## Лекція 14. Введення - виведення даних у мовах С та С++ Файли

**Поняття файлу**

Під час розв’язання задач на комп’ютері часто виникає необхідність у використанні даних, які записані на зовнішніх носіях інформації (дисках) і оформлені у вигляді файлів даних. Незалежно від того, які дані містять файли (числа, символи, рядки, масиви, структури тощо), в мові С++ вони трактуються як потоки даних **(stream),** які являють собою послідовність байтів, що зчитуються або записуються.

За замовчуванням у кожній програмі С++ можна користуватися такими стандартними потоками: стандартного введення **(сіn),** стандартного виведення **(cout)** та виведення помилок **(сеrr).** Щоб користуватися файлами, потоки повинні бути створені і закріплені за цими файлами.

Використання файлів даних у програмі передбачає виконання таких операцій:

* створення потоку обміну даними між файлом і пам’яттю комп’ютера;
* зв’язування цього потоку з конкретним ім’ям файлу на диску і відкриття файлу;
* запис даних у файл або читання їх з файлу;
* закриття файлу.

Для реалізації цих операцій існують спеціальні класи (структури даних разом з функціями обробки цих даних), які містять конструктори створення необхідних потоків:

* **ifstream** — для створення потоку читання даних;
* **ofstream** — для створення потоку запису даних у файл;
* **fstream** — використовується як для запису даних у файл, так і їх читання.

**Конструктори для роботи з файлами**

**Конструктори** — це спеціальні функції, які мають таке саме ім’я, що й ім’я класу. Вони записуються як з параметрами, так і без параметрів. Конструктори з параметрами одночасно створюють відповідний потік, зв’язують його з файлом на диску, відкривають файл для роботи і мають такі форми запису:

**ofstream іп (“іф”, ios::out); або ofstream (“іф”);**

**ifstream іп (“іф”, ios::in); або ifstream (” “);**

**fstream іп (“іф”, ios::in | ios::out);**

де **іп** — ім’я потоку, який створюється для роботи з файлом;

**іф** — константа або змінна типу **char[ ],** її значення — ім’я файлу на диску.

Перший з конструкторів використовується для запису даних у файл, другий — для читання даних з файлу, а третій — як для запису, так і для читання даних, наприклад:

**ofstream fout( "myfile.dat", ios::out);**

Цей запис створює потік з ім’ям **fout**, зв’язує його з файлом на диску, який має ім’я **myfile.dat** і відкриває цей файл для запису даних. Файл **myfile.dat** буде створено у тому ж каталозі, що і програма. Якщо треба створити файл у другому місці, то для запису його імені треба вказати шлях, наприклад, **a:\\pvp\\myfile.dat**. Тепер цей файл буде записано на диску **а:** в каталозі **pvp**.

Зверніть увагу на те, що для запису шляху треба використовувати подвійні зворотні косі риски.

Можна також для роботи з файлами застосувати конструктори без параметрів:

**ofstream іп;**

**ifstream іп;**

**fstream іп;**

де **іп** — ім’я відповідного потоку, тоді для зв’язування потоку з ім’ям файлу на диску і відкриття його для роботи треба додатково використовувати функцію-член відповідного класу, тобто:

**іn.ореn(“іф”, ios :: режим | ios :: режим);**

(в цьому запису  **|** означає "або", тобто можливість завдання декількох режимів).

Наприклад, відкриття файлу для запису до нього даних матиме вигляд:

**ofstream fout;**

**fout.open ( "a:\\pvp\\my file.dat", ios::out);**

Конструктори як з параметрами, так і без них, виконують однакову роботу, тому яким з них надати перевагу — справа користувача.

***Приклад***  Створити файл на диску і записати до нього масив чисел. Прочитати цей файл і вивести його компоненти на екран.

