Чернівецький національний університет імені Юрія Федьковича Навчально-науковий інститут фізико-технічних та комп’ютерних наук Кафедра математичних проблем управління і кібернетики

Звіт

про виконання лабораторної роботи №7

«Основи Об’єктно – орієнтоване програмування мовою С++. Класи. Протокол класу. Конструктори та деструктори.з дисципліни «Об’єктно-орієнтоване програмування»

Виконав студент групи 141 А (1)

Книш Володимир Володимирович

Перевірив: канд.фіз.-мат. наук,

доцент Лазорик В.В.

Оцінка:

Дата захисту:

Чернівці 2024

**Варіант №7**

**Адреса репозиторію де виконувалось завдання**:

<https://github.com/Volodymyr16e/lab7.git>

Тема: Параметризовані типи. Шаблони функції. Шаблони класів. Ітератори.

**Постановка:**

**Завдання 1.**

**Шаблони функцій.**

Задача 1.7. Описати функції-шаблони для знаходження мінімального та максимального елемента пари чисел. Написати специфікацію функції-шаблон для типу **char\*.**

**Код програми(1)**

#include <iostream>

#include <cstring> // Потрібно для функції strcmp

// Шаблонна функція для знаходження мінімального елемента

template <typename T>

T minElement(T a, T b) {

return (a < b) ? a : b;

}

// Шаблонна функція для знаходження максимального елемента

template <typename T>

T maxElement(T a, T b) {

return (a > b) ? a : b;

}

// Специфікація шаблонної функції для типу char\*

template <>

const char\* minElement<const char\*>(const char\* a, const char\* b) {

// Використовуємо strcmp для порівняння рядків

return (std::strcmp(a, b) < 0) ? a : b;

}

// Специфікація шаблонної функції для типу char\*

template <>

const char\* maxElement<const char\*>(const char\* a, const char\* b) {

// Використовуємо strcmp для порівняння рядків

return (std::strcmp(a, b) > 0) ? a : b;

}

int main() {

// Приклад використання шаблонних функцій з різними типами даних

std::cout << "Min of 3 and 7: " << minElement(3, 7) << std::endl;

std::cout << "Max of 3.5 and 2.1: " << maxElement(3.5, 2.1) << std::endl;

// Приклад використання шаблонних функцій з char\*

const char\* str1 = "apple";

const char\* str2 = "orange";

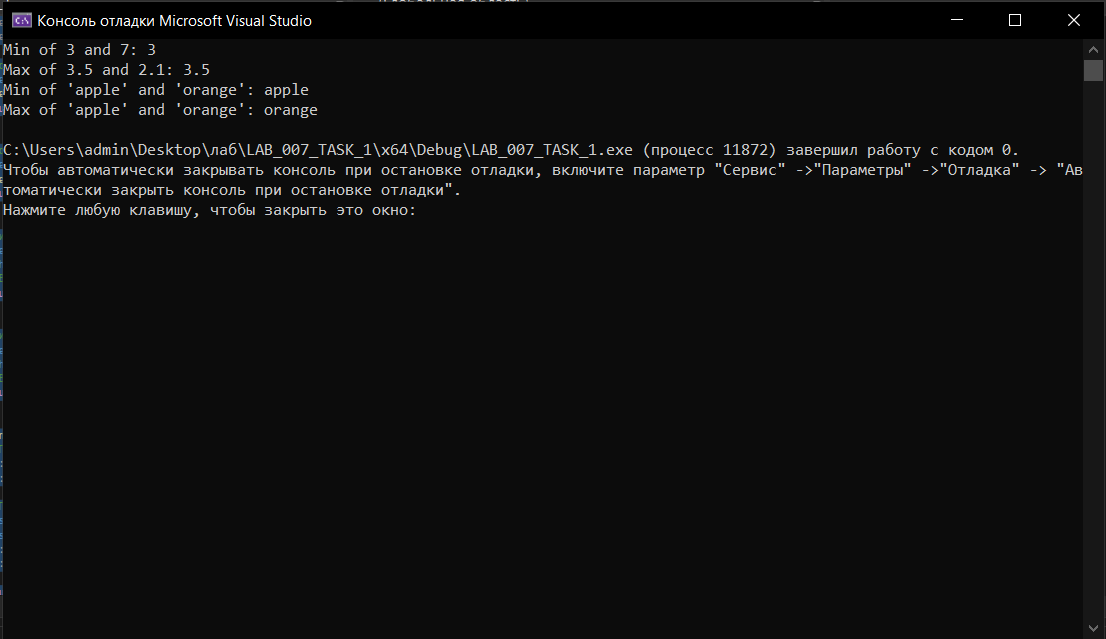
std::cout << "Min of 'apple' and 'orange': " << minElement(str1, str2) << std::endl;

std::cout << "Max of 'apple' and 'orange': " << maxElement(str1, str2) << std::endl;

return 0;

