Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное

образовательное учреждение высшего образования

«Уфимский университет науки и технологий»

Факультет информатики и робототехники

Кафедра вычислительной математики и кибернетики

Лабораторная работа №3

“Контейнер”

По дисциплине “Объектно-ориентированное программирование”

Выполнил:

Студент группы ПРО-232Б

Степанов К.А.

Проверил:

доцент кафедры ВМиК

Котельников В.А.

Уфа-2023

**Цель работы:**

Определение и реализация собственного контейнера, его переменных и методов, его тестирование.

**Задание:**

Определение и реализация класса-контейнера разнообразных объектов (принадлежащих различным классам, имеющим общего предка), и написание программы, иллюстрирующей использование контейнера. Контейнер должен быть максимально универсальным, это не должен быть контейнер фигур, животных или любых других конкретных объектов; вы должны иметь возможность использовать его для хранения любых объектов, которые у вас в будущем могут появиться. Если умеете пользоваться шаблонами (дженериками и т.д.), используйте их; если не умеете – создавайте контейнер указателей на какой-то самый базовый класс, но этот класс должен быть максимально абстрактный.

Предназначение контейнера – хранить объекты, которые в него помещаются. Для создаваемого контейнера вы должны тщательно продумать, как вы будете его использовать для хранения объектов, какие типовые действия вы будете с контейнером производить.

• Функции контейнера объектов

o добавление объектов (в начало, в конец), вставка объектов (в середину)

o изъятие объектов (с удалением самого объекта и без)

o переход по объектам, если это применимо (текущий, предыдущий, последующий, проверка наличия)

o получение очередного объекта из контейнера или объекта по индексу

o поочередное обращение к каждому объекту контейнера; вызов функций, реализуемых всеми классами объектов контейнера

• Функции основной программы: код, который в случайном порядке:

o создаёт объекты

o добавляет в контейнер

o использует (для упрощённого вывода)

o удаляет из контейнера

Контейнер должен представлять собой объект, создаваемый и используемый в основной программе. Контейнер должно вести себя (снаружи) или как массив, или как список (на выбор студента). Контейнер должно быть организовано внутри или как массив, или как список (на выбор студента). Студент должен понимать отличие двух предыдущих предложений. Основная программа должна демонстрировать использование основных функций контейнера и запускать цикл из 100, 1000 и 10000 случайных действий и подсчитывать время работы. Действия должны случайным образом выбираться из списка: создание и вставка в случайное место контейнера нового объекта, удаление и уничтожение случайного объекта, запуск любого метода у случайного объекта из контейнера.

Контейнер должен позволять добавлять, удалять объекты в случайной последовательности, корректно обрабатывать подсчет текущего количества объектов в контейнере (с учетом возможных «пустых мест» после удаления каких-то объектов) и динамически увеличивать свой размер, если добавляется больше объектов, чем было предусмотрено изначально.

**Ход работы:**

Создать Header файл с описанием шаблона класса контейнера, описать в нем все методы. В main файле создается программа для тестирования созданного контейнера на время выполнения разного количества операций с ним (рис. 1).