// Створення файлу та запис до нього масиву

**#include <iostream>**

**#include <fstream>**

**#include <conio.h>**

**#include <Windows.h>**

**using namespace std;**

**int main()**

**{ int i, mas[5];**

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

**// запис елементів масиву до файлу:**

**ofstream fout("massiv.txt"); /\* створення потоку fout та відкриття файлу з іменем massiv. txt для запису \*/**

**if (! fout) cout <<"Cannot open file\n";**

**else {**

**for (i = 0; i <5; i++)**

**{**

**cout << " Enter "<< i << " element\n";**

**cin >> mas[i]; // введення елементу масиву з клавіатури**

**fout << mas[i] << " "; //запис елементу до файлу**

**}**

**fout.close();**

**}**

**//---------- читання компонентів файлу та виведення на екран**

**ifstream fin("massiv.txt"); /\* створення потоку fin для читання файлу\*/**

**if (!fin) cout << " Cannot open file fo reading\n";**

**else {**

**cout <<"REZULTAT \n";**

**for (i = 0; i <5; i++)**

**{**

**fin >> mas[i];**

**// читання поточного елементу масиву з файлу**

**cout << "mas[" << i << "]=" << mas[i] << " ";**

**}**

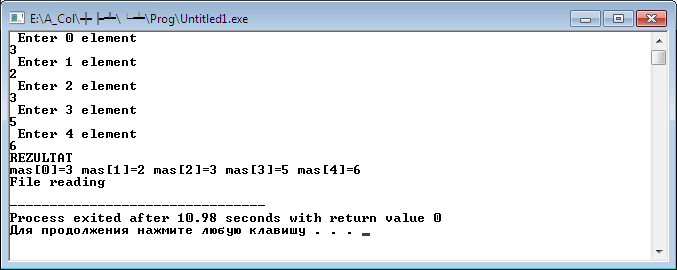
**cout << "\nFile reading\n";};**

**fin.close();**

**return 0;**

**}**

Результат розв’язання прикладу має вигляд:



Для додаткового керування даними, що виводяться, використовують маніпулятори **setw(w)** та **setprecision(d)**. Маніпулятор **setw(w)**призначений для зазначення довжини поля, що виділяється для виведення даних (**w** — кількість позицій). Маніпулятор **setprecision(d)** визначає кількість позицій у дробовій частині дійсних чисел.

Маніпулятори змінюють вигляд деяких змінних в об’єкті **cout**, що у потоці розташовані за ними. Ці маніпулятори називають прапорцями стану. Коли об’єкт посилає дані на екран, він перевіряє прапорці, щоб довідатися, як виконати завдання, наприклад, запис:

**cout << 456 << 789 << 123;**

призводить до виведення значення у вигляді: **456789123**, що ускладнює визначення групи значень.

**Приклад.** Написати програму, використовуючи маніпулятор **setw()**.

**#include <iostream.h>**

**#include <iomanip.h>**

**#include <conio.h>**

**main ( )**

**{**

**cout << 456 << 789 << 123 << endl;**

**cout << setw(5) << 456 << setw(5) << 789 << setw(5) << 123 << endl;**

**cout << setw(7) << 456 << setw(7) << 789 << setw(7) << 123 << endl;**

**float prod\_sum;** // prod\_sum — сумма продаж

**float nalog;**

//---------------  вывод подсказки для пользователя

cout << "Введите сумму продаж";

