**Практична робота № 14 Розробка програм з використанням файлів для введення/виведення даних**

**Мета**: навчитися писати програми **на мові С++** в **консольному режимі** з використанням файлів, рядкових величин та структур**.**

**Методичні вказівки щодо організації самостійної роботи студентів**

1. Повторити Лекції 10 – 14, теоретичні відомості з Лабораторної роботи №11.
2. Запустити середовище програмування С++ .
3. Записати програму, що виконує завдання з пп.4. В першому рядку програми записати

*// Група № Прізвище*

вказавши номер своєї групи та своє прізвище.

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

1. **Завдання**:

Модифікуємо програму з Лабораторної роботи №11. В ній у вас було:

**Завдання до Лабораторної роботи №11** :

Написати програму, в яка містить структуру, що описує об'єкт будинок і складається зі змінних, в яких знаходиться інформація про місто (в якому будинок знаходиться), вулиця, номер будинку, кількість квартир, інтернет (проведено чи ні, задати логічною/булевою змінною).

1. Сформувати масив структур (не менше 3), в якому номер будинку, кількість квартир заповнюється випадковими числами з діапазону від 1 до 20 та від 100 до 200, відповідно (див. ЛР№11), іншим елементам присвоюються значення в програмі.
2. В програмі запитати кількість об’єктів, про які потрібно вивести інформацію, та вивести перші із запитаної кількості елементи масиву структур з вказівкою індексу масиву, а після нього вся інформація про об’єкт. Значення, що виводяться по об’єкту, розділяються їх табуляцією.
3. Після цього підрахувати кількість будинків без інтернету та вивести їх кількість, після цього вивести заголовок "Будівлі з інтернетом" і вивести порядковий номер в списку, місто, вулиця, номер будинку.

**Завдання до Лабораторної роботи №14.**

В програму з Лабораторної роботи №11 внести такі зміни:

1. Описати файли для виведення та читання даних.
2. Відкрити файл для виведення даних та записати до нього інформацію, яка формується у п. а).
3. Закрити файл для виведення даних.
4. Відкрити сформований файл для читання даних та оброблювати прочитані дані за п. b). Примітка. Дані ви можете читати в нову окрему структуру, якщо ви використовували масив структур в п. і.
5. Закрити файл для читання даних, а потім знов відкрити і виконати дії за п. с)
6. Закрити файл для читання даних

Для кращого розуміння процесу, що будете програмувати, доцільно намалювати блок-схему.

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

Файл повинен мати назву в такому форматі:

**ОП+АМ <Номер групи><Номер лабораторної><Прізвище англійською>**

Наприклад, 21-01Ivanov.cpp.

**Тему в заголовку листа записати**

**ОП+АМ <Номер групи>-><Номер лекції / практичної / лабораторної [літера позначення типу роботи L – лекція, P – практична, R – лабораторна]<Прізвище англійською>**

Всі запитання, що виникнуть, надсилайте на електронну адресу викладача, тему в заголовку листа записати

Всі запитання, що виникнуть, надсилайте на електронну адресу викладача, тему в заголовку листа записати

**ОП+АМ-Запитання-<Номер групи>-<Прізвище >**.

**Строк відсилки ЛР для ІПЗ-21 10.11.2022**

**ІПЗ-22 09.11.2022**

***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?

**ТЕОРЕТИЧНІ ВІДОМОСТІ**

**Пригадаємо**

**Поняття файлу**

**Файл** (file — шухляда, тека, папка) — інформаційний об'єкт, що містить дані або програми і розміщується на поіменованій ділянці носія даних, сутність, елемент (одиниця носія інформації; англ. media unit), що дозволяє отримати доступ до певного ресурсу обчислювальної системи і має такі ознаки:

* фіксована назва (назва файлу — послідовність символів (англ. string), число чи щось інше, що однозначно характеризує файл);
* певну логічну будову (структуру) і відповідні йому операції читання/запису.

