ПЛАН

КЛАСИ ПОТОКІВ

Потоки даних

Файлові типи даних введені в мови програмування для роботи із зовнішніми пристроями - файлами на диску, принтерами і т.ін. Передача даних із зовнішнього пристрою в оперативну пам'ять називається *читанням* або *введенням*, зворотний процес - *записом* або *виведенням*.

У загальному випадку, введення даних у програму може здійснюватися з різних пристроїв - консолі (клавіатури), різноманітних накопичувачів; виведення даних - на консоль (монітор), принтер, дискові накопичувачі тощо. Тобто, як джерелами, так і приймачами інформації при такому обміні можуть бути пристрої, які на апаратному рівні суттєво відрізняються один від одного.

Для забезпечення єдиного інтерфейсу, що не залежить від того, до якого конкретного пристрою звертається програма, файлові системи розвинених мов програмування розглядають всі зовнішні пристрої як єдиний абстрактний логічний пристрій, а уніфікований логічний інтерфейс з такими пристроями називають *потоком* (англ. *stream*). Такий потік визначається як неперервна послідовність даних, передача яких не залежить від конкретного пристрою, з яким відбувається обмін інформацією.

Потоки даних схожі своєю поведінкою. Оскільки вони не залежать від пристроїв введення / виведення, то введення з клавіатури можна трактувати так же, як і введення з файла. Аналогічно, використовуючи потоки, програма може обробляти вивід незалежно від того, куди він направляється. Така уніфікація дає змогу використовувати одні й ті ж функції як для обміну даними з файлами, так і для обміну з різноманітними пристроями.

Слід зауважити, що передавання даних здійснюється не безпосередньо між зовнішнім пристроєм і програмою, а через так званий *буфер* - спеціально виділену область оперативної пам'яті. Буфер виділяється для кожного потоку (файлу) і служить для тимчасового зберігання даних під час їхньої передачі від джерела до приймача інформації.

При читанні з потоку (з фізичного файлу) дані спочатку зчитуються у буфер, причому даних зчитується не стільки, скільки запитується, а скільки поміститься в буфер. Розмір буфера визначається ОС і є кратним довжині фізичного сектора (512 байт). Під час запису у потік (у фізичний файл) вся інформація спочатку

спрямовується у буфер і там накопичується доти, поки увесь обсяг буфера не буде заповнений. Тільки після цього або після спеціальної команди скидання даних відбувається передача даних на зовнішній пристрій.

Механізм буферізації дозволяє швидше і ефективніше обмінюватися інформацією із зовнішніми пристроями.

Операції введення-виведення можуть бути представлені у двох форматах: *текстовому* (**text**) та *двійковому* (**binary**). Структура і використання інформації в цих потоках є різною.

Текстовий потік - це послідовність символів. Особливістю текстового потоку є те, що при його передачі можуть відбуватися певні перетворення символів. Наприклад, символ нового рядка може бути замінений парою символів - «повернення каретки» та «переведення рядка». При передачі символів з потоку на екран, частина з них не виводиться (символи «повернення каретки» та «переведення рядка»).

Тому між символами, які пишуться в потік (читаються із потоку) і символами, які зберігаються на зовнішньому пристрої, може і не бути однозначної відповідності. Окрім того, через можливі перетворення кількість символів, прочитаних або записаних, може не збігатися з кількістю символів, які зберігаються на зовнішньому пристрої.

Двійковий потік - це послідовність байтів, які взаємнооднозначно відповідають байтам на зовнішньому пристрої. Тобто, при передачі двійкового потоку ніякого перетворення символів не відбувається. Окрім того, кількість байтів, які пишуться (читаються), співпадає із числом байтів, які зберігаються на зовнішньому пристрої.

За типом пристроїв, з якими працюють потоки даних, вони умовно поділяються на *стандартні* та *файлові*.

Стандартні потоки використовуються для виведення інформації на екран та зчитування її з клавіатури. Стандартні потоки відкриваються автоматично при кожному запуску програми.

У мові С визначені наступні стандартні потоки:

stdin - стандартне консольне введення (клавіатура за замовчуванням);
stdout - стандартний консольний вивід (монітор за умовчанням);
stderr - стандартне виведення помилок (діагностики та повідомлень налаштування в текстовому вигляді).