**cin >> prod\_sum;**

//............... вычисление налога на продажу

**nalog = prod\_sum\* 0.7;**

**cout << " " << setprecision(2) << prod\_sum;**

**cout << " " << setprecision(2) << nalog << "\n";**

**getch ();**

**}**

**Розглянемо конкретні приклади використання файлів**.

***Приклад 1.*** Записати у файл матрицю matr(2,4) поелементно за рядками, прочитати її з файлу і вивести на екран.

**// *Формування файлу з елементів матриці***

**#include <iostream>**

**#include <fstream>**

**#include <conio.h>**

**#include <Windows.h>**

**using namespace std;**

**int main()**

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

**int matr[2][4], i, j;**

**//---------------------------- запис матриці до файлу**

**// відкриття файлу**

**ofstream out("filemat.txt", ios::out | ios::binary);**

**cout << "Input matrix 2x4;" << endl;**

**for (i = 0; i < 2; i++)**

**{**

**for (j = 0; j < 4; j++)**

**{**

**cin >> matr[i][j]; // введення поточного елементу матриці**

**out << matr[i][j] << " "; //запис до файлу цього елементу**

**}**

**}**

**out.close();**

**//----------------------- виведення матриці на екрані**

**ifstream in("filemat.txt", ios::in | ios::binary); // відкриття файлу**

**for (i = 0; i < 2; i++)**

**for (j = 0; j < 4; j++)**

**in >> matr[i][j]; // читання з файлу елементу матриці**

**in.close();**

**//---------------------- виведення матриці на екрані**

**cout << "\nMatrix matr";**

**for (i = 0; i < 2; i++)**

**{**

**cout << endl;**

**for (j = 0; j<4; j++)**

**cout << matr[i][j] << " ";**

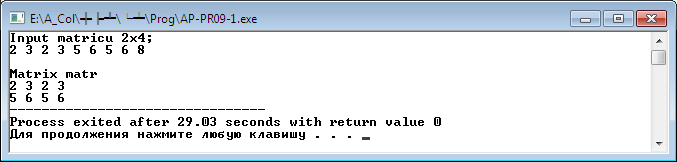
**}**

**getch();**

**return 0;**

**}**

**Результат розв’язання прикладу:**



У цій програмі спочатку елементи матриці з клавіатури вводились у пам’ять комп’ютера, потім кожен з них записувався у файл з ім’ям **filemat**. Для цього попередньо було створено потік **out** і відкрито файл на диску.

Потім було створено потік **in** для зчитування даних з файлу в пам’ять комп’ютера, тобто до матриці **matr[i][j].** Наприкінці програми матрицю виведено на екран.

**Зверніть увагу на опис файлів**.

Оголошення файлової змінної, яка визначає ім’я потоку, та відкриття текстового файлу для читання має наступний синтаксис:

**ifstream in(“filemat.txt”, ознака1|ознака2|…|ознакаN);**

У даному записі **in** – назва файлової змінної/ ім’я потоку, **filemat.txt** – назва файлу на диску. Ознаки слугують для визначення прав доступу до файлу. Перелік ознак відкриття файлу та їх значень наведено в табл. 1.

Оголошення файлової змінної та відкриття файлу для запису має синтаксис:

**ofstream out(“filemat.txt”, ознака1|ознака2|…|ознакаN);**

У даному записі **out** – назва файлової змінної, **filemat.txt** –  назва файлу на диску.

Якщо не використовувати ознаки відкриття файлу для читання чи запису, на диску буде створений новий файл із зазначеним ім'ям, або перезаписаний файл із таким же ім'ям, якщо він вже існує.

Наприклад, для того, щоб записати потрібну інформацію в кінець файлу text1.dat, необхідно скористатися фрагментом програмного коду (тут файлова змінна flags):

ofstream flags(“text1.dat”, ios::app);

Таблиця 1 – Ознаки відкриття файлу

| **Ознака** | **Призначення** |
| --- | --- |
| ios::in | Відкриває файл для читання. Вміст файлу зберігається |
| ios::out | Відкриває файл для запису. Якщо файл не існує, то буде створений |
| ios::app | Відкриває файл для дозапису. Дані будуть записані у кінець файлу |
| ios::trunk | Якщо  файл, який відкривають для запису, вже існує, то його вміст буде видалено. |
| ios::nocreate | Забороняє створювати файл, який відкривають |
| ios::inoreplace | Забороняє перезаписувати існуючий  файл |

Файли класифікують за типом компонентів і за методом доступу до них. За типом компонентів розрізняють текстові та бінарні (двійкові) файли, а за методом доступу – файли послідовного і прямого доступу. Ми будемо розглядати файли послідовного доступу. Текстові файли призначені для збереження текстів (наприклад, текстів програм), а бінарні файли використовуються для збереження даних різних типів. Файл бінарний – це лінійна послідовність байтів, що відповідає внутрішньому поданню даних без поділу на рядки. Для завдання бінарного файлу використовується ознака ios:binary (двійковий режим). Якщо задається декілька ознак (як в прикладі), то вони розділяються вертикальною рискою ( | ).

