**Лесечко Олеся(ДА-02)**

Лабораторна робота №1

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

**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: 8**

Структура **Стипендія** має наступні поля: розмір стипендії, номер картки, наявність.

Хід виконання роботи:

A) УМОВА

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

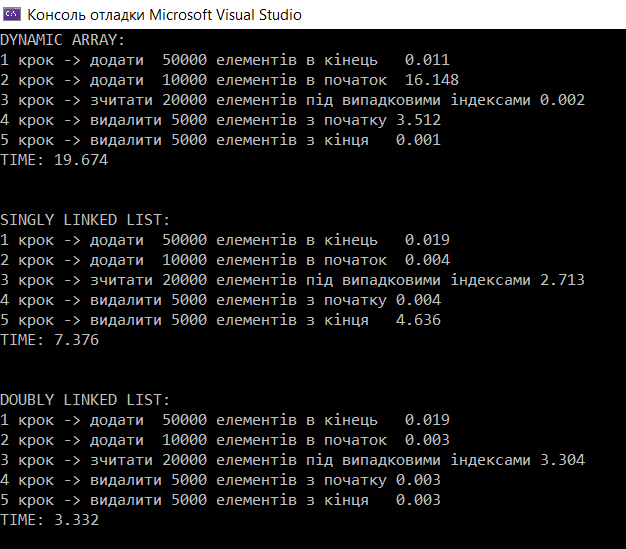
• pop\_back() зчитування та видалення елементу з кінця

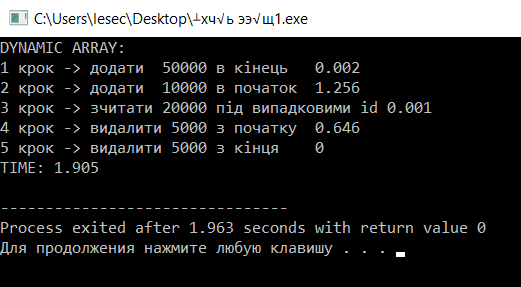
• size() знаходження кількості елементів

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

B) СКРІНШОТ РЕЗУЛЬТАТУ

* Visual Studio



* DEVC++

С) ЛІСТИНГ ПРОГРАМИ (КОД)

ASD\_LAB1.cpp:

#include <iostream>

#include <time.h>

#include <stdlib.h>

#include <time.h>

#include "dynamArrayFunk.h"

#include "linkedListFunk.h"

using namespace std;

int main() {

srand(time(NULL));

setlocale(LC\_ALL, "Russian");

DynamicArray\* myArray = new DynamicArray();

clock\_t Start = clock();

//додати 50000 елементів в кінець

clock\_t step1Start = clock();

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

myArray->push\_back({rand() % 2000, i + 1, random() });

}

clock\_t step1End = clock();

//додати 10000 елементів в початок

clock\_t step2Start = clock();

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

myArray->push\_front({rand() % 2000, i + 50000, random()});

}

clock\_t step2End = clock();

//зчитати 20000 елементів під випадковими індексами

clock\_t step3Start = clock();

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

myArray->get(rand() % 60000 + 1);

}

clock\_t step3End = clock();

//видалити 5000 елементів з початку

clock\_t step4Start = clock();

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

myArray->pop\_front();

}

clock\_t step4End = clock();

//видалити 5000 елементів з кінця

clock\_t step5Start = clock();

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

myArray->pop\_back();

}

clock\_t step5End = clock();

clock\_t End = clock();

cout << "DYNAMIC ARRAY: \n";

float step1Time = (float(step1End - step1Start)) / CLOCKS\_PER\_SEC;

cout << "1 крок -> додати 50000 елементiв в кiнець " << step1Time << endl;

float step2Time = (float(step2End - step2Start)) / CLOCKS\_PER\_SEC;

cout << "2 крок -> додати 10000 елементiв в початок " << step2Time << endl;

float step3Time = (float(step3End - step3Start)) / CLOCKS\_PER\_SEC;

cout << "3 крок -> зчитати 20000 елементiв пiд випадковими iндексами " << step3Time << endl;

float step4Time = (float(step4End - step4Start)) / CLOCKS\_PER\_SEC;

cout << "4 крок -> видалити 5000 елементiв з початку " << step4Time << endl;

float step5Time = (float(step5End - step5Start)) / CLOCKS\_PER\_SEC;

cout << "5 крок -> видалити 5000 елементiв з кiнця " << step5Time << endl;

float time = (float(End - Start)) / CLOCKS\_PER\_SEC;

cout << "TIME: " << time << endl;

