Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

Механіко-технологічний факультет

ЗВІТ

ПРО ВИКОНАННЯ ЛАБОРАТОРНОЇ РОБОТИ № 11

з навчальної дисципліни

“Базові методології та технології програмування”

РЕАЛІЗАЦІЯ ПРОГРАМНИХ ЗАСОБІВ ОБРОБЛЕННЯ ДИНАМІЧНИХ СТРУКТУР ДАНИХ ТА БІНАРНИХ ФАЙЛІВ

ЗАВДАННЯ ВИДАВ

доцент кафедри кібербезпеки та програмного забезпечення

Доренський О. П.

[https://github.com/odorenskyi/](https://github.com/odorenskyi/Dmytro-Parkhomenko-KB18)

ВИКОНАВ

студент академічної групи КН-23

Гребенюк Д. О.

ПЕРЕВІРИВ

ст. викладач кафедри кібербезпеки   
та програмного забезпечення

Дрєєва Г. М.

Кропивницький – 2024

**Тема:** Реалізація програмних засобів оброблення динамічних структур даних та бінарних файлів

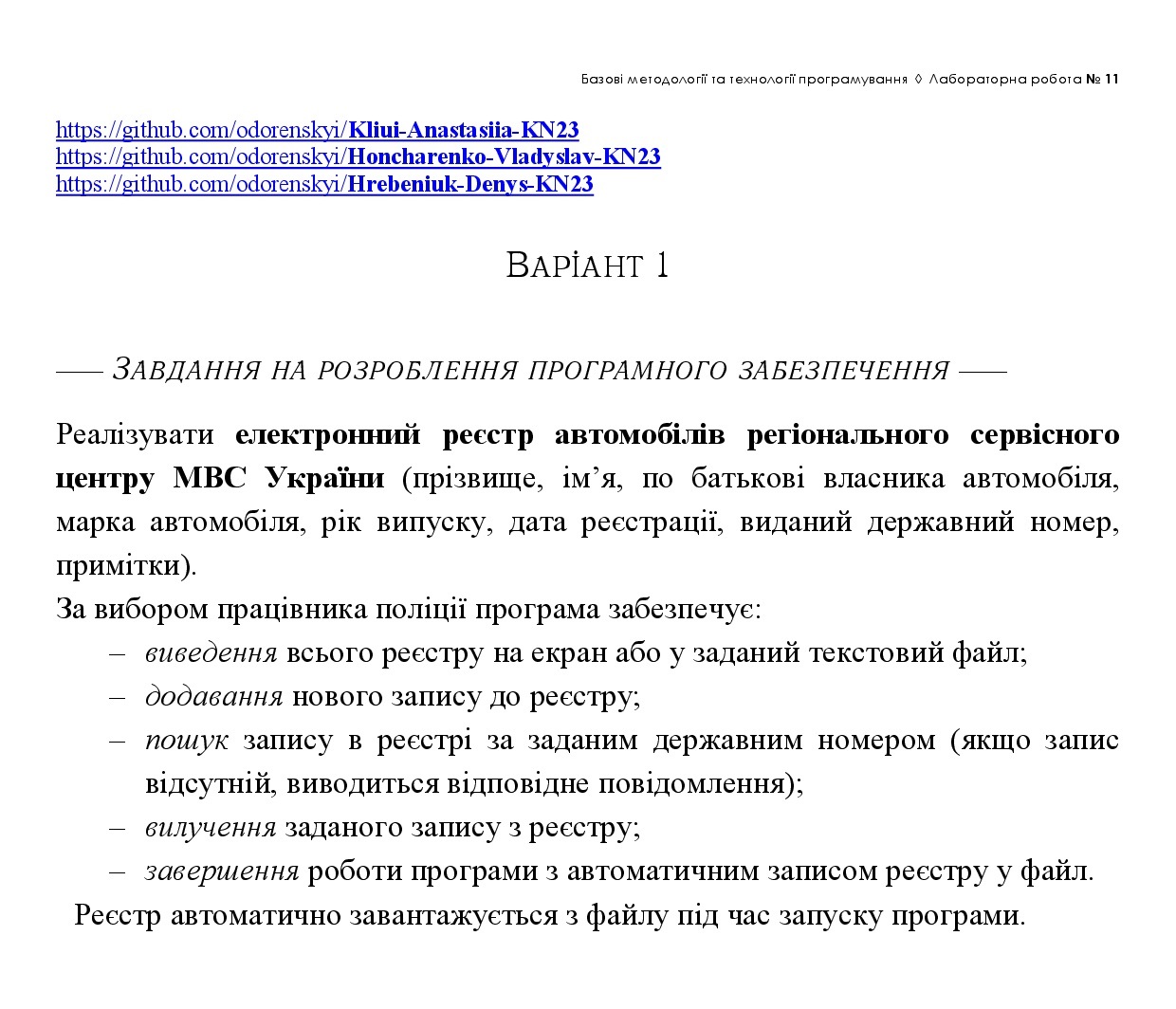
**Мета роботи:** Набуття ґрунтовних вмінь і практичних навичок командної (колективної) реалізації програмного забезпечення, розроблення функцій оброблення динамічних структур даних, використання стандартних засобів С++ для керування динамічною пам’яттю та бінарними файловими потоками.

