**Лабораторна робота №14 Написання програм з використанням функцій та файлів.**

**На період** карантину в дистанційній формі навчання потрібно використовувати замість **С++ Builder онлайн компілятор C++ Shell, який доступний за адресом** [**http://cpp.sh**](http://cpp.sh) (інструкція надана після тексту ЛР№3). При наявності на домашньому комп’ютері іншого компілятору С++ завдання можна виконати на ньому. Результати викласти на платформу коледжу в свій репозиторій та надсилати на електронну адресу викладача [**t.i.lumpova@gmail.com**](mailto:t.i.lumpova@gmail.com)у вигляді cpp-файлу з іменем у форматі

**<Номер групи><Номер лабораторної>[-<Номер завдання>] <Прізвищеанглійською>**

Наприклад,21-1Loban.cpp.

При відсутності можливості доступу до Інтернету текст програми набрати в Блокноті або WordPad Windows та надіслати на електронну адресу викладача

**Увага!** C++Shell не підтримує роботу з файлами.

Для роботи з файлами можна використати безкоштовний компілятор Dev-C++, який потрібно встановити на ваш комп’ютер. Скачати його можна за посиланням:

**https://sourceforge.net/projects/orwelldevcpp/**

Інструкція по роботі з компілятором надається за посиланням (але там забагато реклами та іншого сміття):

[**https://studopedia.ru/18\_64672\_pokrokove-vikonannya-programi.htm**](https://studopedia.ru/18_64672_pokrokove-vikonannya-programi.htm)

Якщо у Вас нема можливості встановити цю програму, то просто запишіть код виконання операцій роботи з файлами та надішліть на перевірку.

Іншим рішенням є надсилання поштою посилання на текст програми за URL адреси, яку надає C++Shell, вказавши в темі листа, номер групи прізвище студента та номер ПР.

В темі листа вказати, номер групи, прізвище студента та номер ПР як "ПР№8".

**Строк відсилки ЛР 04.05.2021**

**Мета**: навчитися писати програми **на мові С++** в **консольному режимі** з використанням функцій та файлів**.**

**Методичні вказівки щодо організації самостійної роботи студентів**

1. Повторити Лекції 9, 10 теоретичні відомості з Практичних робіт №7,8 та з Лабораторних робіт №7 – 13, а також до цієї Лабораторної роботи.
2. Запустити середовище програмування С++ .
3. Записати програму, що виконує завдання з пп.4. В першому рядку програми записати

*// ПТБД-21 Група № Прізвище*

вказавши номер своєї групи та своє прізвище.

Вхідні дані ввести, а результати вивести, використовуючи потокове введення-виведення даних.

**Мета**: навчитися писати програми з використанням з використанням файлів для введення/виведення даних сивів **на мові С++ в консольному режимі.**

**Завдання.**

1. Прочитати (повторити) основні теоретичні відомості. Переглянути лекції №№10-11
2. Запустити середовище програмування С++ . Записати програму, що виконує 2 завдання (пп.3,4) шляхом виклику відповідної функції. В функціях повинен проводитися аналіз результатів звернення до файлу з обробленням помилок. В першому рядку програми записати

*// Група № Прізвище*

вказавши номер своєї групи та своє прізвище

1. Перша функція виконує такі дії:

* Запитує фразу;
* Записує її до бінарного файлу;
* Для перевірки запису файл відкривається для читання, зчитується інформація та виводиться на консоль;
* Запитує номер позиції для читання та кількість літер;
* Через команди прямого доступу зчитується запитаний фрагмент та виводиться на консоль.

1. Друга функція виконує такі дії:

- записує в бінарний файл дані про сканер із приведеної нижче структури scan\_info. Структура файлу: у перших двох байтах розміщується значення типу int, що визначає кількість зроблених у файл записів; далі без пропусків розміщаються записи про сканери. Дані про сканер можна занести ініціалізацією (відповідно, створивши масив структур) або ввести з клавіатури - 3–4 записи. Інформація по кожному сканеру перед записом до файлу виводиться на консоль;

- запитує номер потрібного запису, зчитує потрібну інформацію та виводить на консоль.