}

**Скріншоти:**

****

**Завдання 2.**

**Шаблони функцій 2**.

Задача 2.3. Написати функцію-шаблон функцію впорядкування методом «Вставки». Написати специфікацію функції-шаблон для типу **char\*.**

**Код програми(2):**

#include <iostream>

// Шаблонна функція для сортування методом "Вставки"

template <typename T>

void insertionSort(T arr[], int n) {

// Перебираємо всі елементи масиву, починаючи з другого

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

T key = arr[i]; // Зберігаємо поточний елемент

int j = i - 1;

// Переміщуємо всі більші елементи вправо від key

while (j >= 0 && arr[j] > key) {

arr[j + 1] = arr[j];

j--;

}

arr[j + 1] = key; // Вставляємо key у відповідне місце

}

}

// Виведення масиву на екран

template <typename T>

void printArray(T arr[], int n) {

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

std::cout << arr[i] << " "; // Виводимо кожен елемент масиву

}

std::cout << std::endl;

}

int main() {

// Приклад використання з різними типами даних

// Масив цілих чисел

int intArr[] = { 12, 11, 13, 5, 6 };

int intSize = sizeof(intArr) / sizeof(intArr[0]); // Обчислюємо розмір масиву

std::cout << "Original integer array: ";

printArray(intArr, intSize); // Виводимо початковий масив

insertionSort(intArr, intSize); // Сортуємо масив

std::cout << "Sorted integer array: ";

printArray(intArr, intSize); // Виводимо відсортований масив

// Масив дійсних чисел

double doubleArr[] = { 3.1, 2.2, 1.3, 5.4, 4.5 };

int doubleSize = sizeof(doubleArr) / sizeof(doubleArr[0]); // Обчислюємо розмір масиву

std::cout << "Original double array: ";

printArray(doubleArr, doubleSize); // Виводимо початковий масив

insertionSort(doubleArr, doubleSize); // Сортуємо масив

std::cout << "Sorted double array: ";

printArray(doubleArr, doubleSize); // Виводимо відсортований масив

return 0;

}

**Скріншоти:**

****

**Завдання 3.**

**Шаблони класів**. **(Без використання STL)**

Задача 3.7. Створити параметризованний клас однозв'язного списку.

**Код програми(3):**

#include <iostream>

#include <locale> // Для setlocale

// Клас для вузла однозв'язного списку

template <typename T>

class Node {

public:

T data; // Дані, які зберігаються у вузлі

Node\* next; // Вказівник на наступний вузол

// Конструктор для ініціалізації даних і вказівника на NULL

Node(T value) : data(value), next(nullptr) {}

};

// Клас для однозв'язного списку

template <typename T>

class SinglyLinkedList {

private:

Node<T>\* head; // Вказівник на перший елемент списку

int size; // Розмір списку

public:

// Конструктор за замовчуванням, який ініціалізує пустий список

SinglyLinkedList() : head(nullptr), size(0) {}

// Деструктор для звільнення пам'яті, якщо потрібно

~SinglyLinkedList() {

clear(); // Викликаємо метод очищення списку перед знищенням об'єкта

}

// Метод, що додає елемент в кінець списку

void pushBack(T value) {

Node<T>\* newNode = new Node<T>(value); // Створюємо новий вузол

if (head == nullptr) {

head = newNode; // Якщо список порожній, новий вузол стає першим

}

else {

Node<T>\* current = head;

while (current->next != nullptr) {

current = current->next; // Шукаємо останній елемент списку

}

current->next = newNode; // Додаємо новий вузол в кінець

}

size++; // Збільшуємо розмір списку

}

// Метод, що видаляє всі елементи списку

void clear() {

Node<T>\* current = head;

while (current != nullptr) {

Node<T>\* next = current->next;

delete current; // Видаляємо поточний вузол

current = next; // Переходимо до наступного вузла

}

head = nullptr; // Встановлюємо вказівник на початок списку в NULL

size = 0; // Скидаємо розмір списку до 0

}

// Метод для виведення елементів списку на екран

void display() {

Node<T>\* current = head;

while (current != nullptr) {

std::cout << current->data << " ";

current = current->next; // Переходимо до наступного вузла

}

std::cout << std::endl;