**Завдання до лабораторної роботи:**

1. У складі команди ІТ-проекта розробити програмні модулі оброблення динамічної структури даних.

2. Реалізувати програмний засіб на основі розроблених командою ІТ-проекта модулів.

**Варіант 1**



**Склад команди IT-проекта:** Гребенюк Денис, Клюй Анастасія, Гончаренко Владислав

**Розроблений план виконання IT-проекта**

**«Електронний реєстр автомобілів»**

**1. Аналіз задач ІТ-проекта та вимог до програмного забезпечення**

**● Ціль проекту:**

* розробка програми для автоматизації процесу ведення  електроного реєстру автомобілів у регіональному центрі МВС України.

**● Функціональні вимоги:**

1. ***Виведення*** всього реєстру на екран або у заданий текстовий файл

2. ***Додавання*** нового запису до реєстру автомобілів з такою інформацією: прізвище, ім'я, по батькові власника, марка автомобіля, рік випуску, дата реєстрації, виданий державний номер та примітки;

3. ***Пошук***запису в реєстрі за даним державним номером;

4. ***Вилучення*** заданого запису з реєстру.

5. ***Завершення***роботи програми з автоматичним записом реєстру у файл.

6.  ***Автоматичне завантаження*** з файлу під час запуску програми

**● Нефункціональні вимоги:**

1. програма повинна бути ефективною та швидкою у роботі;

2. надійність та стійкість програми до помилок та відмов.

**● Дані:**

- прізвище, ім'я, по батькові власника автомобіля;

- марка автомобіля, рік випуску, дата реєстрації;

- виданий державний номер;

- примітки.

***Вимоги до програмного забезпечення***

**●Інтерфейс користувача:**

- інтуїтивно зрозумілий та зручний для введення та виведення інформації.

**● Продуктивність:**

- швидке виконання запитів, оптимізація для великих обсягів даних.

**● Сумісність:**

- підтримка різних операційних систем.

**● Масштабованість:**

- легкість оновлення та розширення функціоналу.

**● Технічні обмеження:**

- врахування можливостей використовуваного обладнання та програмного середовища.

**● Документація:**

- повна документація коду

**2. Специфікації ПЗ, концептуальні проектні рішення, архітектура програмного засобу, загальні алгоритми функціонування та інтерфейси модулів**

***Специфікація ПЗ***

**Обговорення щодо:**

1) вимог до програмного забезпечення;

2) вибрання типу динамічної структури даних (список, стек, черга, дерево) для реалізації бази даних ПЗ;

3) створення заголовочного файлу;

4) розподілити підзадачі реалізації операції над динамічною структурою даних;

5) складання плану робіт проекту з урахуванням стандарту ISO/IEC 12207

***Концептуальні проектні рішення***

**1) Модульна архітектура**

• розподіл системи на незалежні модулі для спрощення розробки та тестування:

• модуль виведення реєстру - відповідає за виведення всього реєстру на екран або у заданий текстовий файл.

• модуль додавання нового запису - відповідає за додавання нового запису про автомобіль до реєстру.

• модуль пошуку за державним номером - відповідає за пошук запису в реєстрі за даним державним номером. Якщо запис не буде знайдено, модуль виведе відповідне повідомлення.

• модуль вилучення запису - відповідає за вилучення заданого запису з реєстру.

• модуль автоматичного запису реєстру у файл - відповідає за автоматичний запис у файл під час завершення роботи програми.

**2) Використання списку для реалізації електронного реєстру автомобілів**

• оптимізація швидкості пошуку, вставки та видалення даних

• поліпшення впорядкувння та доступної інформації

***Архітектура програмного засобу***

**1)Модуль управління даними**

**●** Основною структурою даних, є структура **CarRecord**

**●** Операції:

- виведення реєстру: функція для виведення всього реєстру на екран або у заданий текстовий файл. Ця операція дозволяє користувачу переглядати всі дані, що зберігаються у реєстрі.