Для збереження даних про планшетні сканери описати структуру виду:

struct scan\_info{

char model[25]; // найменування моделі

int price; // ціна

double x\_size; // гориз. розмір обл. сканування

double y\_size; // вертик. розмір обл. сканування

int optr; // оптичне розрішення

int grey; // число градацій сірого

};

1. Результати надсилати на електронну адресу викладача [**t.i.lumpova@gmail.com**](mailto:t.i.lumpova@gmail.com)

***Контрольні запитання для самоперевірки***.

1. Як класифікують файли за типом?
2. Визначте основні атрибути файлу.
3. Як працює функція fopen()?
4. Чим різняться текстові та бінарні файли?
5. Яке значення повертає функція fgetc() під  час  виникнення  помилки  або  досягнення кінця файлу?
6. Яка функція призначена для встановлення індикатора позиції файлу у необхідному місці?
7. Яку функцію використовують  для  визначення факту  виникнення  помилки у процесі виконання операції з файлом?
8. Як ми перевіряємо відкриття файлу для потоку **ifstream?**

**ТЕОРЕТИЧНІ ВІДОМОСТІ**

**Структуровані файли**

Структурований файл – це окремий випадок двійкового файлу. В якості порції даних в таких файлах використовують структури. Для ініціалізації, відкриття, зчитування та запису даних до структурованих файлів застосовуються ті ж самі функції,які використовуються при роботі з двійковими файлами. Структуровані файли дозволяють за одне звернення записати/зчитати всі поля структури, що значно пришвидшує роботу з файлом.

Приклад запису до файлу структури, що містить 2 поля – ціле (b.n - 4 байта) і дійсне (b.r - 4 байта):

**#include<stdio.h>**

**#include<windows.h>**

**#include"math.h"**

**int main()**

**{ system("color F0");**

**FILE\*f1;** // оголошення вказівника на файл

**struct{** // опис структури даних

**int n;**

**float r;} b;**

// відкриття файлу для запису

**f1 = fopen("file1","wb");**

**printf("Write to file");**

**for (int j = 1; j < 6; j++)**

**{**// заповнення полів структури

**b.n = j;b.r = sqrt(float(j));**

**//** запис елементу структури до файлу

**fwrite(&b, sizeof(b), 1, f1);**

**printf("\n%d %f", b.n, b.r);**

**}**

**fclose(f1); //** закрити файл

**printf("\n");**

**//** відкриття файлу для зчитування

**f1 = fopen("file1","rb");**

**printf("Read from file\n");**

**int k = 0; //** 0–для 1-го блоку

**while(!feof(f1)) //** доки не досягли кінця файлу

**{//** встановити вказівник перед зчитуваним блоком

**fseek(f1, (k)\*sizeof(b), SEEK\_SET);**

**//** зчитати блок даних розміром sizeof(b) – 1 елемент структури

**if(fread(&b, sizeof(b), 1, f1))**

**//** вивести результат на екран

**printf("%d %f\n", b.n, b.r);**

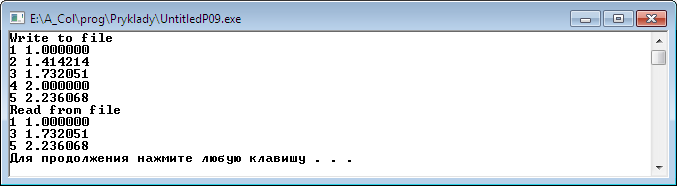
**k+=2; //** пропустити запис

**}**

**system("pause");**

**return 0;}**

Результат



***Пригадаємо.***

Обробка інформації, що зберігається у вигляді файлу передбачає наступні дії:

* визначення змінної – файлового покажчика;
* відкривання та закривання потоку;
* введення-виведення (символів, рядків, форматованих даних, порцій даних певної довжини);
* аналіз можливих помилок операцій введення-виведення;
* керування буферізацією потоку (розміром буферу);
* керування буферним покажчиком.

Потоки бувають двох типів: текстові та двійкові.

Кожний потік має керівну структуру типу FILE , що містить усю необхідну інформацію для роботи з ним. Змінна, що буде представляти потік визначається як покажчик на структуру типу FILE.

Наприклад:  
 FILE \*fp;

Змінна fp зображує потік у подальшій роботі з файлом.

