Міністерство освіти і науки України

Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії

Кафедра програмної інженерії

ЗВІТ

до практичної роботи №2.1

з навчальної дисципліни «Об'єктно-орієнтоване програмування»

**Тема**: «Контейнери бібліотеки STL»

Підготував:

студент групи СПзс-21

Карпюк Олександра Андріївна

Варіант №1

Тернопіль 2024

**Мета**: Метою лабораторної роботи є розробка застосунку на основі бібліотеки STL (Standard Template Library) для роботи з контейнерами та алгоритмами. Студентам слід ознайомитися з основними класами бібліотеки, реалізувати функціонал для обробки контейнерів, використовуючи ітератори та стандартні алгоритми.

**Завдання**

Розробіть програму для ведення списку особистих цілей та досягнень. Використовуйте клас std::vector для зберігання цілей. Кожна ціль повинна містити назву, опис та термін досягнення. Реалізуйте операції додавання нової цілі, видалення цілі, виведення списку цілей та позначення цілей, які вже досягнуті.

Ієрархія класів:

* Створити базовий клас з абстрактними методами та властивостями.
* Розробити принаймні два похідні класи, які успадковують властивості та методи від базового класу.

Операції з контейнером:

* Розробити методи для додавання, видалення та пошуку об'єктів в контейнері.
* Забезпечити можливість сортування об'єктів за різними критеріями.

Запис та зчитування з файлу:

* Реалізувати методи для запису та зчитування об'єктів з файлу.

**Хід роботи**

Лістинг:

#include <iostream> // Library for input and output

#include <vector> // Library for storing lists of goals

#include <string> // Library for working with strings

#include <limits> // Library for input validation

// Class Goal for storing information about a single goal

class Goal {

public:

std::string title; // Goal title

std::string description; // Goal description

std::string deadline; // Goal deadline

// Constructor for initializing a new goal

Goal(const std::string& t, const std::string& d, const std::string& dl)

: title(t), description(d), deadline(dl) {}

// Method to display information about the goal

void display() const {

std::cout << "Title: " << title << "\nDescription: " << description << "\nDeadline: " << deadline << "\n";

}

};

// Class GoalManager to manage multiple goals

class GoalManager {

private:

std::vector<Goal> goals; // List of all active (not achieved) goals

std::vector<Goal> achievedGoals; // List of all achieved goals

public:

// Method to add a new goal to the list

void addGoal(const std::string& title, const std::string& description, const std::string& deadline) {

if (title.empty() || description.empty() || deadline.empty()) {

std::cout << "All fields must be filled. Goal not added.\n";

}

else {

goals.emplace\_back(title, description, deadline);

std::cout << "Goal added.\n";

}

}

// Method to delete a goal by its index

void deleteGoal(int index) {

if (index >= 0 && index < goals.size()) {

goals.erase(goals.begin() + index);

std::cout << "Goal deleted.\n";

}

else {

std::cout << "Invalid index.\n";

}

}

// Method to mark a goal as achieved

void markGoalAsAchieved(int index) {

if (index >= 0 && index < goals.size()) {

achievedGoals.push\_back(goals[index]);

goals.erase(goals.begin() + index);

std::cout << "Goal marked as achieved.\n";

}

else {

std::cout << "Invalid index.\n";

}

}

// Method to display all active goals

void displayGoals() const {

if (goals.empty()) {

std::cout << "The list of goals is empty.\n";

return;

}

for (size\_t i = 0; i < goals.size(); ++i) {

std::cout << "\nGoal #" << i + 1 << ":\n";

goals[i].display();

std::cout << "------------------------\n";

}

}

// Method to display all achieved goals

void displayAchievedGoals() const {

if (achievedGoals.empty()) {

std::cout << "The list of achieved goals is empty.\n";

return;

}

std::cout << "Achieved Goals:\n";

for (size\_t i = 0; i < achievedGoals.size(); ++i) {

std::cout << "\nAchieved Goal #" << i + 1 << ":\n";

achievedGoals[i].display();

std::cout << "------------------------\n";

}

}

// Method to get the size of the goals vector

int getGoalsSize() const {

return goals.size();

}

};

// Function to display and validate the user's menu choice

int getMenuChoice() {

int choice;

std::cout << "\n1. Add a new goal\n2. Delete a goal\n3. Show all goals\n4. Mark a goal as achieved\n5. Show achieved goals\n6. Exit\n";

std::cout << "Choose an option: ";

while (!(std::cin >> choice)) {

std::cout << "Invalid input. Please enter a number: ";

std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

}

std::cin.ignore(); // Clear newline character from the input buffer

return choice;

}

// Main function providing the user with a menu for goal management

int main() {

GoalManager manager;

int choice;

do {

choice = getMenuChoice();

if (choice == 1) { // Adding a new goal

std::string title, description, deadline;

std::cout << "Enter goal title: ";

std::getline(std::cin, title);

std::cout << "Enter goal description: ";

std::getline(std::cin, description);

std::cout << "Enter goal deadline: ";

std::getline(std::cin, deadline);

manager.addGoal(title, description, deadline);

}

else if (choice == 2) { // Deleting a goal

manager.displayGoals();

int index;

std::cout << "Enter the goal number to delete: ";

while (!(std::cin >> index) || index < 1 || index > manager.getGoalsSize()) {

std::cout << "Invalid input. Please enter a valid goal number: ";

std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

}

manager.deleteGoal(index - 1);

}

else if (choice == 3) { // Displaying all active goals

manager.displayGoals();

}

else if (choice == 4) { // Marking a goal as achieved

manager.displayGoals();

int index;

std::cout << "Enter the goal number to mark as achieved: ";

while (!(std::cin >> index) || index < 1 || index > manager.getGoalsSize()) {

std::cout << "Invalid input. Please enter a valid goal number: ";

std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

}

manager.markGoalAsAchieved(index - 1);

}

else if (choice == 5) { // Displaying all achieved goals

manager.displayAchievedGoals();

}

else if (choice == 6) { // Exiting the program

std::cout << "Exiting the program.\n";

}

else {

std::cout << "Invalid choice.\n";

}

} while (choice != 6); // Завершення циклу do-while, яке дозволяє вихід при choice == 6

return 0;

}

Результат виконання:



