**Лабораторна робота N8.**

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

**Методичні вказівки**

**Завдання 1**

Задача 1.6. Описати функції-шаблони для знаходження мінімального та максимального елемента масиву. Написати специфікацію функції-шаблон для типу char\*.

**Код:**

**#include <iostream>**

**#include <cstring> // для strcmp**

**using namespace std;**

**// === Загальні шаблони ===**

**template <typename T>**

**T findMin(T arr[], int size) {**

**T minVal = arr[0];**

**for (int i = 1; i < size; ++i)**

**if (arr[i] < minVal)**

**minVal = arr[i];**

**return minVal;**

**}**

**template <typename T>**

**T findMax(T arr[], int size) {**

**T maxVal = arr[0];**

**for (int i = 1; i < size; ++i)**

**if (arr[i] > maxVal)**

**maxVal = arr[i];**

**return maxVal;**

**}**

**// === Спеціалізація для char\* ===**

**template <>**

**char\* findMin<char\*>(char\* arr[], int size) {**

**char\* minVal = arr[0];**

**for (int i = 1; i < size; ++i)**

**if (strcmp(arr[i], minVal) < 0)**

**minVal = arr[i];**

**return minVal;**

**}**

**template <>**

**char\* findMax<char\*>(char\* arr[], int size) {**

**char\* maxVal = arr[0];**

**for (int i = 1; i < size; ++i)**

**if (strcmp(arr[i], maxVal) > 0)**

**maxVal = arr[i];**

**return maxVal;**

**}**

**// === main ===**

**int main() {**

**int a[] = {4, 2, 8, 1};**

**int n = 4;**

**cout << "Min (int): " << findMin(a, n) << endl;**

**cout << "Max (int): " << findMax(a, n) << endl;**

**char\* strs[] = {"banana", "apple", "cherry"};**

**int m = 3;**

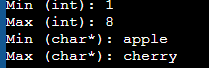
**cout << "Min (char\*): " << findMin(strs, m) << endl;**

**cout << "Max (char\*): " << findMax(strs, m) << endl;**

**return 0;**

**}**

**Скрін:**

****

**Завдання 2**

Задача 2.6. Написати функцію-шаблон функцію об’єднує два масиви в один

впорядкований за зростанням. Написати специфікацію функції-шаблон

для типу char\*.

**Код:**

**#include <iostream>**

**#include <cstring> // для strcmp**

**#include <algorithm> // для sort**

**using namespace std;**

**// === Загальний шаблон функції об’єднання та сортування ===**

**template <typename T>**

**T\* mergeAndSort(T\* arr1, int size1, T\* arr2, int size2) {**

**int totalSize = size1 + size2;**

**T\* result = new T[totalSize];**

**for (int i = 0; i < size1; ++i) result[i] = arr1[i];**

**for (int i = 0; i < size2; ++i) result[size1 + i] = arr2[i];**

**sort(result, result + totalSize);**

**return result;**

**}**

**// === Спеціалізація для char\* ===**

**template <>**

**char\*\* mergeAndSort<char\*>(char\*\* arr1, int size1, char\*\* arr2, int size2) {**

**int totalSize = size1 + size2;**

**char\*\* result = new char\*[totalSize];**

**for (int i = 0; i < size1; ++i) result[i] = arr1[i];**

**for (int i = 0; i < size2; ++i) result[size1 + i] = arr2[i];**

**sort(result, result + totalSize, [](const char\* a, const char\* b) {**

**return strcmp(a, b) < 0;**

**});**

**return result;**

**}**

**// === Функція виводу масиву будь-якого типу ===**

**template <typename T>**

**void printArray(T\* arr, int size) {**

**for (int i = 0; i < size; ++i)**

**cout << arr[i] << " ";**

**cout << endl;**

**}**

**// === main ===**

**int main() {**

**// 🔸 Тест із int**

**int a[] = {5, 1, 3};**

**int b[] = {4, 2};**

**int\* resInt = mergeAndSort(a, 3, b, 2);**

**cout << "Об’єднаний масив int: ";**

**printArray(resInt, 5);**

**delete[] resInt;**

**// 🔸 Тест із char\***

**char\* arr1[] = {"banana", "apple"};**

**char\* arr2[] = {"cherry", "date"};**

**char\*\* resChar = mergeAndSort(arr1, 2, arr2, 2);**

**cout << "Об’єднаний масив char\*: ";**

**printArray(resChar, 4);**

**delete[] resChar;**

**return 0;**

**}**

**Скрін:**

****

**Завдання 3**

Задача 3.6. Створити параметризованний клас бінарного дерева. З методами

- додати елемент у дерево, проходження по дереву в спадному й у

висхідному порядку. Здійснити пошук по дереву.