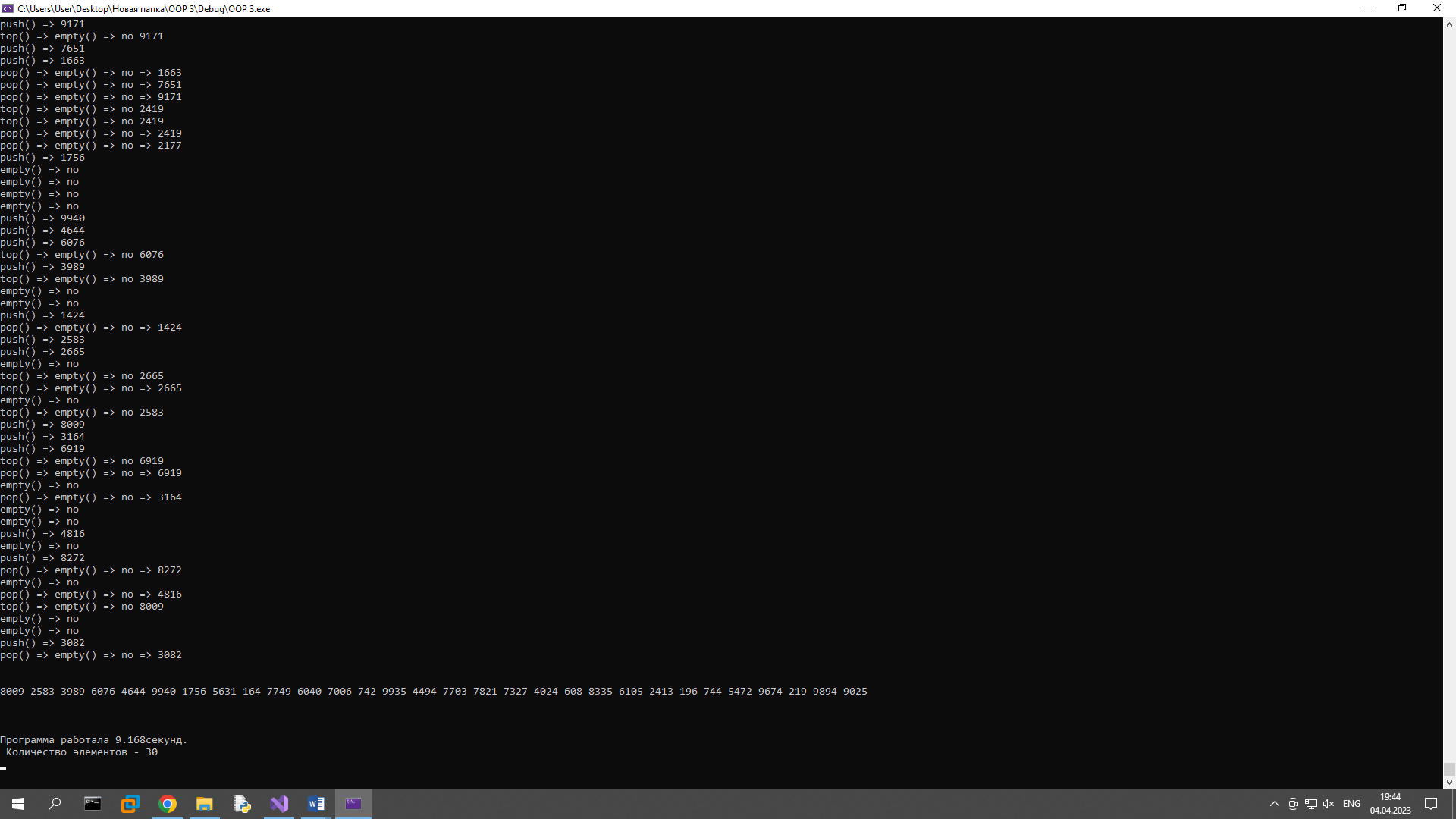


Рис 1 – результат временного тестирования

**Вывод:**

В результате выполнения работы были освоены навыки создания собственных контейнеров, их тестирования, а также работа с Header файлами и шаблонами.

Ссылка на GitHub: <https://github.com/stkirill15/OOP-3>.

**Приложение**

**Листинг программы**

**1. main.cpp**

#include <iostream>

#include <time.h>

#include <conio.h>

using namespace std;

template <typename T> class Stack

{

public:

Stack() : root(nullptr), size(0) {} // По умолчанию стек пуст

Stack <T>(Stack const& a) : root(nullptr), size(0) // Конструктор копирования

{

Node\* copy = nullptr;

for (Node\* n = a.root; n != nullptr; n = n->next) {

Node\* item = new Node();

item->data = n->data;

if (!root)

root = item;

else

copy->next = item;

copy = item;

++size;

}

}

~Stack() // Деструктор

{

while (!empty()) {

pop();

}

}

Stack <T>& operator =(Stack const& a) // Оператор присваивания

{

while (!empty())

pop();

size = 0;

Node\* copy = nullptr;

for (Node\* n = a.root; n != nullptr; n = n->next) {

Node\* item = new Node();

item->data = n->data;

if (!root)

root = item;

else

copy->next = item;

copy = item;

++size;

}

return \*this;

}

void push(const T& x) // Добавление в стек

{

cout << "push() => " << x << endl;

Node\* newnode = new Node;

newnode->data = x;

newnode->next = root;

root = newnode;

size++;

}

bool empty() const // Проверка на пустоту

{

printf("empty() => ");

if (root == nullptr)

printf("yes ");

else

printf("no ");

return root == nullptr;

}

const T& top() const // Вывод вершины стека

{

cout << "top() => ";

if (empty())

{

return NULL;

}

cout << root->data << endl;

return root->data;

}

void pop() { // Удаление вершины стека

cout << "pop() => ";

if (empty())

{

printf("\n");

return;

}

cout << "=> " << root->data << endl;

Node\* delnode = root;

root = delnode->next;

size--;

delete delnode;

}

int get\_size() { // Вывести размер стека

return size;

}

void clear() { // Очистка стека

while (!empty()) {

pop();

}

}

void show() const { // Вывод элементов стека

for (Node\* n = root; n != nullptr; n = n->next)

cout << n->data << ' ';

printf("\n");

}

private:

struct Node // Узел стека

{

T data; // Поле с данными

Node\* next; // Указатель на следующий узел

};

int size;

Node\* root; // Верхний элемент стека

};

int main()

{

setlocale(LC\_ALL, "Russian");

while (1) {

system("cls");

Stack <int> s;

clock\_t start = clock();

srand(std::time(NULL));

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

//srand(std::time(NULL));

int choice = rand() % 4;

if (choice == 0) {

int a = rand() % 10000;

s.push(a);

}

else if (choice == 1) {

s.pop();

}

else if (choice == 2) {

s.empty();

printf("\n");

}

else if (choice == 3) {

s.top();

}

}

clock\_t end = clock();

printf("\n\n");

s.show();

printf("\n\n");

long double time = (long double)(end - start) / CLOCKS\_PER\_SEC;

cout << "\nПрограмма работала" << time << "секунд.\n Количество элементов - " << s.get\_size() << endl;

\_getch();

}

return 0;

}