На практиці *це іменований блок інформації, який зберігається на носії інформації.*

* Файл обов'язково має назву і може мати будь-який розмір інформації (максимальна довжина назви та розміру файлу обмежується властивостями конкретної файлової системи).
* **Файл - це впорядкована сукупність даних, що зберігається на диску і займає іменовану область зовнішньої пам'яті.**
* Файл - це довільний блок інформації, або пристрій вводу-виводу, асоційований із ним (як середовище, засіб передачі "довільних блоків інформації").
* Кожна комп'ютерна програма відкриває принаймні три файли стандартних потоків:
  + вхідний файл (stdin) - асоціюється із клавіатурою,
  + вихідний файл (stdout) - із дисплеєм терміналу,
  + файл виводу повідомлень про помилки (stderr) - із дисплеєм терміналу.
* Файлом може бути також ділянка оперативної пам'яті програми.

**Структура файлу**

Файли можна умовно поділяти на файли *простої та складної структури* (хоча точка зору на структуру файлу залежить від тієї програми, яка його обробляє).

Файли простої структури складаються з послідовності записів (records) - елементарних одиниць, в термінах яких виконуються операції обміну з файлом. Записи можуть бути:

* рядками, якщо це текстовий файл;
* двійковими даними фіксованої довжини;
* двійковими даними змінної довжини.

*Файли складної структури можуть бути самого різного виду.*

Складна структура файлу може бути змодельована записами шляхом додавання відповідних керуючих символів.

*Файли інтерпретуються* операційною системою або програмами їх обробки.

**Атрибути файлів**

* **Ім'я** (Name) - назва файлу в символьній формі, сприймається користувачем.
* **Тип** (Type) - тип збереженої у файлі інформації. Окремий атрибут тип необхідний для систем, які підтримують різні типи файлів. Наприклад, в системі Ельбрус значенням атрибута тип файлу є число, кодує тип: 0 - дані, 2 - код, 3 - текст і т.д. Однак більш загальноприйнятим підходом є підхід, прийнятий в системах MS DOS, Windows, UNIX: тип файлу кодується розширенням імені, наприклад, book.txt- текстовий файл (.txt), що містить текст книги.
* **Розміщення** (Location) - покажчик на розміщення файлу на пристрої.
* **Розмір** (Size) - поточний розмір файлу.
* **Захист** (Protection) - керуюча інформація, що задає повноваження читання, зміни і виконання файлу.
* **Час** і **дата**. Наприклад, у всіх системах зберігається дата створення файлу і дата останньої модифікації файлу. Остання відіграє важливу роль при компіляції (збірці) великих програмних проектів, так як утиліти для збірки проектів (наприклад, make) визначають по співвідношенню дат останньої модифікації файлів вихідного коду і двійкового коду, чи слід перекомпілювати вихідний файл.

**Операції над файлами**

Хоч набір операцій над файлами і особливо їх позначень відрізняється від системи до системи, можна виділити наступні основні операції над файлами.

**Створення файлу** (Create). Створюється заголовок файлу; спочатку його вміст (пам'ять) порожньо.

**Запис в файл** (Write). Як правило, відбувається записами (records) або блоками - більшими логічними одиницями інформації, що об'єднують кілька записів, з метою оптимізації операцій введення-виведення.

**Читання з файлу** (Read). Зазвичай також виконується записами або блоками.

**Пошук позиції всередині файлу** (позиціонування) (Seek). Позиція задається номером запису або блоку, або спеціальними іменами, які позначають початок файлу (позиція перед першим записом) або кінець файлу (позиція після останнього запису).

**Видалення файлу** (Delete). Залежно від реалізації системи файлів, помилкове видалення файлу може бути фатальним (UNIX) або виправних (MS DOS).

**Скорочення файлу** (Truncate).

**Відкриття файлу** (Open) - пошук файлу в структурі директорій по його символьному імені (шляху) і зчитування його заголовка і одного або декількох суміжних блоків в буфера в основний пам'яті.

**Закриття файлу** (Close) - запис вмісту буферів в блоки файлу; оновлення файлу у зовнішній пам'яті відповідно до його поточним станом; звільнення всіх структур в основний пам'яті, пов'язаних з файлом.

Для виконання операцій обміну з файлом (read, write), як правило, файл необхідно відкрити. Закриття файлу є обов'язком користувача процесу; однак, якщо він з якоїсь причини цього не виконує, то закриває всі файли, відкриті процесом, операційна система після завершення або припинення процесу.

**Типи файлів - імена та розширення**

| **тип файлу** | **розширення імені** | **Функціональність** |
| --- | --- | --- |
| виконуваний код (завантажувальний модуль) | exe, com, bin або відсутній | готова до виконання програма в бінарному машинному коді |
| об'єктний модуль | obj, о | відкомпільоване програма в бінарному коді |
| вихідний код на мові програмування | срр, h, с, сс, Java, pas, asm, а | вихідний код на різних мовах (Сі, Паскаль і ін.) |
| командний файл | bat, sh | файл з командами для командного інтерпретатора |
| текст | txt, doc | текстові дані, документи |
| документ для текстового процесора | wp, tex, rtf, doc | документ в форматі будь-якого текстового процесора |
| бібліотека | lib, a, so, dll, mpeg, mov, rm | бібліотеки модулів для програмування |
| файл для друку або візуалізації | arc, zip, tar | ASCII або бінарний файл у форматі для друку або візуалізації |
| архів | arc, zip, tar | кілька файлів, згрупованих в один файл, для архівації або зберігання |
| мультимедіа | mpeg, mov, rm | бінарний файл, який містить аудіо- або відео інформацію |

**Методи доступу до файлів**

Традиційно розрізняються файли послідовного та прямого доступу.

**Файл послідовного доступу** - це файл, доступ до якого можливий тільки позиціонуванням на початок і кінець і потім операціями обміну виду вважаєте оновити наступну (попередню) запис.

**Файл прямого доступу** - це файл, для якого можливий безпосередній доступ по номеру запису і операція обміну з явним зазначенням номера запису. (при виконанні обміну з файлом завжди існує деяка поточна позиція по файлу, яка вказує на деякий запис, на позицію перед початком або після кінця файлу)

В операціях над файлом послідовного доступу довільна установка позиції *не допускається*, а дозволені тільки операції, автоматично пересувають поточну позицію на наступну (попередню) запис.

Подібна особливість пов'язана з різницею пристроїв, на яких розміщені файли (наприклад, магнітна стрічка - по суті справи, послідовне пристрій), проте необхідність організації послідовних або прямих файлів може бути пов'язана з суттю завдання.

Мабуть, послідовний доступ використовується частіше: саме так відбувається введення даних, виведення результатів на друк або на екран.

**Текстовий файл**

Те́кстовий файл — форма подання послідовності символів у комп'ютері, де кожен символ із задіяного набору символів кодується одним байтом чи послідовністю двох, трьох і т. д. байтів.

На відміну від терміна «текстовий формат», що характеризує вміст даних, термін «текстовий файл» стосується файлу та характеризує його як контейнер, який зберігає такі дані.

**Текстовий файл** — послідовність символів (переважно друкованих знаків, що належать тому чи іншому набору символів). Ці символи зазвичай згруповані в рядки (англ. lines, rows).

Іноді кінець текстового файлу (особливо тоді, коли в файловій системі не зберігається інформація про розмір файлу) також позначається спеціальними знаками (одним або більше), відомими як маркери кінця файлу.

Текстовий файл може містити як форматований, так і неформатований текст.

Текстовим файлам протиставляються двійкові (бінарні) файли, в яких інформація організована за іншими принципами (вона містить інформацію, не прив'язану до набору символів).