**Код:**

**#include <iostream>**

**using namespace std;**

**// === Клас вузла дерева ===**

**template <typename T>**

**class TreeNode {**

**public:**

**T data;**

**TreeNode\* left;**

**TreeNode\* right;**

**TreeNode(T val) : data(val), left(nullptr), right(nullptr) {}**

**};**

**// === Параметризований клас бінарного дерева ===**

**template <typename T>**

**class BinaryTree {**

**private:**

**TreeNode<T>\* root;**

**// Допоміжні рекурсивні методи**

**void insert(TreeNode<T>\*& node, T val) {**

**if (!node) {**

**node = new TreeNode<T>(val);**

**} else if (val < node->data) {**

**insert(node->left, val);**

**} else {**

**insert(node->right, val);**

**}**

**}**

**void inOrder(TreeNode<T>\* node) {**

**if (!node) return;**

**inOrder(node->left);**

**cout << node->data << " ";**

**inOrder(node->right);**

**}**

**void reverseOrder(TreeNode<T>\* node) {**

**if (!node) return;**

**reverseOrder(node->right);**

**cout << node->data << " ";**

**reverseOrder(node->left);**

**}**

**bool search(TreeNode<T>\* node, T val) {**

**if (!node) return false;**

**if (val == node->data) return true;**

**return val < node->data ? search(node->left, val) : search(node->right, val);**

**}**

**void destroy(TreeNode<T>\* node) {**

**if (!node) return;**

**destroy(node->left);**

**destroy(node->right);**

**delete node;**

**}**

**public:**

**BinaryTree() : root(nullptr) {}**

**~BinaryTree() {**

**destroy(root);**

**}**

**void add(T val) {**

**insert(root, val);**

**}**

**void printInOrder() {**

**cout << "In-order (висхідний): ";**

**inOrder(root);**

**cout << endl;**

**}**

**void printReverseOrder() {**

**cout << "Reverse-order (спадний): ";**

**reverseOrder(root);**

**cout << endl;**

**}**

**bool find(T val) {**

**return search(root, val);**

**}**

**};**

**// === main ===**

**int main() {**

**BinaryTree<int> tree;**

**tree.add(10);**

**tree.add(5);**

**tree.add(15);**

**tree.add(3);**

**tree.add(8);**

**tree.printInOrder(); // 3 5 8 10 15**

**tree.printReverseOrder(); // 15 10 8 5 3**

**int value = 8;**

**if (tree.find(value))**

**cout << "Значення " << value << " знайдено у дереві." << endl;**

**else**

**cout << "Значення " << value << " не знайдено у дереві." << endl;**

**return 0;**

**}**

**Скрін:**

****

**Завдання 4**

Задача 4.5. Описати шаблон класу List, який визначає однозв’язний список

для елементів будь-якого типу. Побудувати клас ітератор, що дозволяє

проходити список.

**Код:**

**#include <iostream>**

**using namespace std;**

**// === Шаблон класу List ===**

**template <typename T>**

**class List {**

**private:**

**// Внутрішній вузол списку**

**struct Node {**

**T data;**

**Node\* next;**

**Node(T val) : data(val), next(nullptr) {}**

**};**

**Node\* head;**

**Node\* tail;**

**public:**

**List() : head(nullptr), tail(nullptr) {}**

**~List() {**

**Node\* current = head;**

**while (current) {**

**Node\* tmp = current;**

**current = current->next;**

**delete tmp;**

**}**

**}**

**// Додати елемент у кінець списку**

**void push\_back(T val) {**

**Node\* newNode = new Node(val);**

**if (!head)**

**head = tail = newNode;**

**else {**

**tail->next = newNode;**

**tail = newNode;**

**}**

**}**

**// === Внутрішній клас ітератора ===**

**class Iterator {**

**private:**

**Node\* current;**

**public:**

**Iterator(Node\* ptr) : current(ptr) {}**

**T& operator\*() { return current->data; }**

**Iterator& operator++() {**

**if (current) current = current->next;**

**return \*this;**

**}**

**bool operator!=(const Iterator& other) const {**

**return current != other.current;**

**}**

**bool operator==(const Iterator& other) const {**

**return current == other.current;**

**}**

**};**

**// Повертає ітератори**

**Iterator begin() { return Iterator(head); }**

**Iterator end() { return Iterator(nullptr); }**

**};**

**// === main ===**

**int main() {**

**List<int> myList;**

**myList.push\_back(10);**

**myList.push\_back(20);**

**myList.push\_back(30);**

**cout << "Елементи списку: ";**

**for (auto it = myList.begin(); it != myList.end(); ++it) {**

**cout << \*it << " ";**

**}**

**cout << endl;**

**return 0;**

**}**

**Скрін:**

****