- додавання нового запису: функція для додавання нового запису про автомобіль до реєстру. Ця операція дозволяє користувачу ввести дані про новий автомобіль та додати його до реєстру.

- пошук за державним номером: функція для пошуку запису в реєстрі за заданим державним номером. Ця операція дозволяє знайти інформацію про автомобіль за його номером.

- вилучення  запису: функція для вилучення заданого запису з реєстру. Ця операція дозволяє користувачу вилучити інформацію про автомобіль з реєстру.

- завершення: функція для завершення роботи програми з автоматичним записом реєстру у файл. Під час завершення роботи програми всі дані про автомобілі автоматично записуються у файл, щоб забезпечити збереження інформації.

- автоматичне завантаження: під час запуску програми, автоматично завантажує реєстр автомобілів з файлу. Ця операція дозволяє користувачу продовжувати роботу з програмою без необхідності вручного завантаження даних.

**2) *Інтерфейс користувача***

• Консольний інтерфейс: простий текстовий інтерфейс, що дозволяє користувачу вводити команди для управліня записами;

• Взаємодія з користувачем: чіткі інструкції та запити для забезпечення правильного введення даних;

***Загальні алгоритми функціонування***

1.**Виведення всього реєстру на екран або у заданий текстовий файл:**

- проходження через всі записи в реєстрі;

- виведення кожного запису на екран або збереження у файл.

**2. Додавання нового запису до реєстру:**

- перевірка наявності даних про автомобіль;

- введення нових даних;

- збереження нового запису в реєстрі.

**3. Пошук запису в реєстрі за даним державним номером:**

- введення державного номера для пошуку;

- пошук в реєстрі за державним номером;

- виведення результату пошуку або відповідного повідомлення, якщо запис не знайдено.

**4. Вилучення заданого запису з реєстру:**

- введення даних для ідентифікації запису, який потрібно вилучити;

- пошук запису в реєстрі;

- вилучення знайденого запису.

**5. Завершення роботи програми з автоматичним записом реєстру у файл:**

- збереження всіх змін у реєстрі у файл.

-завершення виконання програми.

**6. Автоматичне завантаження реєстру з файлу під час запуску програми:**

- перевірка наявності файлу з реєстром;

- якщо файл існує, завантаження даних з файлу у пам'ять програми;

- якщо файл відсутній, створення порожнього реєстру.

**3.  Обгрунтування вид динамічної структури даних для реалізації бази даних ПЗ, складових її елементів (поля структури і їх типи).**

Для реалізації бази даних програмного забезпечення «Електронний реєстр автомобілів» у мові програмування C++ було обрано двонапрямлений (двозв’язний)  список.

**Обґрунтування цього вибору наступне:**

**1) Швидкий доступ:**

Двонапрямлені списки дозволяють швидше переміщатися як вперед, так і назад по списку. Це важливо для операцій, таких як пошук та видалення записів за державним номером, де може знадобитися звернення до попередніх записів.

**2) Ефективність додавання та вилучення:**

Додавання та видалення елементів у двонапрямленому списку може бути ефективними, оскільки не потрібно переміщатися через весь список для зміни посилань.

**3) Простота реалізації:**

Двонапрямлені списки є достатньо простими для реалізації та керуються досить зрозумілими правилами, що полегшує їх розуміння та використання.

**4) Зручність використання:**

Для операцій, таких як виведення всього реєстру на екран або у заданий текстовий файл, двонапрямлений список також може бути зручним, оскільки його можна ітерувати у будь-якому напрямку.

**5) Навчальний аспект:**

Часто використовується у курсах зі структур даних. Відповідно, він ідеально підходить для проектів, спрямованих на вивчення студентами фундаментальних принципів роботи структур даних.

**Складові елементи структури**

**1) Прізвище власника автомобіля:**

- поле структури: lastName

- тип даних: string

**2) Ім'я власника автомобіля:**

- поле структури: firstName

- тип даних: string

**3) По батькові власника автомобіля:**

- поле структури: middleName

- тип даних: string

**4)  Марка автомобіля:**

- поле структури: carBrand

- тип даних: string

**5) Рік випуску автомобіля:**

- поле структури: year

- тип даних: int

**6) Дата реєстрації автомобіля:**

- поле структури: registrationDate

- тип даних: string

**7) Державний номер автомобіля:**

- поле структури: licensePlate

- тип даних: string

**8) Примітки:**

- поле структури: notes

- тип даних: string

**4. Створення заголовочного файлу**

**Задача:** створити заголовочний файл struct\_type\_project\_N.h (N ― номер варіанта завдання) з описом елементів динамічної структури даних мовою С++

**5. Розподіл завдань між учасниками команди підзадачі з реалізації операцій над динамічною структурою даних відповідно до розробленої архітектури програмного засобу.**

**Клюй Анастасія**

1) ***виведення*** всього реєстру на екран або у заданий текстовий файл: реалізація функції для виведення вмісту дерева на екран чи у текстовий файл.

2) ***додавання*** нового запису до реєстру: розробка методу, який дозволяє користувачу вводити новий запис, і додає цей запис до двійкового дерева пошуку. Це включає збір даних від користувача і створення нового вузла в дереві.

**Гончаренко Владислав**

1) ***пошук*** запису в реєстрі за даним державним номером: розробка функції пошуку, яка дозволяє користувачу ввести державний номер і знаходить відповідний запис у дереві, виводячи інформацію про автомобіль або повідомлення про те, що автомобіль не знайдено.

2) ***завершення*** роботи програми з автоматичним записом реєстру у файл:  розробка методу, що забезпечує збереження всіх даних у файл при закритті програми та їх відновлення при наступному запуску.

**Гребенюк Денис**

1) ***вилучення***заданого запису з реєстру:  реалізація методу видалення запису за введеним користувачем державним номером, що вимагає впевненого видалення вузла з дерева та балансування його після видалення.

2) ***автоматичне*** завантаження реєстру з файлу під час запуску програми:  розробка механізму для читання даних з файлу і відновлення структури дерева при запуску програми, забезпечуючи коректне відновлення стану реєстру.

Модуль складається з функцій для обробки даних реєстру автомобілів, що містяться у двонаправленому списку.

**Основні функції**

1. removeCarFromRegister(List \*list, std::string stateLicensePlate): Видаляє автомобіль з реєстру за номерним знаком.

2. loadRegisterData(std::string registerFileName): Завантажує дані з файлу бази даних у двонаправлений список.

**Проектування**

1. Функція removeCarFromRegister

- Вхідні дані: вказівник на об'єкт типу List (List \*list) і номерний знак автомобіля (std::string stateLicensePlate).

- Логіка:

1) Проходить через список і шукає вузол з відповідним номерним знаком;

2) Якщо вузол знайдено, видаляє його, оновлюючи покажчики сусідніх вузлів;

3) Якщо вузол не знайдено, виводить відповідне повідомлення про помилку.

2. Функція loadRegisterData

- Вхідні дані: ім'я файлу реєстру (std::string registerFileName).

- Логіка:

1) Відкриває файл для читання;

2) Створює змінну-вказівник типу List (List);

3) Зчитує дані з файлу, формуючи вузли списку;

4) Заповнює кожен вузол даними про автомобіль;

5) Установлює покажчики між вузлами.

**Тестування**

1. Тестування функції removeCarFromRegister

- Позитивні тести:

1) Видалення автомобіля, що є в середині списку;

2) Видалення автомобіля, що є першим елементом списку;

3) Видалення автомобіля, що є останнім елементом списку.

- Негативні тести:

1) Видалення автомобіля, якого немає в списку

2. Тестування функції loadRegisterData

- Позитивні тести:

1) Завантаження непорожнього файлу з даними;

2) Завантаження файлу з одним записом.

- Негативні тести:

1) Спроба завантаження неіснуючого файлу

**Лістинг ModulesHrebeniuk**

#include <iostream>

#include <string>

#include <fstream>

#include "struct\_type\_project\_1.h"

using namespace std;

void removeCarFromRegister(List \*list, std::string stateLicensePlate) {

Node \*current = list->head;

while (current != nullptr) {

if (current->data.licensePlate != stateLicensePlate) {

current = current->next;

continue;

}

if (current == list->head) {

list->head = current->next;

if (list->head != nullptr) {

list->head->previous = nullptr;

}

} else if (current == list->tail) {

list->tail = current->previous;

if (list->tail != nullptr) {

list->tail->next = nullptr;

}

} else {

current->previous->next = current->next;

current->next->previous = current->previous;

}

Node \*nodeToDelete = current;

current = current->next;

delete nodeToDelete;

cout << "Автомобіль з державним номером " << stateLicensePlate << " було видалено з реєстру." << endl;

return;

}

cerr << "Автомобіль з державним номером " << stateLicensePlate << " для видалення не знайдено в реєстрі." << endl;

}

List \*loadRegisterData(std::string registerFileName) {

ifstream registerFile(registerFileName, ios::binary | ios::in);

if (!registerFile) {

cerr << "Не вдалось відкрити " << registerFileName << " для того, щоб завантажити реєстр!" << endl;

return nullptr;

}

List \*list = new List;

list->head = nullptr;

list->tail = nullptr;

Node \*current = list->head;

size\_t stringSize;

registerFile.seekg(0, registerFile.end);

int fileLength = registerFile.tellg();

registerFile.seekg(0, registerFile.beg);

while (registerFile.tellg() < fileLength) {

if (current != nullptr) {

current->next = new Node;

current->next->previous = current;

current = current->next;

} else {

list->head = new Node;

list->head->previous = nullptr;

current = list->head;

}

CarRecord data;

registerFile.read((char\*) &stringSize, sizeof(size\_t));

data.lastName.resize(stringSize);

registerFile.read((char\*) &data.lastName[0], stringSize);

registerFile.read((char\*) &stringSize, sizeof(size\_t));

data.firstName.resize(stringSize);

registerFile.read((char\*) &data.firstName[0], stringSize);

registerFile.read((char\*) &stringSize, sizeof(size\_t));

data.middleName.resize(stringSize);

registerFile.read((char\*) &data.middleName[0], stringSize);

registerFile.read((char\*) &stringSize, sizeof(size\_t));

data.carBrand.resize(stringSize);

registerFile.read((char\*) &data.carBrand[0], stringSize);

registerFile.read((char\*) &data.year, sizeof(int));

registerFile.read((char\*) &stringSize, sizeof(size\_t));

data.registrationDate.resize(stringSize);

registerFile.read((char\*) &data.registrationDate[0], stringSize);

registerFile.read((char\*) &stringSize, sizeof(size\_t));

data.licensePlate.resize(stringSize);

registerFile.read((char\*) &data.licensePlate[0], stringSize);

registerFile.read((char\*) &stringSize, sizeof(size\_t));

data.notes.resize(stringSize);

registerFile.read((char\*) &data.notes[0], stringSize);

current->data = data;

}

registerFile.close();

if (list->head != nullptr) {

current->next = nullptr;

list->tail = current;

}

return list;

}

**Лістинг prj\_1\_Hrebeniuk**

#include <iostream>

#include <ctime>

#include <fstream>

#include <conio.h>

#include "struct\_type\_project\_1.h"

#include "ModulesHrebeniuk.h"

#include "ModulesHoncharenko.h"

#include "ModulesKlui.h"

#define REGISTER\_NAME "register\_database"

using namespace std;

void displayMenu() {

system("cls");

cout << " ======================= Реєстр МВС України ========================= " << endl;

cout << "| 1. Додати автомобіль у реєстр |" << endl;

cout << "| 2. Пошук автомобіля за номером |" << endl;

cout << "| 3. Видалення автомобіля з реєстру за номерним знаком |" << endl;

cout << "| 4. Вивести усі дані з реєстру |" << endl;

cout << "| 5. Вийти з програми |" << endl;

cout << " ==================================================================== " << endl;

cout << "Виберіть опцію: ";

}

void exitApp(List \*list) {

delete list->head;

delete list->tail;

delete list;

exit(0);

}

int main()

{

system("chcp 1251 & cls");

List \*list = loadRegisterData("register\_database");

if (list == nullptr) {

list = new List;

list->head = nullptr;

list->tail = nullptr;

}

int choice;

do {

displayMenu();

cin >> choice;

cin.ignore();

system("cls");

switch (choice) {

case 1: {

addCarInRegister(list);

saveRegisterData(list, REGISTER\_NAME);

break;

}

case 2: {

string searchLicensePlate;

cout << "Введіть номер авто для пошуку: ";

cin >> searchLicensePlate;

if (searchLicensePlate.empty() || searchLicensePlate.size() > 10) {

cout << "Помилка: номерний знак не може бути порожнім та його максимальна довжина повинна бути не більше 10 символів!" << endl;

break;

}

searchCarInRegister(list, searchLicensePlate);

break;

}

case 3: {

string removeLicensePlate;

cout << "Введіть номер авто для видалення: ";

cin >> removeLicensePlate;

if (removeLicensePlate.empty() || removeLicensePlate.size() > 10) {

cout << "Помилка: номерний знак не може бути порожнім та його максимальна довжина повинна бути не більше 10 символів!" << endl;

break;

}

removeCarFromRegister(list, removeLicensePlate);

saveRegisterData(list, REGISTER\_NAME);

break;

}

case 4: {

outputRegisterData(list);

break;

}

case 5: {

cout << "До побачення!" << endl;

exitApp(list);

break;

}

default:

cout << "Невірний вибір. Спробуйте ще раз." << endl;

}

cout << "Продовжити роботу застосунку? (y/n): ";

choice = getch();

} while (choice == 'Y' || choice == 'y');

exitApp(list);

return 0;

}

**Висновок**

Виконуючи цю лабораторну роботу, я здобув важливі знання та практичний досвід у розробці програмних засобів для оброблення динамічних структур даних та бінарних файлів. Під час роботи я опанував використання стандартних засобів С++ для керування динамічною пам’яттю, а також навчився ефективно працювати з файловими потоками. Особливу увагу приділено реалізації функцій з видалення елементів зі списку та завантаження даних із файлів, що дозволило мені глибше зрозуміти принципи роботи з динамічними структурами даних.