**Текстовий файл:Переваги та недоліки**

* **Переваги**
* *Універсальність* — текстовий файл може бути прочитаний (так чи інакше) на будь-якій системі або ОС, особливо, якщо йдеться про однобайтові кодування на кшталт ASCII, які не схильні до проблеми, характерної для інших форматів файлів — для них не важлива різниця в порядку байтів або довжині машинного слова на різних платформах.
* *Стійкість* — кожне слово та символ у такому файлі самодостатні і, якщо трапиться пошкодження байтів у такому файлі, то зазвичай можна відновити дані за контекстом або продовжити обробку решти вмісту, в той час як у стиснених чи двійкових файлів пошкодження декількох байтів може зробити файл абсолютно невідновним. Багато систем управління версіями розраховані на текстові файли і з двійковими файлами можуть працювати лише як з єдиним цілим.
* *Формат текстового файлу вкрай простий і його можна змінювати текстовим редактором — програмою, яка входить в комплект практично будь-якої ОС.*
* **Недоліки**
* У великих нестиснутих текстових файлів *низька інформаційна ентропія* — ці файли займають більше місця, ніж мінімально необхідно. Хоча ця ж надмірність інформації визначає підвищену стійкість до збоїв у каналах передачі даних і при отриманні даних з носіїв, наприклад, з магнітної стрічки.
* *Деякі операції з текстовими файлами неефективні.*
* Наприклад, якщо в файлі зустрінеться число, обчислювальна система до початку операцій з ним повинна буде перетворити його в свій внутрішній формат, застосувавши порівняно складну процедуру конвертації числа; щоб перейти на 1000-ий рядок, потрібно порахувати попередні 999 рядків; складно замінити один рядок іншим, тощо. Тому при роботі з великими обсягами даних текстові файли застосовують лише як проміжний формат, що забезпечує інтероперабельність.

**Формати, засновані на текстових файлах**

В силу своєї простоти текстові файли нерідко використовуються для *зберігання службової інформації* (наприклад, логів): оскільки операція додавання в кінець текстового файлу нових даних не вимагає значних обчислювальних ресурсів (незалежно від уже наявного обсягу файлу і виду текстових даних, що додаються), ведення текстових лог-файлів зазвичай відбувається ефективно та непомітно для користувача і для інших додатків (аж до вичерпання дискового простору).

Текстовий формат служить основою для багатьох спеціалізованих форматів (наприклад, .ini, SGML, HTML, XML, TeX, вихідних текстів мов програмування).

В текстовому файлі текст може зберігатися як в неформатованому, так і в форматованому або розміченому вигляді (наприклад, Rich Text Format, HTML), де кожен символ чи група символів (рядки, абзаци, таблиці тощо) може бути відформатований (визначений шрифт, накреслення, розмір і т. д.).

**Розширення імен файлів**

В DOS і Windows для файлів з неформатованим текстом зазвичай використовується розширення .txt. Проте, текстовими можуть бути файли з будь-яким іншим розширенням або й без нього. Наприклад, вихідні коди програм зазвичай зберігаються в файлах з розширеннями, відповідними мові програмування, якою вони написані (.срр, .bas, .pas, .c тощо).