Опис типу FILE , а також прототипи більшості функцій, макросів та констант файлової системи містяться у заголовному файлі <cstdio> (та <stdio.h>).

Потік можна відкрити для читання або/та запису в текстовому або двійковому режимі. Функція відкриття потоку має формат:

FILE \*fopen(const char \*filename, const char \*mode);

**В минулій ЛР ми розглядали цей оператор. Що визначає mode і які значення може приймати?**

Якщо відкривання було успішним, функція повертає *покажчик файлу*, що містить всю необхідну для роботи з потоком інформацію; інакше функція повертає NULL.

Параметр filename – задає ім`я файлу у вигляді рядка в *стилі С*.

**Як ви гадаєте чим рядок в *стилі С буде різнитися від рядка в "стилі" С++?***

Параметр mode – визначає режим відкривання файлу.

* “r” – відкриття існуючого файлу для читання;
* “w” – створення нового файлу для запису (якщо файл з таким ім`ям існує – він перезаписується);
* “a” – відкриття існуючого файлу для додавання в його кінець нової інформації;
* “r+” – відкриття існуючого файлу для читання й запису;
* “w+” – створення нового файлу для читання й запису (перезаписується файл з таким ім`ям існує, якщо існує);
* “a+” – відкриття існуючого файлу для читання та додавання в його кінець нової інформації.

Режим може містити символи t – текстовий, або b – двійковий. По умовчанню передбачається текстовий режим (t).

Приклади

FILE \*fp;

fp = fopen(“file1.txt”, “a+”);

FILE \*f = fopen(“c:\\temp\\data.dat”, “rb+”);

FILE \*fp;

if (fp = fopen(“file2.txt”, “r”) == NULL)

{perror(“ERROR open file2.txt”); return 1;}

FILE \*fp = fopen(“.\\data.dat”, “w”);

Потік закривається або при завершенні програми, або явно функцією fclose:  
int fclose(FILE \* );

Якщо потік, заданий параметром-покажчиком файлу, був відкритий для запису, то перед закриттям у файл записуються данні, що містяться у буферах потоку. У випадку успішного закриття функція повертає 0, інакше – EOF.

Рекомендується завжди явно закривати потоки, що відкриті для запису, щоб уникнути втрати даних.  
 int fflush(FILE \* ); - примусово скидає буфер у файл.

Максимальна кількість одночасно відкритих файлів визначається макросом FOPEN\_MAX.

Операції введення/виведення завжди виконуються починаючи з поточної позиції потоку, що визначається *покажчиком потоку*. *Покажчик потоку* встановлюється при відкритті на початок, або кінець (в залежності від режиму відкриття) й змінюється автоматично при виконанні операцій введення/виведення. Поточне положення *покажчика потоку* можна отримати функціями ftell, fgetpos й задати функціями fseek, fsetpos . Ці функції не можна використовувати для *стандартних потоків*.

**В минулій ЛР ми розглядали оператор** fseek**.**

**Повторимо важливі оператори.**

**Запис потоку байтів**

fwrite – записує об`єкт у файл, відкритий у двійковому режимі. Функція повертає кількість записаних об`єктів. *Формат*:  
size\_t fwrite(const void \*Buf, size\_t Size, size\_t Count, FILE \*fp);

Buf – покажчик на об`єкт;

Size – розмір об`єкту у байтах;

Count – кількість об`єктів, що записуються;

fp – файловий-покажчик.

**Приклад.**

#include <cstdio>

#include <iostream>

using namespace std;

int main() {

FILE \*fp = fopen("d:\\Tmp\\data1.dat", "wb");

struct Point {

int x, y;

} point;

point.x = 10; point.y = 20;

int x = fwrite(&point, sizeof(Point), 1, fp);

cout << x << endl;

system("pause"); return 0;

}

**Читання потоку байтів**

fread – зчитує об`єкт з файлу, відкритого у двійковому режимі. Функція повертає кількість зчитаних об`єктів. *Формат*:  
size\_t fread(void \*Buf, size\_t Size, size\_t Count, FILE \*fp);

Buf – покажчик на об`єкт;

Size – розмір об`єкту у байтах;

Count – кількість об`єктів, що записуються;

fp – файловий-покажчик.