Завдання, які було виконано, включали розробку програмних модулів у складі команди ІТ-проекту, де важливою частиною було проектування та реалізація функцій оброблення динамічних структур даних. У ході роботи я навчився застосовувати оператори `new` та `delete` для виділення та звільнення пам’яті, що дозволило мені ефективно керувати ресурсами програми.

Також було реалізовано функції для роботи з файловими потоками, що включали відкриття, читання та запис бінарних файлів. Це дозволило мені здобути практичні навички у збереженні та завантаженні даних, що є важливим аспектом при роботі з великими обсягами інформації, особливо базами даних.

Розробка та тестування функцій проходили з використанням як позитивних, так і негативних тестів, що допомогло мені вдосконалити навички виявлення та усунення помилок у програмному забезпеченні. Всі функції були протестовані та показали коректну роботу, що підтвердило правильність вибраного підходу до реалізації задач.

Завдяки цій лабораторній роботі я значно покращив свої знання та практичні навички у розробці модульних проектів, роботі з динамічними структурами даних та бінарними файлами. Мені було цікаво виконувати цю лабораторну роботу, отримувати нові знання та проектувати задачі такого рівня складності.

**Контрольні запитання і завдання**

**1.** *Що у програмуванні розуміють під часом життя динамічних змінних (об’єктів)? Наведіть приклад мовою С/С++.*

Час життя динамічних змінних (об'єктів) - це період, протягом якого пам'ять, виділена для об'єкта, є дійсною і може бути використана. Час життя динамічної змінної починається з моменту виділення пам'яті за допомогою оператора new і закінчується, коли пам'ять звільняється за допомогою оператора delete.

Приклад:

int main() {

int\* ptr = new int; // Виділення пам'яті для динамічної змінної

\*ptr = 10; // Використання динамічної змінної

delete ptr; // Звільнення пам'яті динамічної змінної

return 0;

}

**2.** *Дайте визначення вказівникові та посиланню. Перелічіть випадки, у яких їх варто й доцільно використовувати.*

Вказівник - це змінна, яка зберігає адресу іншої змінної або об'єкта в пам'яті. Вказівники дозволяють отримувати доступ до даних за адресою і можуть бути використані для передачі даних за посиланням, динамічного виділення пам'яті тощо.

Посилання - це альтернативне ім'я для існуючого об'єкта. Посилання надає інший спосіб доступу до того самого об'єкта і повинно бути ініціалізоване при оголошенні.

Випадки, коли варто використовувати вказівники:

- Динамічне виділення пам'яті;

- Передача даних за посиланням у функції;

- Реалізація зв'язаних структур даних (списки, дерева тощо).

Випадки, коли варто використовувати посилання:

- Передача об'єктів у функції без копіювання;

- Повернення посилань на об'єкти з функцій;

- Реалізація оператора присвоєння для класів.

**3.** *Яким є синтаксис оголошення вказівників мовою програмування С/С++? Наведіть приклад оголошення вказівника та його ініціалізації адресою статичного об’єкта (змінної).*

Синтаксис оголошення вказівника:

int\* ptr; // Оголошення вказівника на int

Приклад оголошення вказівника та його ініціалізації адресою статичної змінної:

int value = 10;

int\* ptr = &value; // Ініціалізація вказівника адресою статичної змінної

**4.** *Скільки байт пам’яті виділяється для порожнього вказівника? Відповідь обґрунтуйте і доведіть експериментально.*

Розмір порожнього вказівника залежить від архітектури системи. На 32-бітних системах розмір вказівника зазвичай становить 4 байти, а на 64-бітних системах - 8 байтів. Це пов'язано з тим, що вказівник зберігає адресу пам'яті, а розмір адреси залежить від розрядності системи.

Експериментальне доведення:

#include <iostream>

int main() {

int\* ptr = nullptr;

std::cout << "Size of empty pointer: " << sizeof(ptr) << " bytes" << std::endl;

return 0;

}

Вивід на 64-бітній системі:

Size of empty pointer: 8 bytes

**5.** *Перелічіть допустимі операції над вказівниками та оператори С/С++, які їх реалізовують.*

Допустимі операції над вказівниками та оператори:

- Оператор взяття адреси (&): отримує адресу змінної або об'єкта.

- Оператор розіменування (\*): доступ до значення за адресою, на яку вказує вказівник.

- Арифметичні операції (+, -, ++, --): дозволяють переміщуватися по масиву або послідовності елементів.

- Операції порівняння (==, !=, <, >, <=, >=): порівняння адрес вказівників.

- Оператор присвоєння (=): присвоєння адреси вказівнику.

- Оператор приведення типу (static\_cast, dynamic\_cast, const\_cast, reinterpret\_cast): зміна типу вказівника.

**6.** *Яке призначення ключових слів new та delete у С++?*

Ключові слова new та delete у C++ використовуються для динамічного виділення та звільнення пам'яті:

- Оператор new виділяє пам'ять для об'єкта або масиву об'єктів і повертає вказівник на виділену пам'ять.

- Оператор delete звільняє пам'ять, яка була виділена за допомогою new.

**7.** *В чому полягає відмінність між виконанням операції delete та delete[] у С++?*

Відмінність між delete та delete[]:

- delete використовується для звільнення пам'яті, виділеної для одного об'єкта за допомогою new.

- delete[] використовується для звільнення пам'яті, виділеної для масиву об'єктів за допомогою new[].

Якщо використати delete замість delete[] для масиву, це призведе до неповного звільнення пам'яті.

**8.** *Що міститиме stm при: short ptt,\*stm = &ptt;?*

При оголошенні short ptt, \*stm = &ptt;:

- ptt - змінна типу short

- stm - вказівник на змінну типу short, який ініціалізується адресою змінної ptt

Тобто stm міститиме адресу змінної ptt.

**9.** *Яке ключове слово мови програмування С/С++ реалізує операцію визначення розміру змінної (об’єкта) або типу даних?*

Для визначення розміру змінної або типу даних у C/C++ використовується оператор sizeof.

**10.** *Яким є синтаксис запису та механізм виконання унарної операції опосередкованої адресації у С/С++?*

Синтаксис унарної операції опосередкованої адресації у C/C++: \*ptr, де ptr - вказівник. Механізм: операція \* повертає значення, на яке посилається вказівник ptr. Тобто, якщо ptr вказує на область пам'яті, то \*ptr поверне значення, що зберігається за цією адресою.

**11.** *Мовою програмування С++ наведіть приклад створення динамічного одновимірного масива та звільнення виділеної для нього динамічної пам’яті.*

Приклад:

int\* arr = new int[10]; // виділення пам'яті під масив з 10 елементів

// ... використання масиву ...

delete[] arr; // звільнення пам'яті

**12.** *Що буде включено у стандартний потік при:*

*double \*dptr;*

*short \*sptr;*

*cout << (sizeof dptr - sizeof sptr);*

*Відповідь обґрунтовано поясніть.*

Вираз sizeof dptr - sizeof sptr поверне різницю між розмірами вказівників dptr і sptr в байтах. Розмір вказівника залежить від розрядності системи (32/64 біт), але не залежить від типу даних, на який він вказує. Тому в більшості випадків розміри вказівників будуть однакові, і в потік буде виведено 0.

**13.** *Яка константа класу ios С++ призначена для відкриття файлового потоку в двійковому режимі? Наведіть приклад відкриття бінарного файла для запису даних у його кінець.*

Константа ios::binary.

Приклад відкриття бінарного файлу для дозапису:

ofstream outputFile("data.bin", ios::binary | ios::app);

**14.** *Яка функція-член об’єкта fstream С++ забезпечує читання заданої кількості байт з асоційованого потоку в змінну-буфер, а яка ― включення (запис) даних у потік?*

Для читання з потоку використовується функція read(), а для запису - write().

Наприклад:

char buffer[100];

file.read(buffer, sizeof(buffer)); // читання у буфер

const char\* data = "Hello";

file.write(data, strlen(data)); // запис у файл

**15.** *Що буде записано у файл, якщо в асоційований з ним бінарний потік включити рядок символів у кодуванні UTF-8?*

У бінарний файл будуть записані байти, що представляють символи рядка в кодуванні UTF-8. Тобто кожен символ буде представлений послідовністю від 1 до 4 байт, залежно від самого символу. При читанні цих даних назад, треба враховувати кодування і правильно інтерпретувати отримані байти як символи UTF-8.

**16.** *Дайте визначення наступним динамічним структурам даних: список, черга, стек, бінарне дерево. Перелічіть випадки (задачі), у яких є доцільним представлення інформації саме означеними структурами.*

Список (Linked List) - це лінійна структура даних, яка складається з вузлів, кожен з яких містить дані та посилання на наступний вузол. Списки можуть бути односпрямованими або двоспрямованими. Вони корисні для зберігання та маніпулювання динамічними наборами даних, де розмір може змінюватися під час виконання.

Черга (Queue) - це лінійна структура даних, яка працює за принципом "перший прийшов - перший вийшов" (FIFO). Елементи додаються в кінець черги і видаляються з початку. Черги корисні для моделювання реальних сценаріїв, таких як обробка завдань, планування процесів тощо.

Стек (Stack) - це лінійна структура даних, яка працює за принципом "останній прийшов - перший вийшов" (LIFO). Елементи додаються і видаляються з одного кінця стеку, який називається вершиною. Стеки корисні для відстеження стану, реалізації рекурсії, обробки виразів тощо.

Бінарне дерево (Binary Tree) - це нелінійна структура даних, де кожен вузол має не більше двох дочірніх вузлів (лівий і правий). Бінарні дерева корисні для ефективного пошуку, сортування та представлення ієрархічних даних.

**17.** *В чому полягає відмінність одно- від двонаправленого списка з погляду їх реалізації? Наведіть приклад опису мовою С++ елемента одно- та двонаправленого списків і виділення для них динамічної пам’яті.*

Основна відмінність між односпрямованим і двоспрямованим списком полягає в тому, що односпрямований список має посилання лише на наступний вузол, тоді як двоспрямований список має посилання як на наступний, так і на попередній вузол.

Приклад опису елемента односпрямованого списку:

struct Node {

int data;

Node\* next;

};

Node\* newNode = new Node;

Приклад опису елемента двоспрямованого списку:

struct Node {

int data;

Node\* previous;

Node\* next;

};

Node\* newNode = new Node;

**18.** *Перелічіть допустимі операції над лінійними списками як простими динамічними структурами даних.*

Допустимі операції над лінійними списками:

- Вставка елемента на початок, кінець або задану позицію

- Видалення елемента з початку, кінця або заданої позиції

- Пошук елемента за значенням

- Доступ до елемента за позицією

- Перевірка, чи список порожній

- Отримання розміру списку

**19.** *Які розрізняють потоки і в чому полягає їх відмінність з погляду використання під час реалізації програмного забезпечення?*

В C++ розрізняють такі основні типи потоків:

- Вхідні потоки (istream) - використовуються для зчитування даних. Наприклад, cin - стандартний вхідний потік для зчитування з клавіатури, ifstream - для зчитування з файлу.

- Вихідні потоки (ostream) - використовуються для запису даних. Наприклад, cout - стандартний вихідний потік для виведення в консоль, ofstream - для запису в файл.

- Потоки введення/виведення (iostream) - поєднують функціональність вхідних і вихідних потоків. Дозволяють як зчитувати, так і записувати дані. Наприклад, fstream дозволяє читати і писати файли.

- Рядкові потоки (stringstream) - працюють з рядками в пам'яті. Дозволяють перетворювати рядки в інші типи даних і навпаки. Включають istringstream, ostringstream, stringstream.

Основні відмінності:

- Напрям передачі даних (введення чи виведення)

- Джерело/призначення даних (консоль, файл, рядок в пам'яті)

- Можливості форматування і керування станом потоку

**20.** *Сформулюйте алгоритм запису в двійковий (бінарний) файл однонаправленого списка. Чим він відрізняється від алгоритмів зберігання у файл стека та черги?*

1. Відкрити двійковий файл для запису за допомогою функції ofstream з параметром ios::binary.

2. Пройтись по списку, починаючи з голови (першого елемента).

3. Для кожного елемента списку:

- Записати у файл значення елемента за допомогою функції write(), передавши адресу елемента та його розмір.

- Перейти до наступного елемента списку.

4. Повторювати крок 3, поки не буде досягнуто кінця списку (коли вказівник на наступний елемент дорівнює nullptr).

5. Закрити файл за допомогою функції close().

Стек:

- При записі стека у файл, елементи записуються у зворотному порядку (з вершини стека до дна).

- При зчитуванні елементів зі стека з файлу, вони відновлюються у зворотному порядку запису.

Черга:

- При записі черги у файл, елементи записуються у тому ж порядку, що і в черзі (з початку черги до кінця).

- При зчитуванні елементів з черги з файлу, вони відновлюються у тому ж порядку, що і були записані.