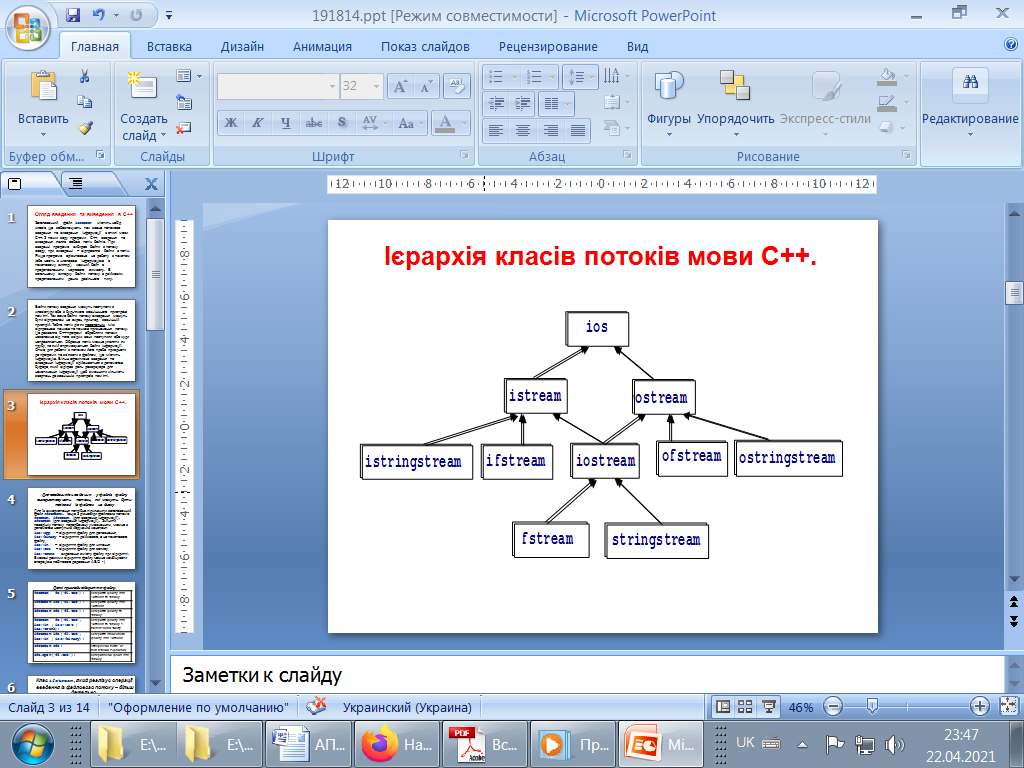
Форматований текст (текст із розміткою) зазвичай зберігається у файлах з розширенням, відповідним формату або мові розмітки — .rtf, .htm, .html тощо.

**Огляд введення та виведення в С++**

Заголовочний файл iostream містить набір класів, що забезпечують так зване потокове введення та виведення інформації в стилі мови С++. З точки зору програми С++, введення та виведення являє собою потік байтів. При введенні програма вибирає байти з потоку вводу, при виведенні – відправляє байти в потік. Якщо програма орієнтована на роботу з текстом (або навіть з числовою інформацією в текстовому вигляді), кожний байт є представленням чергового символу. В загальному випадку байти потоку є двійковим представленням даних довільного типу.

Байти потоку введення можуть поступати з клавіатури або з будь-якого зовнішнього пристрою пам'яті. Так само байти потоку виведення можуть бути відправлені на екран, принтер, зовнішній пристрій. Тобто потік діє як посередник між відправною точкою та точкою призначення потоку. Це дозволяє С++-програмі обробляти потоки незалежно від того, звідки вони поступили або куди направляються. Образно потік можна уявляти як трубу, по якій спрямовуються байти інформації. Отже, для роботи з потоком його треба приєднати до програми та зв'язати з файлом, що містить інформацію. Більш ефективно введення та виведення інформації здійснюється з допомогою буфера, який відіграє роль резервуара для накопичення інформації щоб зменшити кількість звертань до зовнішніх пристроїв пам'яті.

**Ієрархія класів потоків** **мови С++.**

****

*Для введення/виведення у файл/з файлу використовують потоки, які можуть бути пов'язані із файлом на диску*

Для їх використання потрібно підключити заголовковий файл <**fstream**>. Існує 3 різновиди файлових потоків: fstream, ifstream (для введення інформації), ofstream (для введення інформації). Змінити поведінку потоку, передбачену умовчанням, можна з допомогою наступних керуючих констант:

ios::app     – відкриття файлу для доповнення;

ios::binary  – відкриття двійкового, а не текстового файлу;

ios::in       – відкриття файлу для читання;

ios::out     – відкриття файлу для запису;

ios::trunc   – видалення змісту файлу при відкритті.