Щоб мати можливість використовувати ці потоки, необхідно підключити заголовний файл <**stdio.h**>.

В С++ механізм потоків оновлено і змінено: потоки повністю реалізовані через класи. Проте загальний принцип під'єднання потоків залишився. Коли запускається програма на С++, автоматично створюються чотири стандартних потоки:

cin - асоціюється із стандартним пристроєм вводу (зазвичай, клавіатурою);
cout - асоціюється із стандартним пристроєм виводу (зазвичай, монітором);
cerr, clog - зв'язані із стандартним пристроєм виводу повідомлень про помилки (зазвичай, монітором).

Потоки **clog**, на відміну від потоків **cerr** є буферизованими. Тобто, кожна операція "помістити в потік cerr" приводить до миттєвої появи повідомлення про помилки, тоді, як кожна операція "помістити в потік clog" може привести до того, що вивід буде зберігатися в буфері до тих пір, поки буфер повністю не заповниться або ж поки вміст буферу не буде виведено примусово.

Потоки у C++ фактично являють собою класові об'єкти. Зокрема, потоки **cin**, **cout**, **cerr**, **clog** є об'єктами класів **istream** і **ostream**, які служать відповідно для введення та виведення стандартних типів у C++. Щоб мати можливість використовувати стандартні потоки у C++, необхідно підключити заголовний файл <**iostream**>.

Файлові потоки забезпечують файлове введення або виведення. Воно аналогічне стандартному, єдина відмінність - це те, що таке введення/виведення виконується не на екран, а у файл. Розвинені мови програмування також абстрагують файлові операції, перетворюючи їх на операції з потоками, які можуть бути "потоками введення" і "потоками виведення".

У C++ для реалізації файлового введення-виведення передбачено класи **ifstream** (для читання даних з файлу), **ofstream** (для запису даних у файл) і **fstream** (для читання і для запису даних), які є похідними через класи **istream**, **ostream** і **iostream** від класу **ios**, що містить базові функції введення-виведення низького рівня (рис. 1).

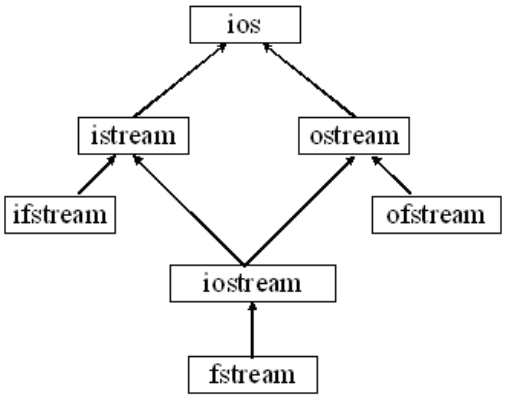

Рис. 1. Частина ієрархії класів потоків введення-виведення

Для реалізації файлового введення-виведення у C++ потрібно включити у програму заголовний файл **<fstream>**.

Кожен потік пов'язують з певним фізичним файлом, виконуючи операцію його *відкриття*. Основна відмінність між використанням стандартних і файлових потоків якраз і полягає у тому, що файловий потік повинен бути пов'язаний з файлом перш, ніж його можна буде використовувати; стандартні ж потоки можуть використовуватися відразу після запуску програми. Окрім того, файловий потік можна позиціонувати в довільну позицію у файлі, в той час як для стандартних потоків це, зазвичай, не має сенсу. Якщо дані читаються з файлу, то в будь-який момент можна повернутися до початку файлу і зчитати все заново.

За напрямком обміну інформацією потоки можна розділити на *вхідні* (дані вводяться в пам'ять), *вихідні* (дані вибираються з пам'яті) і *двонаправлені*. Зокрема, у C/C++ потік можна відкрити для читання і/або запису в текстовому або бінарному (двійковому) режимі.

Особливості обробки файлів у C++

Організація роботи з файлами

C++ підтримує всі функції введення/виведення мови C і, крім того, визначає свою власну об'єктно-орієнтовану систему введення/виведення. Файлові потоки у C++ створюються як об'єкти наступних класів:

ofstream - для виводу (запису) даних у файл;
ifstream - для введення (читання) даних з файлу;
fstream - для читання і для запису даних (двонаправлений обмін).

Щоб мати змогу використати ці класи, в текст програми необхідно включити заголовний файл **<fstream>**. Наприклад,

```
#include <fstream>
ofstream outFile;      // створення файлового потоку виведення
ifstream inFile;       // створення файлового потоку введення
fstream ioFile;        // створення файлового потоку для введення і виведення
```

У C++-програмах зв'язування файлового потоку з фізичним файлом на диску, як і в C, відбувається при відкритті файлу.

Створений потік можна відкрити за допомогою метода **open()**, який є членом кожного з трьох файлових класів **ofstream**, **ifsream** і **fstream**. Цей метода має наступний прототип:

```
void open(const char filename, int mode, int access);
```

Тут параметр **filename** є ім'ям файлу, яке може включати специфікатор шляху до файлу на диску; **mode** - визначає, в якому режимі відкривається файл; **access** - визначає, яким чином здійснюється доступ до файлу.

Усі можливі режими відкриття файла (параметр **mode**) визначені у заголовному файлі **<fstream>**. Їх перелік наведений в табл. 1.

Таблиця 1. Основні режими відкриття файлів у C++

Режим	*Призначення*
ios::in	відкрити файл для зчитування
ios::out	відкрити файл для запису
ios::binary	відкрити файл, вважаючи його бінарним
ios::app	відкрити для запису даних в кінець файла
ios::ate	після відкриття перемістити позицію зчитування-запису на кінець файла
ios::trunc	вилучити дані з існуючого файла

Імена прапорів відкриття є абревіатурами виконуваних дій: **app** – від *append* (додати, долучити), **ate** – від *at end* (в кінець), **trunc** – від *truncate* (урізати, відкинути) тощо. Всі вони є членами базового класу **ios**. Тому слід вказувати: **ios::in**, **ios::out** і т.ін.

Прапори **ate** і **app** є дуже схожими, оскільки переміщують файловий покажчик на кінець файлу. Відмінність між ними полягає в тому, що прапор **app** дозволяє

дописувати дані лише в кінець файла, в той час як прапор **ate** просто переміщує покажчик на кінець файла і не обмежує місця запису.

Режими відкриття файлів можна комбінувати за допомогою порозрядної логічної операції **"АБО"** (символ **“|”**), наприклад, режим **ios::out | ios::trunc** передбачає відкриття файла для запису, попередньо очистивши його.

В табл. 2 наведена відповідність поміж режимами відкривання файлів в С та С++.

Таблиця 2. Відповідність режимів відкривання файлів в С і С++

Режим С++	*Режим С*	*Призначення*
ios::in	**"r"**	Відкрити файл для зчитування. Позиція встановиться на початок файла
ios::out	**"w"**	Відкрити файл для запису. Позиція встановиться на початок файла. Якщо файл існує, записування здійснюватиметься поверх даних, тобто попередньо відбудеться урізання довжини файла до нуля
ios::out | ios::trunc		
ios::out | ios::app	**"a"**	Відкрити файл для запису даних в кінець файла
ios::in |ios::out	**"r+"**	Відкрити файл для зчитування-запису даних, починаючи з довільної позиції, тобто позиція може змінюватись
ios::in | ios::out | ios::trunc	**"w+"**	Відкрити файл для зчитування-запису, урізавши довжину файла до нуля для існуючого файла

Значення параметра **access** визначає режим доступу до файлу. Перелік можливих значень атрибутів файлів наведений у табл. 3.

Таблиця 3. Атрибути доступу до файлів

Атрибут	*Значення*
0	Звичайний файл, відкритий доступ
1	Файл тільки для читання
2	Прихований файл
4	Системний файл
8	Архівний файл

Відкриваючи файл, для параметрів **mode** і **access** можна використовувати значення, що діють за замовчуванням. Так при відкритті файлових потоків введення **ifstream** параметр **mode** за замовчуванням має значення **ios::in**; файлових потоків виведення **ofstream** - значення **ios::out | ios::trunc**; файлових потоків введення-виведення **fstream** - значення **ios::in | ios::out**. Для параметра **access** у будь-якому випадку за замовчуванням використовується варіант звичайного файлу (0). Якщо при відкритті файлу його тип не заданий, то вважається, що файл відкривається як текстовий. Наприклад, відкриття бінарного файлу з ім’ям **FileName.dat** як для зчитування, так і для запису:

```
#include <fstream>
fstream ioFile;                          // створення файла ioFile для введення і виведення
ioFile .open("FileName.dat", ios::binary);    // відкриття файла ioFile як бінарного
```

Слід пам'ятати, що будь-який файл можна відкрити як в текстовому, так і в бінарному режимі. Єдина відмінність між ними полягає в тому, проводиться перетворення символів при передачі чи ні.

У мові C++ файл можна відкрити і при створенні відповідного файлового потоку, оскільки класи **ofstream**, **ifstream** і **fstream** містять конструктори з параметрами, що відкривають файл автоматично:

```
ifstream(const char *name, ios::openmode mode= ios::in) ;
ofstream(const char* name, ios::openmode mode=ios::out | ios::trunc);
fstream(const char* name, ios::openmode mode= ios::in | ios::out).
```

Параметри цих конструкторів приймають за умовчанням ті ж значення, що і метод **open()**. З цієї причини файли зазвичай відкривають таким чином:

```
#include <fstream>
fstream  ioFile("FileName.dat", ios::binary);  //створення бінарного файла введення-виведення
ifstream  inFile("Test .dat");                 //створення текстового  файла введення
```

Якщо з яких-небудь причин файл не може бути відкритий, значення асоційованого потоку дорівнюватиме **NULL**. Можна скористатися наступним кодом для того, щоб переконатися, що файл дійсно відкритий:

```
ifstream  inFile ("Test .dat");                 // відкриття файла для читання
if (! inFile)
{  cout << "Cannot open file \n";               // обробка помилок
   return;
}
```

Перевірити успішність відкриття файлу можна також за допомогою метода **is_open()**, який є членом класів **fstream**, **ifstream** і **ofstream**. Він має наступний прототип:

```
bool  is_open();
```

Якщо потік вдалося зв'язати з відкритим файлом, ця функція повертає значення **true**, інакше - **false**. Наприклад, наступний фрагмент перевіряє, чи відкритий файл, пов'язаний з потоком iFile:

```
ifstream  inFile ("Test .dat");                 // відкриття файла для читання
if (! inFile.is_open())
{  cout << "Cannot open file \n";
   return 0;
}
```

Як тільки файл відкритий, можна проводити обмін інформацією між ним і програмою.

Мова C++ підтримує потокове введення-виведення. Відповідні програмні засоби для його реалізації містяться у заголовному файлі <**iostream**>.

Для визначення моменту досягнення кінця файлу використовується метод **eof()** потокового об'єкта, що має наступний прототип:

```
int eof();
```

При досягненні кінця файлу вона повертає ненульове значення, в іншому випадку повертається нуль. Використовуючи цикл **while**, можна читати вміст файлу, поки не буде досягнутий його кінець, як показано нижче:

```
while (! inFile.eof())
{
    // оператори
}
```

У даному випадку програма буде продовжувати виконувати цикл, поки функція **eof()** повертає значення **false** (0).

Для того, щоб закрити файл, слід викликати функцію close():

```
void close().
```

Функція **close()** не має параметрів і не повертає ніяких значень. Наприклад, закриття файла, зв'язаного з потоком input_file:

```
inFile.close();
```

Робота з текстовими файлами

У С++-програмах текстові файли створюються як файлові потоки читання (**ifstream**), запису (**ofstream**) і читання/запису (**fstream**). Якщо при відкритті файлового потоку його тип не заданий, то вважається, що файл відкривається як текстовий. Наприклад, створення текстового потоку для зчитування інформації із фізичного файлу з ім’ям **FileName.txt**:

```
ifstream  inFile("FileName.txt");
```

Читання і запис текстових файлів здійснюється за допомогою операцій “<<“ (“*помістити в потік*”) і “>>“ (“*взяти з потоку*”), як це робиться для консольного введення-виведення, тільки замість потоків **cin** і **cout** необхідно підставити потік, що зв’язаний з файлом:

```
filename << об’єкт1 << об’єкт2 << ... << об’єктN;
filename >> об’єкт1 >> об’єкт2 >> ... >> об’єктN;
```

Тут **filename –** ідентифікатор файлового потоку.

Перевагою цих операцій порівняно з розглянутими раніше функціями, є простота використання і автоматичне розпізнавання типів даних.

Однак при такому читанні з файла зчитування здійснюється до пробілу чи символу нового рядка. Тому це зчитування придатне лише для чисел та окремих слів. Для зчитування з потоку цілого рядка використовується метод **getline()**, який має наступний прототип:

```
istream& getline (char* buffer, int size, char delimiter = '\n');
```

Тут параметр **buffer** – це покажчик на рядок, що приймає символи з потоку; **size** - задає максимальне число символів для читання; **delimiter** - вказує символ-роздільник, який викликає припинення введення рядка до того, як буде введено

кількість символів, вказану в параметрі **size**. За замовчуванням параметру **delimiter** присвоюється значення '**\n**'.

Наприклад, оператор

```
fin.getline(s, n);
```

читає з потоку не більше n-1 символів і записує їх до змінної **s**.

Розглянемо, наприклад, такий код:

```
int i=1, j=25, i1, j1;
double a=25e6, a1;
char s[20], s1[20];
strcpy(s, "Іванов");
ofstream outFile ("Test .txt ");          // створення текстового файла як вихідного потоку
if(! outFile)
{  cout << "Cannot open file \n";         // обробка помилок
   return;
}
outFile << i << ' ' << j << ' ' << a << ' ' << s <<endl;
outFile.close();                          // закриття файла
ifstream in("Test .txt ");                // відкриття текстового файла як вхідного потоку
if(! inFile)
{ cout << "Cannot open file \n";          // обробка помилок
   return;
}
inFile >> i1 >> j1 >> a1 >> s1;
inFile.close();                           // закриття файла
```

У цьому коді створюється файл з ім'ям **Test.txt** і до нього записуються у текстовому вигляді два цілі числа – **i** та **j**, дійсне число **a** і рядок **s**, який містить одне слово, після чого маніпулятором потоку **endl** здійснюється перехід до нового рядка. Причому записування всіх цих даних у файловий потік **out** здійснюється одним оператором, що містить зчеплені операції "<<". Після того як файл закриється, в ньому буде записано текст "1 25 2.5e+07 Іванов". Наступні команди створюють вхідний потік, пов'язаний з цим файлом, і за допомогою операції ">>" читають дані з нього в змінні **i1**, **j1**, **a1** і **s1**.

Допускається форматування файлових потоків, так як і стандартних потоків **cin** і **cout**. Для управління форматуванням введення-виведення передбачені три види засобів: форматуючі функції, прапори і маніпулятори. Всі ці засоби є членами класу **ios** і тому доступні для всіх потоків. Наприклад, записати ціле число в шістнадцятковій системі в файл можна наступним чином:

```
ofstream outFile("test .txt");
outFile << hex << 1234;
```

Робота з бінарними файлами

Для отримання доступу до бінарного файлу потрібно створити потік відповідного типу (**ifstream**, **ofstream** або **fstream**) і відкрити його у двійковому режимі з використанням прапорця **ios::binary.** Наприклад,

```
ofstream outFile ("FileName.dat", ios::binary);        //створення бінарного файла для запису
```

Слід зауважити, що багато років двійкове введення-виведення в мовах C/C++ було байтовим (*byte oriented*). Це відбувалося тому, що символ (*char*) був еквівалентом байта, і потоки виведення були символьними. Проте з появою розширених *unicod*-символів (*wchar_t*) і пов'язаних з ними потоків систему введення-виведення мови C++ вже не можна назвати байтовою. Тепер її слід називати символьною (*character oriented*). Зрозуміло, що потоки звичайних символів (*char*) залишаються байтовими, особливо при обробці нетекстових даних. Проте еквівалентність понять "символ" і "байт" більше не гарантується.

Читання і запис двійкових даних у C++ може здійснюватися декількома способами.

Так блоки бінарних даних можна читати/записувати за допомогою методів **read()** та **write()** класів **ifstream** та **ofstream** відповідно:

```
istream &read(unsigned char *buf, int num);
ostream &write(const unsigned char *buf, int num);
```

Метод **read()** читає **num** символів з асоційованого потоку і посилає їх в буфер **buf**, метод **write**() пише **num** символів в асоційований потік з буфера **buf**. Ці функції працюють з символами (байтами). Для них немає жодного значення, в який спосіб організовані і що собою являють дані, – вони просто переносять їх посимвольно (побайтово) з буфера до файла і навпаки. Наприклад,

```
outFile.write((char*)&t, sizeof(t));
inFile.read((char*)&t, sizeof(t));
```

Тут перший оператор обчислює довжину змінної **t** у байтах і копіює до файла, зв’язаного з об’єктом **out**, обчислену кількість байтів, починаючи з вказаної адреси. Другий оператор копіює кількість байтів, яка відповідає довжині змінної **t**, з файла **in** до змінної **t**. Оскільки методи **read**() і **write**(), як перший параметр, очікують покажчик на символьний масив, були виконані операції приведення типу (**char** *) &t. Для визначення довжини змінної **t** використана функція **sizeof**(), яка обчислює розмір змінної-аргумента у байтах.

Якщо кінець файлу досягається до того, як буде прочитано задане число символів (байт), метод **read**() просто припиняє роботу і буфер містить стільки символів, скільки було прочитано. Щоб визначити, скільки символів було прочитано, використовують іншу функцію-член **gcount**(), що має наступний прототип:

```
int gcount();
```

Вона повертає число символів, прочитаних останньою функцією **read**().

Інший спосіб читання і запису двійкових даних полягає у використанні метода **get()** потоку **istream** і метода **put()** потоку **ostream**. При читанні символу (а не розширення символу), ці функції читають і записують байти.

Метод **get()** має багато форм, але найбільш вживана його версія показана нижче, де наведений також і метод **put()**:

```
istream &get(char &ch);
ostream &put(char ch);
```

Метод **get()** читає окремий символ з асоційованого потоку і записує його значення в змінну **ch**. Крім того, метод **get()** повертає посилання на потік. Метод **put()** записує значення змінної **ch** в потік і повертає посилання на цей потік. Наприклад, виведення вмісту файла на екран:

```
char ch;
ifstream  inFile ("FileName.dat", ios::binary);
if (! inFile)
{  cout << "Cannot open file\n";
   return 1;
}
while (inFile)              // inFile буде нулем при досягненні кінця файла
{  inFile.get (ch);
   cout << ch;
}
inFile.close();
```

Коли файловий покажчик **in** досягає кінця файлу, то приймає значення **NULL**, в результаті чого цикл **while** закінчується. Можливий більш компактний запис коду для цього циклу:

```
while (inFile.get(ch))
    cout << ch;
```

Файлова система введення/виведення C++ обробляє два покажчика, асоційовані з кожним файлом: покажчик зчитування (**get**-покажчик), який задає місце у файлі, звідки буде вводитися наступна порція інформації і покажчик запису (**put**-покажчик), який задає місце у файлі, куди буде виводитися наступна порція інформації.

При кожному введенні або виведенні відповідний вказівник послідовно просувається далі. Проте за допомогою методів **seekg()** і **seekp()** можливий довільний (прямий) доступ до файлу (*random access*). Їх найбільш уживана форма наступна:

```
istream&  seekg(long offset, seek_dir dir);
ostream&  seekp(long offset, seek_dir dir);
```

Метод **seekg()** переміщує поточний **get**-покажчик на **offset** байт в напрямку, заданому **dir**; метод **seekp()** переміщує на **offset** байт в напрямку, заданому **dir**,

поточний **put**-покажчик. Параметр **dir** задає напрямок зсуву, який може приймати значення, визначені типом-перерахуванням **seek_dir**[1]:

ios :: beg – зсув від початку файла;
ios :: cur - зсув відносно поточної позиції;
ios :: end - зсув від кінця файла.

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

Отримати поточне значення позиції в потоці введення або виведення можна за допомогою методів **tellg()** і **tellp()** відповідно. Ці методи мають наступні прототипи:

```
long tellg();
long tellp();
```

Наступний приклад демонструє можливість позиціонування потоку введення інформації:

```
#include <iostream>
#include <fstream>
void main( )
{  int  size = 0;
   char name[20];
   cout << "File name : "; cin >> name;
   ofstream outFile;
   outFile.open(name, ios::binary );
   for (int  i= 0; i<100; i++)
     outFile.put ((char)(i+27));
   outFile.close();
   ifstream  inFile;
   inFile.open(name, ios::binary);
   if (inFile)
   {  inFile.seekg(0, ios::end);
      size = inFile.tellg();
      cout << name << " size = " << size<< endl;
   }
   systems("pause");
}
```

Дана програма виводить на екран довжину вказаного файлу.

[1] enum seek_dir { beg, cur, end };