//----------------------------------------------------------------------------------------------

SinglyLinkedList stipendListSingly;

Start = clock();

//додати 50000 елементів в кінець

step1Start = clock();

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

stipendListSingly.push\_back({rand() % 2000,i + 1,random()});

}

step1End = clock();

//додати 10000 елементів в початок

step2Start = clock();

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

stipendListSingly.push\_back({rand() % 2000,i + 50000,random()});

}

step2End = clock();

//зчитати 20000 елементів під випадковими індексами

step3Start = clock();

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

stipendListSingly.get(rand() % 60000 + 1);

}

step3End = clock();

//видалити 5000 елементів з початку

step4Start = clock();

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

stipendListSingly.pop\_front();

}

step4End = clock();

//видалити 5000 елементів з кінця

step5Start = clock();

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

stipendListSingly.pop\_back();

}

step5End = clock();

End = clock();

cout << "\n\nSINGLY LINKED LIST: \n";

step1Time = (float(step1End - step1Start)) / CLOCKS\_PER\_SEC;

cout << "1 крок -> додати 50000 елементiв в кiнець " << step1Time << endl;

step2Time = (float(step2End - step2Start)) / CLOCKS\_PER\_SEC;

cout << "2 крок -> додати 10000 елементiв в початок " << step2Time << endl;

step3Time = (float(step3End - step3Start)) / CLOCKS\_PER\_SEC;

cout << "3 крок -> зчитати 20000 елементiв пiд випадковими iндексами " << step3Time << endl;

step4Time = (float(step4End - step4Start)) / CLOCKS\_PER\_SEC;

cout << "4 крок -> видалити 5000 елементiв з початку " << step4Time << endl;

step5Time = (float(step5End - step5Start)) / CLOCKS\_PER\_SEC;

cout << "5 крок -> видалити 5000 елементiв з кiнця " << step5Time << endl;

time = (float(End - Start)) / CLOCKS\_PER\_SEC;

cout << "TIME: " << time << endl;

//----------------------------------------------------------------------------------------------

DoublyLinkedList stipendListDoubly;

Start = clock();

//додати 50000 елементів в кінець

step1Start = clock();

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

stipendListDoubly.push\_back({ rand() % 2000,i + 1,random() });

}

step1End = clock();

//додати 10000 елементів в початок

step2Start = clock();

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

stipendListDoubly.push\_back({ rand() % 2000,i + 50000,random() });

}

step2End = clock();

//зчитати 20000 елементів під випадковими індексами

step3Start = clock();

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

stipendListDoubly.get(rand() % 60000 + 1);

}

step3End = clock();

//видалити 5000 елементів з початку

step4Start = clock();

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

stipendListDoubly.pop\_front();

}

step4End = clock();

//видалити 5000 елементів з кінця

step5Start = clock();

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

stipendListDoubly.pop\_back();

}

step5End = clock();

End = clock();

cout << "\n\nDOUBLY LINKED LIST: \n";

step1Time = (float(step1End - step1Start)) / CLOCKS\_PER\_SEC;

cout << "1 крок -> додати 50000 елементiв в кiнець " << step1Time << endl;

step2Time = (float(step2End - step2Start)) / CLOCKS\_PER\_SEC;

cout << "2 крок -> додати 10000 елементiв в початок " << step2Time << endl;

step3Time = (float(step3End - step3Start)) / CLOCKS\_PER\_SEC;

cout << "3 крок -> зчитати 20000 елементiв пiд випадковими iндексами " << step3Time << endl;

step4Time = (float(step4End - step4Start)) / CLOCKS\_PER\_SEC;

cout << "4 крок -> видалити 5000 елементiв з початку " << step4Time << endl;

step5Time = (float(step5End - step5Start)) / CLOCKS\_PER\_SEC;

cout << "5 крок -> видалити 5000 елементiв з кiнця " << step5Time << endl;

time = (float(End - Start)) / CLOCKS\_PER\_SEC;

cout << "TIME: " << time << endl;

return 0;

}

DynamicArray.cpp:

#include <iostream>

#include "dynamArrayFunk.h"

using namespace std;

struct stipend;

int capacity = 1;

int realsize = 0;

const int alpha = 2;

stipend\* innerArray = new stipend[capacity];

bool random() {

return rand() % 2 == 0 ? true : false;