***Приклад 2.*** Записати у файл 5 прізвищ, потім прочитати їх і вивести на екран.

**/\* *програма запису до файлу та читання з файлу масиву з 5 прізвищ* \*/**

**#include <iostream>**

**#include <fstream>**

**#include <conio.h>**

**#include <Windows.h>**

**using namespace std;**

**int main()**

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

**char st[5][15];**

**int i;//------------------ запис до файлу**

**ofstream fout("st\_file.dat"); // відкриття файлу**

**if (!fout) (cout << "Cannot open file\n";return 1;);**

**for (i = 0; i <5; i++)**

**{**

**cout << " Enter " << (i+1) << " name\n";**

**cin.getline(st[i],15); // введення поточного прізвища**

**fout << st[i] << '\n'; // запис прізвища до файлу**

**}**

**fout.close();//-------читання файлу та виведення на екран**

**cout << "\nReading file\n\n";**

**ifstream fin("st\_file.dat");**

**if (!fin)(cout << "Cannot open file.dat\n";return 2;);**

**for (i = 0; i < 5; i++)**

**{**

**fin.getline(st[i],15);**

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

**}**

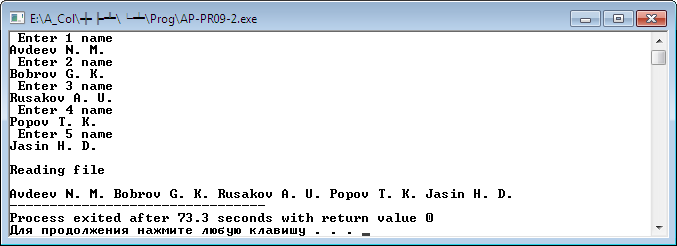
**fin.close();**

**getch();**

**return 0;**

**}**

Результат роботи програми має вигляд:

**.**

У попередніх програмах запис даних у файл та їх читання з файлу здійснювалось послідовно поелементно. Але записати або прочитати декілька даних (наприклад, масив чисел) можна однією операцією. Для цього використовують функції-члени відповідних класів, які мають вигляд:

**in.write((char\*)&p,sizeof(p));** — для запису даних у файл,

**in.read((char\*)&p,sizeof(p));** — для читання даних з файлу,

де **іn** — ім’я потоку введення або виведення;

**р** — змінна будь-якого типу, якщо змінна **р** має тип **char[ ],** то операція її приведення не потрібна.

Коли нам потрібно записати одну символьну змінну ми використовуємо одновимірний масив типу **char[ ]**. Але коли нам потрібно записати масив символьних змінних ми використовуємо двовимірний масив, наприклад,

**char str [5][20] = {“Петренко И. И. “Головко С. С. “, . . . ,};**

де перший індекс визначає номер змінної, а другий максимальну довжину змінної. Функція **getline** працює з типом **char[ ],** саме тому ми так описуємо масив прізвищ.

*Ще одне зауваження* стосується перевірки доступності файлу, це вирази **(!fout)**та**(!fin). Якщо файл відкрити неможливо, то fout та fin будуть мати значення false**.

***Приклад 3.*** Розробити програму, за допомогою якої здійснюється запис даних типу структура (список прізвищ абонентів та їх телефонів) у файл з ім’ям **struct.txt**.

/\* *програма запису до файлу* struct.txt *даних типу структура — списку прізвищ абонентів та їх номерів телефонів* \*/

**#include <iostream>**

**#include <fstream>**

**#include <string>**

**#include <conio.h>**

**#include <Windows.h>**

**using namespace std;**

**struct telefon**

**{char fio[15]; char tel[10];};**

**main()**

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

**int i;**

**telefon spis[5];**

**ofstream out("struct.txt ");**

**if (!out) (cout << "Cannot open file\n";return 1;);**

**for (i = 0; i < 5; i++) {**

**cout << "Enter " << i+1 << " last name and phone number\n";**

**//------------ Введення прізвища і телефону з клавіатури**

**cin >> spis[i].fio;**

**cin >> spis[i].tel;**

**//------------ Виведення прізвища і телефону до файлу**

**out << spis[i].fio << " " << spis[i].tel << endl;**

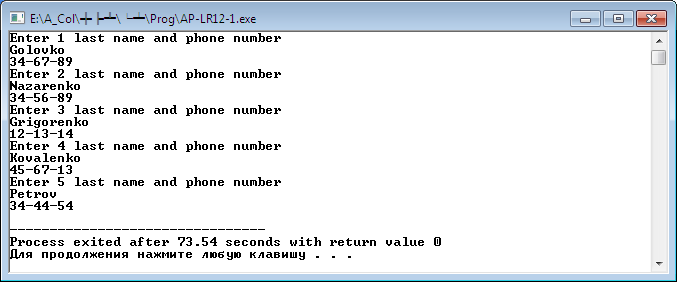
**}**

**out.close();**

**getch ();**

**}**

Результат роботи цієї програми такий:



***Приклад 4.*** Розробити програму читання файлу (**struct.txt**), створеного у прикладі 3, і виведення на екран за запитом користувача або списку прізвищ абонентів і їх телефонів, або тільки прізвища і номера телефону потрібного абонента.

/\* *програма читання з файлу даних типу структура — список прізвищ абонентів та номерів телефонів* \*/

**#include <iostream>**

**#include <fstream>**

**#include <string>**

**#include <conio.h>**

**#include <Windows.h>**

**using namespace std;**

**struct telefon**

**{char fio[15]; char tel[10];};**

**main()**

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

**int i, p;**

**char name[15];**

**bool t;**

**telefon spis;**

**//------------------ відкриття раніше створеного файлу**

**ifstream in("struct.txt");**

**if (!in) (cout << "\nCannot open file fo reading\n";**

**return 1););**

**//------------------ завдання режиму роботи з файлом**

**cout << "Who make: reading list(1) or name(2)\n";**

**cin >> p; //---введення номеру режиму**

**if (p == 1) //---оброблення 1 режиму**

**{**

**//------------------ читання даних з файлу "struct.txt"**

**while (in >> spis.fio >> spis.tel)**

**//------------------ виведення даних на екран**

**cout << spis.fio << " " << spis.tel << endl;**

**}**

**else if (p == 2) //---оброблення 2 режиму**

**{**

**t = true;**

**cout << "Enter name\n";**

**cin >> name; //---введення прізвища**

**//------------------------цикл для читання даних з файлу**

**while (in >> spis.fio >> spis.tel)**

**if (strcmp(spis.fio,name) == 0)**

**{**

**//-----------------------виведення даних на екран**

**cout << spis.fio << " "<<spis.tel<<"\n";**

**t = false;**

**}**

**in.close();**

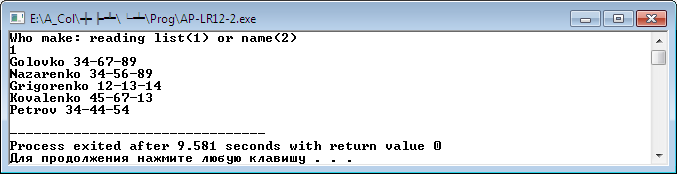
**if (t) cout <<"Name "<< name <<" is not exist\n";**

**}**

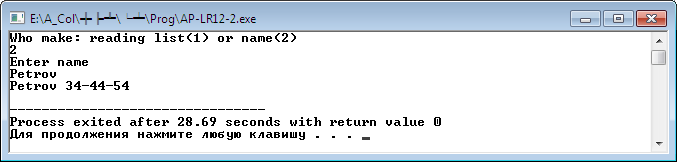
**getch();**

**}**

Результат розв’язання програми для першого режиму її роботи, коли потрібно вивести усі дані з файлу:



Результат розв’язання цієї програми для другого режиму роботи, коли необхідно вивести задане прізвище і телефон, має вигляд:



**Приклад 5. Читання/запису масиву структур у файл. Функції write(), read()**

У прикладі використовуються функції write(), read() для роботи зі структурою типу BOOK, а саме:

* запис масиву типу BOOK у файл, що складається з трьох структурних змінних;
* читання масиву структур типу BOOK з файлу.