}

// Метод для отримання розміру списку

int getSize() const {

return size;

}

};

int main() {

// Встановлення української локалізації

setlocale(LC\_CTYPE, "Ukrainian");

// Приклад використання класу однозв'язного списку з різними типами даних

// Створення та заповнення списку цілими числами

SinglyLinkedList<int> list1;

list1.pushBack(1);

list1.pushBack(2);

list1.pushBack(3);

std::cout << "Список цілих чисел: ";

list1.display(); // Вивід: 1 2 3

std::cout << "Розмір списку: " << list1.getSize() << std::endl; // Вивід: 3

// Створення та заповнення списку рядками

SinglyLinkedList<std::string> list2;

list2.pushBack("яблуко");

list2.pushBack("банан");

list2.pushBack("вишня");

std::cout << "Список рядків: ";

list2.display(); // Вивід: яблуко банан вишня

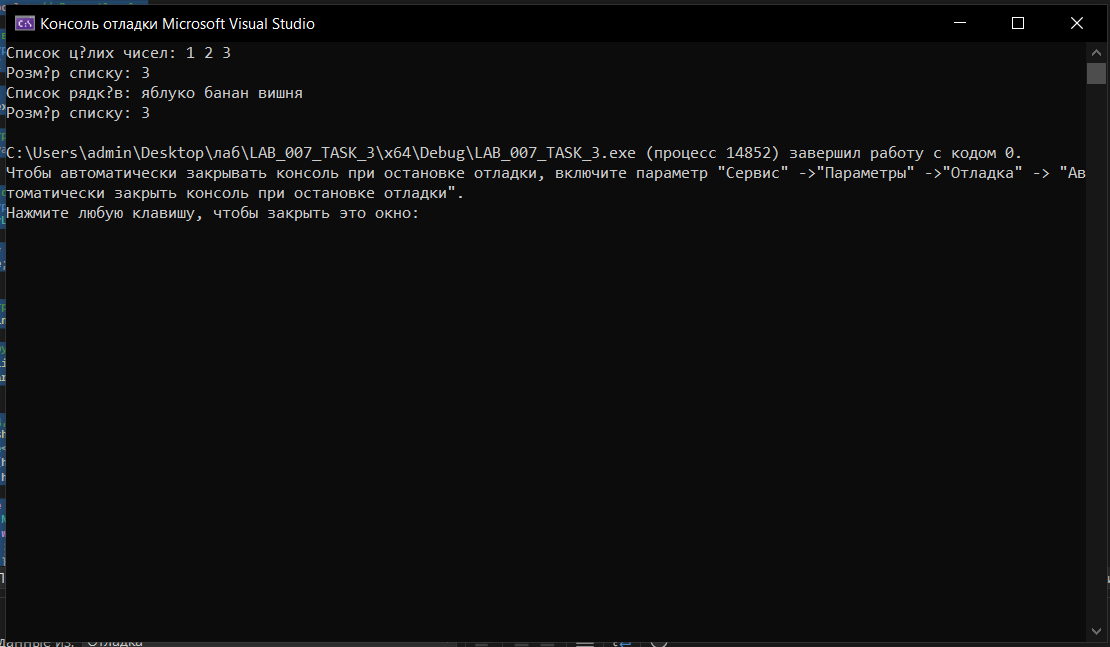
std::cout << "Розмір списку: " << list2.getSize() << std::endl; // Вивід: 3

// Звільнення пам'яті, коли об'єкти list1 та list2 виходять з області видимості

return 0;

}

**Скріншоти:**

****

**Висновок:**

У цьому проекті ми розглянули основні аспекти роботи з шаблонами класів і функцій у мові програмування C++. Використовуючи шаблони, ми змогли створити загальні класи для однозв'язного списку і його ітератора, які працюють з будь-яким типом даних. Шаблони дозволяють зберегти код універсальним і повторно використовуваним, що значно полегшує розробку і підтримку програм, особливо у випадках, коли потрібно працювати з різними типами даних без втрати ефективності і читабельності коду.