Вказані режими відкриття файлу можна комбінувати операцією побітового додавання АБО - |

***Деякі приклади відкриття файлу***

|  |  |
| --- | --- |
| fstream fs("f1.txt"); | Відкриття файлу для читання та запису |
| ifstream ifs("f2.txt"); | Відкриття файлу для читання |
| ofstream ofs("f3.txt"); | Відкриття файлу та запису |
| fstream fs("f1.txt", ios::in | ios::out | ios::trunk); | Відкриття файлу для читання та запису з видаленням змісту |
| ifstream ifs("f2.txt", ios::in | ios::binary); | Відкриття двійкового файлу для читання |
| ofstream ofs; | Створюємо потік, не пов’язаний із файлом |
| ofs.open("f3.txt"); | Відкриваємо файл для запису |

**Прості операції з файлами введення**

Припустимо, що програма має читати інформацію із файлу. Для цього необхідно виконати наступні кроки:

1. Створити екземпляр класу ifstream, для керування потоком введення:

ifstream ifile;

1. Поставити цей екземпляр у відповідність конкретному файлу:

ifile.open (“D:\\in.txt”);

(Кроки 1 та 2 можна замінити створенням екземпляру з допомогою конкретного конструктора:

ifstream ifile (“D:\\in.txt”, ios::in);)

1. Працювати із цим об'єктом так само, як із об'єктом cin.

**Прості операції з файлами виведення.**

Припустимо, що програма має виводити інформацію у файл. Для цього необхідно виконати наступні кроки:

1. Створити екземпляр класу ofstream, для керування потоком виведення:

ofstream ofile;

1. Поставити цей екземпляр у відповідність конкретному файлу:

ofile.open (“D:\\in.txt”);

(Кроки 1 та 2 можна замінити створенням екземпляру з допомогою конкретного конструктора:

ofstream ofile (“D:\\in.txt”, ios::out);)

1. Працювати із цим об'єктом так само, як із об'єктом cout.

***Приклад 1.*** Розробити програму, за допомогою якої здійснюється запис даних типу структура (список прізвищ абонентів та їх телефонів) у файл з ім’ям **struct.txt**.

/\* *Р10\_5.СРР1 — 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";**

**else {**

**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;**

**}**

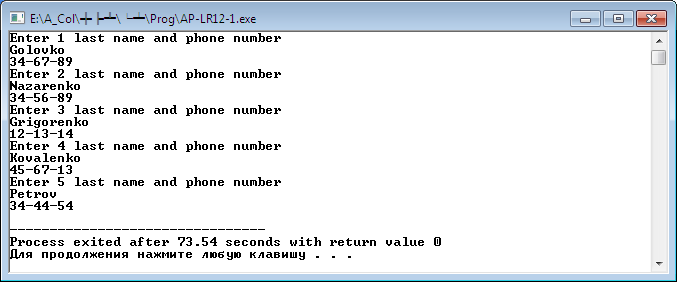
**out.close();**

**}**

**getch ();**

**}**

Результат роботи цієї програми такий:



***Приклад 2.*** Розробити програму читання файлу (**struct.txt**), створеного у прикладі1, і виведення на екран за запитом користувача або списку прізвищ абонентів і їх телефонів, або тільки прізвища і номера телефону потрібного абонента.

/\* *Р10\_6.СРР\_1 — 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";**

**else {**

**//-----------------------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\n";**

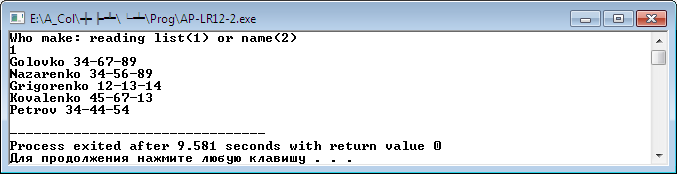
**}**

**}**

**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:

