl;

times\_linked[1] = clock() - start;

start = clock();

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

{

linked.get(rand() % 60000);

}

cout << "get " << (clock() - start) / CLOCKS\_PER\_SEC << endl;

times\_linked[2] = clock() - start;

start = clock();

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

{

linked.pop\_front();

}

cout << "pop front " << (clock() - start) / CLOCKS\_PER\_SEC << endl;

times\_linked[3] = clock() - start;

start = clock();

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

{

linked.pop\_back();

}

cout << "pop back " << (clock() - start) / CLOCKS\_PER\_SEC << endl;

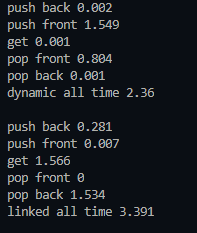
times\_linked[4] = clock() - start;

cout << "linked all time " << (clock() - l\_start) / CLOCKS\_PER\_SEC << endl << endl;

l\_start = clock() - l\_start;

linked.clean();

return 1;}

**Результат виконання програми:  
**

**Висновок:**

В ході виконання лабораторної роботи було реалізовано структури даних зв’язний список та динамічний масив і всі необхідні для роботи з ними функції. Було заміряно час виконання цих алгоритмів та порівняно. З результатів замірів випливає, що структура динамічний масив набагато продуктивніше працює з додаванням та видалянням елементів в кінець масиву та зчитуванні будь-якого елементу масиву. Усі ці операції виконуються за константний відрізок часу. Однозв’зний список швидше працює з додаванням елементів в початок масиву та в кінець масиву і з видалянням елементів з початку масиву. Ці функції виконуються за сталий відрізок часу. Час виконання всіх решти функцій обох структур має лінійну залежність від кількості елементів у масивах.