**Обробка помилок**

Функції роботи з потоком повертають значення, які потрібно аналізувати й обробляти помилкові ситуації. Крім того використовують функції:

int feof(FILE \*) – повертає ненульове значення, якщо досягнутий кінець файлу й 0 у протилежному випадку.

int ferror(FILE \*) – повертає ненульове значення, якщо виявлена помилка введення/виведення й 0 у протилежному випадку.

До функцій потрібно звертатись відразу після виконання операцій з файлом.

**Функції для роботи з буферним покажчиком**

Дозволяють здійснювати читання та запис у файл з довільної позиції. Використовуються в основному для файлів відкритих у двійковому режимі.

long ftell(FILE \*fp) – повертає поточну позицію покажчика (при помилці ).

int fgetpos(FILE \*fp, fpos\_t \*pos) – записує поточну позицію покажчика у другий параметр, повертає 0, якщо помилок не має, інакше ненульове значення.

void rewind(FILE \*fp) – встановлює покажчик на початок файлу.

int fsetpos(FILE \*fp, const fpos\_t \*pos) – встановлює покажчик за параметром pos , повертає 0, якщо помилок не має, інакше ненульове значення.

int fseek(FILE \*fp, long offset, int origin) – встановлює покажчик згідно зміщення offset відносно позиції origin.

Для параметру origin можуть бути вказані макроси:  
SEEK\_SET – початок файлу;  
SEEK\_CUR – поточна позиція покажчика;  
SEEK\_END – кінець файлу.

Тут и в ЛР№13 розглянули роботу з файлами з використанням введення-виведення в "стилі С" (с-style)

**В стилі мови С++** використовуються набір класів, що забезпечують потокове введення та виведення. Для цього підключається заголовочний файл iostream. Для введення/виведення у файл/з файлу використовують потоки, які можуть бути пов'язані із файлом на диску іїх використання потрібно підключити заголовочний файл <**fstream**>.

**Для яких цілей використовуються потоки ofstream, ifstream, fstream?**

**Пригадаємо приклади відкриття файлу, вкажіть які режими роботи з файлами визначаються:**

fstream fs("f1.txt");

ifstream ifs("f2.txt");

ofstream ofs("f3.txt");

fstream fs("f1.txt", ios::in | ios::out | ios::trunk);

ifstream ifs("f2.txt", ios::in | ios::binary);

**Які дії виконуються в операторах?**

ofstream ofs;

ofs.open("f3.txt");

**Як ми виконуємо введення та виведення даних при роботі з файлами?**

**Прямий доступ в стилі мови С++**

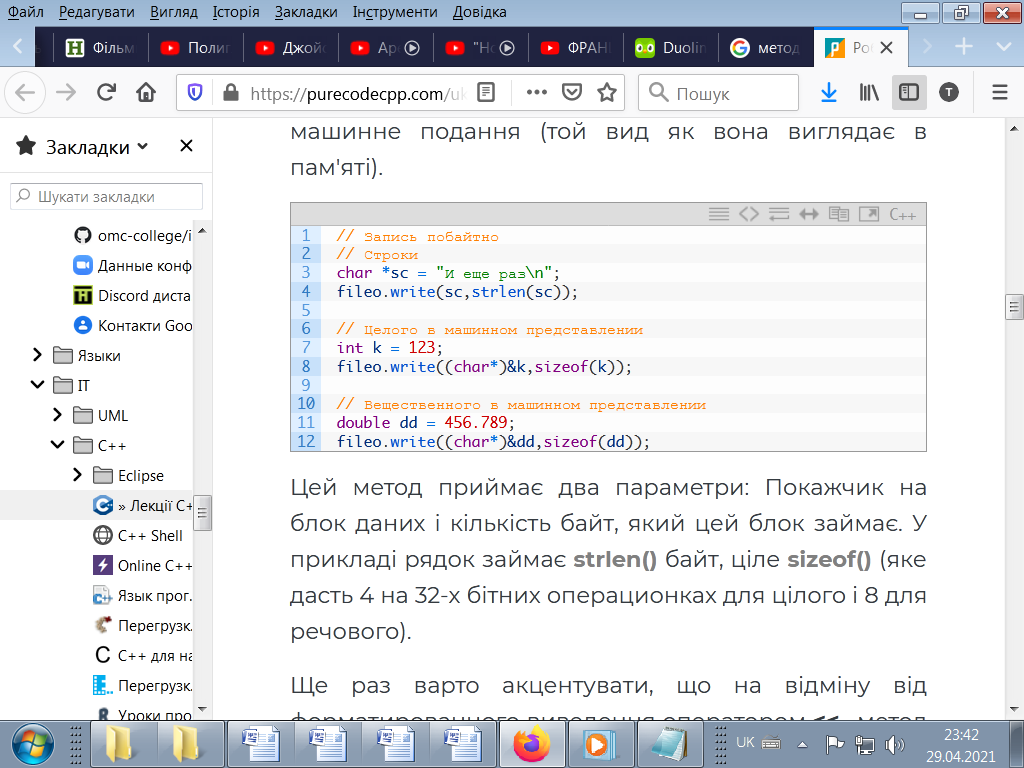
**Функція (метод) write**

Використовується в бінарних файлах для запису блоку пам'яті (масиву байт) в файл у вигляді як вони є. Будь-яка змінна так само є масивом байт, вірніше, її так можна розглядати. Відповідно цей метод запише в файл її машинне подання (той вид як вона виглядає в пам'яті).

Ця функція (цей метод) приймає два параметри: Покажчик на блок даних і кількість байт, який цей блок займає. У прикладі далі рядок займає **strlen()** байт, ціле **sizeof().**

На відміну від форматованого виведення оператором **<<**,  **функція (метод) write**  не виводить дані в текстовому поданні.

**Приклад**

****

**Функція (метод) read** використовується аналогічно.

Функції write() і read() зручно використовувати для організації прямого доступу до даних у файлі. Але для цього потрібні також функції, які дозволяють переміщати покажчик потоку в будь-яке місце файлу. Ці функції застосовуються при запису даних у файл та при їх читанні з файлу і мають відповідно вигляд:

in.seekp(n, dir);

in.seekg(n, dir);

де **in** — ім’я відповідного потоку (введення або виведення);

**n** — параметр, що вказує кількість байт, на яку треба перемістити покажчик потоку;

**dir** — необов’язковий параметр, що вказує на спосіб переміщення покажчика і приймає одне зі значень:

**ios::beg** — переміщення від початку файлу;

**ios::cur** — переміщення від поточної позиції;

**ios::end** — переміщення від кінця файлу.

Якщо параметр **dir** відсутній, то переміщення покажчика здійснюється з початку файлу.

Розглянемо програми, в яких використовуються методи прямого доступу до даних у файлі. (не встигли розглянути на попередній ЛР).

**Приклад**. Читання/запис масиву об’єктів у файл. Функції write(), read()

У прикладі використовуються функції write(), read() для роботи зі об’єктами класу BOOK, а саме:

* запис масиву об’єктів класу BOOK у файл;
* читання масиву об’єктів класу BOOK з файлу.

**#include <iostream>**

**#include <fstream>**

**#include <string.h>**

**#include <Windows.h>**

**using namespace std;**

**// Клас BOOK**

**class BOOK**

**{public:**

**char title[100]; // Назва книги**

**char author[70]; // Автор**

**int year; // Рік видання**

**float price; // вартість книги**

**};**

**// Запис масиву об’єктів в файл з допомогою функції write()**

**// читання масиву об’єктів з файлу з допомогою функції read()**

**bool Example7(const char \* filename)**

**{**

**// створити масив об’єктів**

**BOOK B[3] =**

**{**

**{ "Title-01", "Author-01", 2005, 100.95 },**

**{ "Title-02", "Author-02", 2008, 90.25 },**

**{ "Title-03", "Author-03", 2002, 180.50 }}**

**;**

**int n = 3; // кількість елементів у масиві**

**BOOK C[3]; // інший масив, з якого буде виконуватись читання**

**int n2; // кількість елементів у масиві C**

**int i;**

**// 1. Запис масиву об’єктів в файл**

**// outF - екземпляр файлу, в який здійснюється запис**

**ofstream outF(filename, ios::out | ios::binary);**

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

**// записати значення n**

**outF.write((char\*)&n, sizeof(int));**

**// запис масиву B[] в файл wf**

**for (i = 0; i < n; i++)**

**{**

**outF.write((char\*)&(B[i]), sizeof(BOOK));**

**}**

**cout << "Array is written\n" << endl;**

**// після закінчення роботи з файлом його потрібно закрити (обов'язково)**

**outF.close();**

**// 2. Читання масиву структур з файлу**

**// inF - екземпляр файлу, з якого здійснюється читання**

**ifstream inF(filename, ios::in | ios::binary);**

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

**cout << "Read the array...\n";**

**// Спочатку прочитати кількість записаних структур**

**inF.read((char\*)&n2, sizeof(int));**

**// цикл читання масиву об’єктів в змінну C**

**for (i = 0; i < n2; i++)**

**inF.read((char\*)&(C[i]), sizeof(BOOK));**

**inF.close(); // закрити файл**

**//**  **вивід масиву C на екран**

**cout << "Array C:" << endl;**

**for (i = 0; i < n2; i++)**

**{**

**cout << "Title = " << C[i].title << ", ";**

**cout << "Author = " << C[i].author << ", ";**

**cout << "Year = " << C[i].year << ", ";**

**cout << "Price = " << C[i].price << endl;**

**}**

**}**

**int main ()**

**{system("color F0");**

**string a="book.txt";**

**char c1[50]=" ";**

**strncpy(c1,a.c\_str(),a.length()+1);**

**if (Example7(c1))**

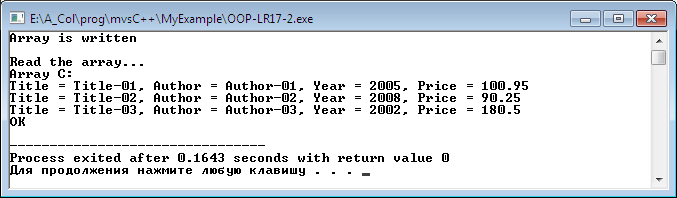
**{cout << "OK" << endl;**

**return 0;**

**}else return 1;**

**}**

Результат роботи



**Приклад.** Запису/читання масиву чисел типу double

Демонструється:

* запис у файл масиву **M** чисел типу double функцією write();
* читання з файлу масиву чисел типу double функцією read().

Файл відкривається у двійковому форматі.

**#include <iostream>>**

**#include <fstream>**

**using namespace std;**

**// використання функцій read(), write() для запису/читання масиву чисел**

**bool Example8(const char \* filename)**

**{**

**double M[] = { 2.44, 3.85, -3.23, 11.85, 3.38 }; // масив чисел**

**int i;**

**int n = 5; // кількість елементів у масиві M**

**// 1. Запис масиву в файл**

**// 1.1. Створити екземпляр outF, зв'язаний з файлом filename**

**ofstream outF(filename, ios::out | ios::binary); // для запису, двійковий формат**

**// 1.2. Перевірка, чи відкривається файл**

**if (!outF)**

**{**

**cout << "Error. Cannot open the file.";**

**return false;**

**}**

**// 1.3. Записати кількість елементів у масиві M**

**outF.write((char\*)&n, sizeof(int));**

**// 1.4. Записати увесь масив в файл**

**outF.write((char\*)&M, sizeof(double)\*n);**

**outF.close(); // закрити файл**

**// 2. Читання даних з файлу filename в масив M2**

**double M2[5];**

**int n2;**

**// 2.1. Відкрити файл для читання**

**ifstream inF(filename, ios::in | ios::binary);**

**// 2.2. Перевірка чи файл відкрито**

**if (!inF)**

**{**

**cout << "Error. Cannot open file.";**

**return false;**

**}**

**// 2.3. Зчитати к-сть елементів у масиві**

**inF.read((char\*)&n2, sizeof(int));**

**// 2.4. Зчитати дані з файлу в масив M2**

**inF.read((char\*)&M2, sizeof(double)\*n2);**

**inF.close(); // закрити файл**

**// 2.5. Вивести масив M2 на екран**

**cout << "Array M2:\n";**

**for (i = 0; i < n2; i++)**

**cout << M2[i] << " ";**

**cout << endl;**

**return true;**

**}**

**void main()**

**{**

**Example8("file8.bin");**

**}**

Результат роботи програми:

Array M2:

2.44 3.85 -3.23 11.85 3.38