Lec_ Файли.

Не дивлячись на те що система введення-виведення мови С++ у цілому являє собою єдиний механізм, система файлового введення-виведення має свої особливості. Частково це пояснюється тим, що на практиці найчастіше використовуються файли на жорсткому диску, можливості яких значно відрізняються від усіх інших обладнань. Однак слід мати на увазі, що файлове введення- виведення є лише частиною загальної системи введення- виведення, в лекції розглядатимуться потоки, які можуть бути пов'язані з іншою апаратурою.

Заголовок <fstream> і класи файлів

Для реалізації файлового введення- виведення в програму слід включити заголовок <*fstream*>. У ньому визначені деякі класи, зокрема , *ifstream*, *ofstream* і *fstream*. Ці класи є похідними від класів *istream*, *ostream* і *iostream* відповідно. Слід пам'ятати, що класи *istream*, *ostream* і *iostream*, у свою чергу, є похідними від класу *ios*. Файлова система використовує також клас *filebuf*, що надає низькорівневі засоби керування файловим потоком. Звичайно клас *filebuf* безпосередньо не застосовується, однак він є складовою частиною інших класів.

Відкриття й закриття файлу

У мові С++ відкриття файлу означає його зв'язування з потоком. Отже, спочатку необхідно одержати потік. Існують три види потоків: уведення, виведення й вводу-виведення. Для того щоб створити потік уведення, необхідно оголосити потік, що представляє собою об'єкт класу ifstream. Для генерації потоку виведення необхідно оголосити потік, що представляє собою об'єкт класу ofstream. Потоки, що здійснюють уведення й виведення, оголошуються як об'єкти класу fstream. Наприклад, у наступному фрагменті створюється потік уведення, потік виведення й потік вводу-виведення.

```
ifstream in;      //   Уведення
      ofstream out;   //   Виведення
      fstream  io;        //   Уведення й  виведення
```

Створений потік можна зв'язати з файлом за допомогою функції open(). Ця функція є членом кожного із трьох потокових класів. Її прототипи в кожному класі показані нижче.

```
ifstream::open(const  char* filename, ios::openmode mode = ios::in);
ifstream::open(const  char* filename, ios::openmode mode = ios::out | ios::trunc);
ifstream::open(const  char* filename, ios::openmode mode = ios::in | ios::out);
```

Тут параметр filename задає ім'я файлу. Він може містити шлях до цього файлу. Значення параметра mode визначає спосіб відкриття файлу. Цей параметр може ухвалювати наступні значення, описані у функції openmode.

```
ios::app
ios::ate
ios::binary
ios::in
ios::out
ios::trunc
```

Дані значення можна комбінувати за допомогою логічної операції "АБО"

Якщо задається значення **ios::app**, усі результати дописуються в кінець файлу, відкритого для виведення.

Якщо зазначене значення **ios::ate**, при відкритті виконується пошук кінця файлу. Незважаючи на це, запис проводиться в будь-яке місце файлу.

Якщо задається значення **ios: :in**, файл відкривається для введення, а якщо **ios: :out** - для виведення.

Значення **ios::binary** дозволяє відкрити файл у бінарному режимі. За замовчуванням усі файли відкриваються в текстовому режимі. У цьому випадку проводиться перетворення деяких символів, наприклад, эскейп- послідовність "повернення каретки" і "прогін паперу" перетвориться в символ переходу на новий рядок. Однак, якщо файл відкритий у бінарному режимі, перетворення символів не проводиться. Слід пам'ятати, що будь-який файл можна відкрити як у текстовому, так і в бінарному режимі. Єдина відмінність між ними полягає в тому, проводиться перетворення символів чи ні.

Значення **ios: :trunc** сигналізує, що попередній уміст існуючого файлу з тим же іменем буде знищено, а довжина файлу зменшена до нуля. При відкритті потоку виведення за допомогою класу ofstream уміст будь-якого існуючого файлу із зазначеним іменем стирається.

У наступному фрагменті відкривається текстовий файл для виведення.

```
ofstream out;
out.open("test", ios::out);
```

Однак функція **open()** рідко застосовується для відкриття файлів, тому що для кожного типу потоку параметр mode має значення, задані за замовчуванням. Прототипи функції **open** демонструють, що значення параметра **mode**, задане за замовчуванням, у класі **ifstream** рівно **ios:: in**, у класі **ofstream - ios: :out | ios: :trunc**, а в класі **fstream - ios: :in | ios: :out**. Із цієї причини попередній виклик функції open звичайно записують так:

```
out.open("test");   //   Текстовий файл для  виведення
```

Залежно від компілятора параметр mode для функції f stream: :open() може мати значення за замовчуванням, відмінне від in \ out. Отже, іноді його необхідно задавати явно.

Якщо потік застосовується в логічних виразах, у випадку відмови функції **open()** йому привласнюється значення **false**. Таким чином, перед використанням файлу слід переконатися, що він успішно відкритий. Для цього можна скористатися наступними операторами.

```
if(!mystream) {
      cout << "Неможливо  відкрити  файл.\n"; //  Обробка помилки }
```

Однак частіше всього функцію **open()** не застосовують, оскільки класи **ifstream**, **ofstream** і **fstream** містять конструктори, що автоматично відкривають файл. Параметри цих конструкторів ухвалюють ті ж значення за замовчуванням, що й функція **open()**. Із цієї причини файли звичайно відкривають у такий спосіб.

```
ifstream mystream("myfile");   //   Відкриття  файлу для  введення
```

Якщо з якої-небудь причини файл відкрити не вдалося, пов'язаному з ним потоку привласнюється значення false. Отже, якщо якийсь конструктор викликає функцію **open()**, слід переконатися, що файл дійсно відкритий, перевіривши значення потоку.
Щоб перевірити, чи відкритий файл, можна викликати функцію **is_open()**, що є членом класу **fstream**, **ifstream** і **ofstream** Вона має наступний прототип.

```
bool   is_open();
```

Якщо потік пов'язаний з відкритим файлом, ця функція повертає значення **true**, а якщо ні, то вона повертає значення **false**. Наприклад нище фрагмент, перевіряє, чи відкритий файл, пов'язаний з потоком **mystream**.

```
if(!mystream.is_open()) {
            cout << "Файл не відкритий.\n";
      }
```

Щоб закрити файл, слід викликати функцію **close()**. Наприклад, щоб закрити файл, пов'язаний з потоком **mystream**, можна застосувати наступний оператор.

```
mystream.close();
```

Функція **close ()** не має параметрів і не повертає ніяких значень.

Читання й запис текстових файлів

Читання й запис текстових файлів здійснюються дуже легко. Для цього досить застосувати оператори "<<" і ">>", як це звичайно робиться для консольного введення- виведення, тільки замість потоків cin і cout необхідно підставити потік, звязаний з файлом. Наприклад наступна програма створює короткий файл, що містить назву предмета і його вартість.

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
      ofstream out("INVNTRY"); // Текстовий файл для виведення.
      if(!out) {
            cout << "Неможливо відкрити файл INVENTORY.\n";
```

```
            return 1;
      }
      out << "Радіоприймачі " << 39.95 << endl;
      out << "Тостери " << 19.95 << endl;
      out << "Міксери " << 24.80 << endl;
      out.close();
      return 0;
}
```

Програма, наведена нижче, зчитує файл, створений попередньої програмою, і виводить його вміст на екран.

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
      ifstream in("INVNTRY"); // Уведення
      if(in) {
            cout << "Неможливо відкрити файл INVENTORY.\n";
            return 1;
      }
      char item[20]; float cost;
      in >> item >> cost;
      cout << item << " " << cost << "\n";
      in >> item >> cost;
      cout << item << " " << cost << "\n";
      in >> item >> cost;
      cout << item << " " << cost << "\n";
      in.close(); return 0;
}
```

Наступна програма зчитує рядки, уведені із клавіатури, і записує їх на диск. Програма зупиняється, якщо користувач увів знак оклику.

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {

      ofstream out("my.txt"); // Текстовий файл для виведення.
      if(!out) {
            cout << "Неможливо відкрити файл для виведення.\n";
            return 1;
      }
      char str[80];
      cout << "Запис рядків на жорсткий диск. Для припинення роботи введіть знак оклику.\n";
      do {
            cout << ": ";
            cin << str;
            out << str << endl;
      } while(*str != '!');
      out.close();
      return  0;
}
```

Зчитуючи файли за допомогою оператора ">>", майте на увазі, що деякі символи при введенні трансформуються. Наприклад, роздільники ігноруються. Щоб запобігти перетворенню символів при читанні, файл слід відкрити в бінарному режимі й застосувати функції, описані в наступному розділі.
Якщо при введенні досягається кінець файлу, потоку, пов'язаному з файлом, привласнюється значення false. (Ця ситуація ілюструється в наступному розділі.)

Бесформатне і бінарне введення- виведення

Отже, читання й запис форматних текстових файлів не викликає ніяких труднощів, хоча це не найефективніший спосіб роботи з файлами. Крім того, іноді виникає необхідність зберігати бесформатні дані, а не текст. Розглянемо функції, призначені для роботи з такими даними.

Виконуючи з файлом бінарні операції, слід переконатися, що він відкритий у режимі ios::binary. Бесформатні дані можуть зберігатися й у текстовому файлі, але в цьому випадку при читанні деякі символи будуть перетворені. Бінарні файли застосовуються саме для того, щоб цього уникнути.

Порівняння символів і байтів.

При вивченні бесформатного введення-виведення необхідно враховувати наступне. Багато років введення-виведення у мовах C и C++ було байтовим (byte oriented). Це відбувалося тому, що символ (char) є еквівалентом байта, і потоки введення-виведення були символьними. Однак з появою розширених символів (wchar_t) і пов'язаних з ними потоків систему введення-виведення мови C++ не можна назвати байтовою. Тепер її слід називати символьною (character oriented). Зрозуміло, потоки звичайних символів (char) залишаються байтовими, особливо при обробці нетекстових даних. Однак еквівалентність понять "символ" і "байт" більше не гарантується.

Функції put() і get()

Один зі способів зчитування й запису бесформатных файлів заснованих на застосуванні функцій **put()** і **get()**. Ці функції оперують символами. Точніше кажучи, функція **get()** зчитує символ, а функція **put()** - записує його. Зрозуміло, якщо файл відкритий у бінарному режимі, то при зчитуванні символу (а не розширеного символу), ці функції зчитують і записують байти.

Функція **get()** має кілька форм, однак найчастіше використовується її наступна версія. "Клас:ostream: функція-член:put”

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

Функція **get()** зчитує окремий символ з потоку й записує його в змінну **ch**. Крім того, функція **get()** повертає посилання на потік. Функція **put()** записує змінну **ch** у потік і повертає посилання на потік.

Наступна програма відображає на екрані вміст будь-якого файлу (як текстового, так і бінарного). Для зчитування данных вона використовує функцію **get()**.

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {

	char ch;

	ifstream in("myfile.txt", ios::in | ios::binary);
	if(!in) {
		cout << "Неможливо відкрити файл.";
		return 1;
	}
	while(in) {  // Якщо досягнуться кінець файлу,
	// значення об'єкта in рівно false.
		in.get(ch);
		if(in) cout << ch;
	}
	return 0;
}
```

По досягненню кінця файлу потік, пов'язаний із цим файлом, набуває значення **false**. Отже, рано або пізно об'єкт **in** прийме значення **false**, і виконання циклу **while** припиниться.

Наведений вище цикл можна записати коротше.

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

Цей фрагмент є правильним, оскільки функція **get()** повертає посилання на потік **in**, який прийме значення **false** при виявленні кінця файлу.

У наступній програмі для запису в файл CHARS усіх символів від 0 до 255 застосовується функція **put()**. Як відомо, символи ASCII займають лише половину з можливих значень типу **char**. Інші символи, як правило, називаються розширеними (**extended character set**). До них відносяться букви національних алфавітів і математичні знаки. (Деякі системи не підтримують розширені символи.)

```
#include <iostream>
#include <fstream>
```

```
using namespace std;
int main() {
	int i;
	ofstream out("chars", ios::out | ios::binary);
		if(!out) {
		cout << "неможливо відкрити файл.\n";
		return 1;
	}
	// Записати символи на диск.
	for (i = 0; i < 256; i++) out.put((char)i);
	out.close(); return 0;
}
```

За допомогою цієї програми можна перевірити, чи підтримує ваш комп'ютер розширені символи.