**#include <iostream>**

**#include <fstream>**

**#include <string>**

**#include <Windows.h>**

**using namespace std;**

**// Структура BOOK**

**struct BOOK**

**{**

**char title[100]; // назва книги**

**char author[70]; // автор**

**int year; // рік випуску**

**float price; // вартість книги**

**};**

**// Запис масиву структур в файл з допомогою функції write()**

**// читання масиву структур з файлу з допомогою функції read()**

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

**{**

**// продемонстровано запис в файл та читання з цього файлу масиву структур типу BOOK**

**// створити масив структур**

**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 B**

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

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

**int i;**

**// 1. Запис масив структур 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 структур 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;**

**}**

**}**

**void main()**

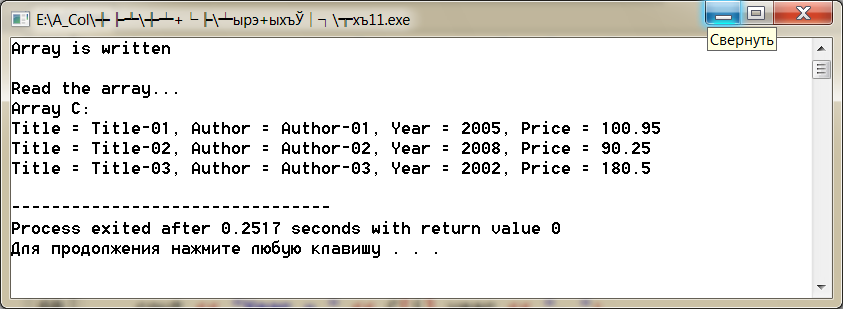
**{ system("color F0");**

**Example7("file7.bin");**

**return 0;**

**}**

Результат роботи програми:



*Для самостійного вивчення (10 годин)*: Вивчення лекційного матеріалу та додаткових джерел. Розгляд запитань і виконання завдань для самостійної роботи, запропонованих на лекції.

*Література*

1. Ковалюк Т. В. Алгоритмізація та програмування: Підручник. — Львів: «Магнолія 2006», 2013. — 400 с., ил.
2. Єжова Л. Ф. Алгоритмізація і програмування процедур обробки інформації: Навч.-метод. посіб. для самост. вивч. дисц. К.: КНЕУ, 2000.
3. Вступ до програмування мовою С++. Організація обчислень: навч. посіб. / Ю. А. Бєлов, Т. О. Карнаух, Ю. В. Коваль, А. Б. Ставовський. – К. : Видавничо-поліграфічний центр "Київський університет", 2012. – 175 с. с.: іл. ISBN (укр.) . URL: <http://csc.knu.ua/uk/library/books/belov-24.pdf>
4. Джейс Либерти Освой самостоятельно С++ за 21 день: 3-е изд. пер. с англ.: Уч. пос. – М.: Издательский дом „Вильямс”, 2001. – 816 с.: ил..

*Контрольні запитання для самоперевірки*.

1. Які види потоків ви знаєте?
2. Для чого призначені стандартні потоки?
3. Який файл необхідно включати у програму для використання стандартних потоків?
4. Назвіть класи вхідних та вихідних потоків.
5. Який клас є базовим для потоків?
6. Для чого призначені маніпулятори потоків, і який файл необхідно включати у програму для їх використання?
7. Для чого призначені маніпулятори endl, setprecision, setw?
8. Які є файли за способом доступу?
9. Назвіть класи для створення файлових потоків і їх призначення.
10. Які дії необхідно виконати для використання файлів у програмі?
11. Які існують режими відкриття файлів?
12. Для чого призначений метод open() і які він має параметри?
13. Які способи відкриття та закриття файлу ви знаєте?