}

void DynamicArray::push\_back(stipend data) {

if (realsize == capacity) {

capacity \*= alpha;

stipend\* newArr = new stipend[capacity];

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

newArr[i] = innerArray[i];

}

delete[] innerArray;

innerArray = newArr;

}

innerArray[realsize] = data;

realsize++;

}

stipend DynamicArray::pop\_back() {

stipend deleted = innerArray[realsize - 1];

realsize--;

return deleted;

}

stipend DynamicArray::get(int n) {

if (n > size() || n <= 0) {

cout << "element outside the array\n";

}

else {

return innerArray[n - 1];

}

}

int DynamicArray::size() {

return realsize;

}

void DynamicArray::print() {

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

cout << innerArray[i].size << " " << innerArray[i].number << " " << innerArray[i].availability << endl;

}

}

void DynamicArray::push\_front(stipend data) {

if (realsize == capacity) {

capacity \*= alpha;

stipend\* newArr = new stipend[capacity];

for (int i = 1; i <= realsize; i++) {

newArr[i] = innerArray[i - 1];

}

newArr[0] = data;

delete[] innerArray;

innerArray = newArr;

}

else {

for (int i = realsize; i > 0; i--) {

innerArray[i] = innerArray[i - 1];

}

innerArray[0] = data;

}

realsize++;

}

stipend DynamicArray::pop\_front() {

stipend deleted = innerArray[0];

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

innerArray[i] = innerArray[i + 1];

}

realsize--;

return deleted;

}

LinkedList.cpp:

#include <iostream>

#include "linkedListFunk.h"

using namespace std;

struct SinglyLinkedList;

struct DoublyLinkedList;

struct Node;

struct Node2;

Node\* head = NULL;

Node\* tail = NULL;

Node2\* head2 = NULL;

Node2\* tail2 = NULL;

Node2\* parent = NULL;

int quantity1 = 0;

int quantity2 = 0;

void SinglyLinkedList::push\_back(stipendList data) {

Node\* newNode = new Node();

newNode->data = data;

newNode->next = NULL;

if (head == NULL) {

head = newNode;

tail = newNode;

}

else {

Node\* currentNode = newNode;

tail->next = newNode;

tail = newNode;

}

quantity1++;

}

stipendList SinglyLinkedList::pop\_back() {

if (size() != 0) {

Node\* currentNode = head;

while (currentNode->next != tail) {

currentNode = currentNode->next;

}

stipendList deleted = tail->data;

Node\* last = tail;

tail = currentNode;

tail->next = NULL;

delete last;

quantity1--;

return deleted;

}

else {

cout << "array is empty\n";

}

}

stipendList SinglyLinkedList::get(int n) {

n -= 1;

if (n >= size() || n < 0) {

cout << "element outside the array\n";

}

else {

Node\* currentNode = head;

int id = 0;

while (currentNode->next != NULL && id != n) {

id++;

currentNode = currentNode->next;

}

return currentNode->data;

}

}

int SinglyLinkedList::size() {

return quantity1;

}

void SinglyLinkedList::print() {

Node\* currentNode = head;

while (currentNode != NULL) {

cout << currentNode->data.size << " " << currentNode->data.number << " " << currentNode->data.availability << endl;

currentNode = currentNode->next;

}

}

void SinglyLinkedList::push\_front(stipendList data) {

Node\* newNode = new Node();

newNode->data = data;

newNode->next = NULL;

if (head == NULL) {

head = newNode;

tail = newNode;

}

else {

Node\* currentNode = newNode;

currentNode->next = head;

head = newNode;

}

quantity1++;

}

stipendList SinglyLinkedList::pop\_front() {

if (size() != 0) {

stipendList deleted = head->data;

Node\* currentNode = head;

head = head->next;

delete currentNode;

quantity1--;

return deleted;

}

else {

cout << "array is empty\n";

}

}

void SinglyLinkedList::push(stipendList data, int n) {

n -= 1;

Node\* newNode = new Node();

newNode->data = data;

newNode->next = NULL;

if (n >= size() || n < 0) {

cout << "outside the array\n";

}

else if (n == 0) {

Node\* currentNode = head;

head = newNode;

newNode->next = currentNode;

}

else {

Node\* currentNode = head;

int idTmp = 1;

while (idTmp != n) {

currentNode = currentNode->next;

idTmp++;

}

Node\* next = currentNode->next;

currentNode->next = newNode;

newNode->next = next;

}

}

//------------------------------------------------------

void DoublyLinkedList::push\_back(stipendList data) {

Node2\* newNode = new Node2();

newNode->data = data;

newNode->next = NULL;

newNode->parent = tail2;

if (head2 == NULL) {

head2 = newNode;

tail2 = newNode;

}

else {

Node2\* currentNode = newNode;

tail2->next = newNode;

tail2 = newNode;

}

quantity2++;

}

stipendList DoublyLinkedList::pop\_back() {

if (size() != 0) {

stipendList last = tail2->data;

Node2\* deleted = tail2;

tail2 = tail2->parent;

tail2->next = NULL;

delete deleted;

quantity2--;

return last;

}

else {

cout << "array is empty\n";

}

}

stipendList DoublyLinkedList::get(int n) {

n -= 1;

if (n >= size() || n < 0) {

cout << "element outside the array\n";

}

else if(size() / 2 >= n){

Node2\* currentNode = head2;

int id = 0;

while (currentNode->next != NULL && id != n) {

id++;

currentNode = currentNode->next;

}

return currentNode->data;

}

else {

Node2\* currentNode = tail2;

int id = size();

while (currentNode->parent != NULL && id != n) {

id--;

currentNode = currentNode->parent;

}

return currentNode->data;

}

}

int DoublyLinkedList::size() {

return quantity2;

}

void DoublyLinkedList::print() {

Node2\* currentNode = head2;

while (currentNode != NULL) {

cout << currentNode->data.size << " " << currentNode->data.number << " " << currentNode->data.availability << endl;

currentNode = currentNode->next;

}

}

void DoublyLinkedList::push\_front(stipendList data) {

Node2\* newNode = new Node2();

newNode->data = data;

newNode->next = NULL;

newNode->parent = NULL;

if (head2 == NULL) {

head2 = newNode;

tail2 = newNode;

}

else {

Node2\* currentNode = newNode;

currentNode->next = head2;

head2->parent = newNode;

head2 = newNode;

}

quantity2++;

}

stipendList DoublyLinkedList::pop\_front() {

if (size() != 0) {

stipendList deleted = head2->data;

Node2\* currentNode = head2;

head2 = head2->next;

delete currentNode;

quantity2--;

return deleted;

}

else {

cout << "array is empty\n";

}

}

linkedListFunk.h:

#include <iostream>

struct stipendList {

int size;

int number;

bool availability;

stipendList(int size\_tmp, int number\_tmp, bool availability\_tmp) {

size = size\_tmp;

number = number\_tmp;

availability = availability\_tmp;

}

};

struct Node { // 3.1

stipendList data = { 0, 0, 0 };

Node\* next = NULL;

};

struct Node2 { // 3.1

stipendList data = { 0, 0, 0 };

Node2\* next = NULL;

Node2\* parent = NULL;

};

struct SinglyLinkedList { // 3.2

Node\* head = NULL;

Node\* tail = NULL;

int quantity1 = 0;

void push\_front(stipendList data); // 3.2.1

void push\_back(stipendList data); // 3.2.2

stipendList get(int n); // 3.2.3

stipendList pop\_front(); // 3.2.4

stipendList pop\_back(); // 3.2.5

int size(); // 3.2.6

void print(); // 3.2.7

void push(stipendList data, int n);

};

struct DoublyLinkedList {

Node2\* head2 = NULL;

Node2\* tail2 = NULL;

Node2\* parent = NULL;

int quantity2 = 0;

void push\_front(stipendList data); // +

void push\_back(stipendList data); // +

stipendList get(int n); // +

stipendList pop\_front(); // +

stipendList pop\_back(); // +

int size(); // +

void print(); // +

};

dynamArrayFunk.h:

#pragma once

bool random();

struct stipend {

int size;

int number;

bool availability;

};

struct DynamicArray {

int capacity = 1;

int realsize = 0;

const int alpha = 2;

stipend\* innerArray = new stipend[capacity];

void push\_back(stipend data); // 2.1.1

stipend pop\_back(); // 2.1.2

stipend get(int n); // 2.1.3

int size(); // 2.1.4

void print(); // 2.1.5

void push\_front(stipend data); // 2.2.1

stipend pop\_front(); // 2.2.2

};

Висновки:

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