Функції read() і write()

Блоки бінарних даних можна зчитувати за допомогою функцій **read()** і **write()**. Їхні прототипи виглядають у такий спосіб.

```
	istream& read(char* buf, streamsize num);
	istream& write(const char* buf, streamsize num);
```

Функція **read** зчитує **num** символів з потоку й записує їх у буфер, на який посилається покажчик **buf**. Функція **write** записує **num** символів у потік, зчитуючи їх з буфера, на який посилається покажчик **buf**. Тип **streamsize** визначений у бібліотеці як різновид типу **int**. Він дозволяє зберігати максимальна кількість символів, які можуть перетворюватися при виконанні операцій введення-виведення.
Наступна програма записує структуру на диск, а потім зчитує назад.

```
#include <iostream>
#include <fstream>
#include <cstring>
using namespace std;
struct status {
	char name[80];
	double balance;
	unsigned long account_num;
};
int main() {
	struct status ace, acc;
	strcpy(ace.name, "Ральф Трантор");
	ace.balance = 1123.23;
	ace.account_num = 34235678;
	// Записуємо дані
	ofstream outbal("balance", ios::out | ios::binary);
	if(!outbal) {
		cout << "Неможливо відкрити файл.\n";
		return 1;
	}
	outbal.write((char*)&ace, sizeof(struct status));
	outbal.close();
	// Зчитуємо дані знову
	ifstream inbal("balance", ios::in | ios::binary);
	if(!inbal) {
		cout << "Неможливо відкрити файл.\n";
		return 1;
	}
	inbal.read((char*)&acc, sizeof(struct status));
	cout << acc.name << endl;
	cout << "Рахунок # " << ace.account_num;
	cout.precision(2);
	cout.setf(ios::fixed);
	cout << endl << "Баланс: $" << ace.balance;
	inbal.close(); return 0;
}
```

Як бачимо, для зчитування або запису цілої структури досить одного виклику функції **read()** або **write()**. Окреме поле структури неможливо вважати або записати окремо. Крім того, цей приклад показує, що буфером може служити об'єкт будь-якого типу.

Якщо буфер не є символьним масивом, при виклику функцій read() й write() необхідно виконувати переведення типів. Оскільки в мові C++ виконується строга перевірка типів, покажчик одного типу не може автоматично перетворюватися в покажчик іншого типу.

Якщо кінець файлу виявиться перш, ніж будуть злічені **num** символів, функція **read()** просто припинить роботу, а в буфері буде записана максимально можлива кількість символів. Кількість зчитаних символів можна визначити за допомогою функції **gcount()**. Її прототип виглядає в такий спосіб.

```
streamsize  gcount();
```

Дана функція повертає кількість символів, зчитаних при виконанні останньої операції бінарного введення. Розглянемо ще один приклад використання функцій **read()**, **write()** і **gcount()**.

```
#include <iostream>
#include <fstream>
#include <cstring>
using namespace std;
struct status {
	char name[80];
	double balance;
	unsigned long account_num;
};
int main() {
	double fnum[4] = { 99.75, -34.4, 1776.0, 200.1 };
	int i;
	ofstream out("numbers", ios::out | ios::bwnary);
	if(!out) {
		cout << "Неможливо відкрити файл.";
		return 1;
	}
	out.write((char*)&fnum, sizeof fnum);
	out.close();
	for (i = 0; i < 4; i++) // Очищення масиву.
		fnum[i] = 0.0;
	ifstream in("numbers", ios::in | ios::binary);
	in.read((char*)&fnum, sizeof fnum);
	// Визначаємо кількість лічених символів.
	cout << "Зчитано " << in.gcount() <<" байтів.\n";
	for (i = 0; i < 4; i++) // Показати значення, зчитані з файлу.
		cout << fnum[i] << " ";
			in.close();
	return 0;
}
```

Ця програма записує масив десяткових чисел на жорсткий диск, а потім зчитує їх назад. Після виклику функції **read()** застосовується функція **gcount()**, що повертає кількість лічених символів.

Додаткові функції get()

Крім попередніх форм функція **get()** має ще кілька перевантажених різновидів. Розглянемо три основні прототипи.

```
istream& get(char* buf, streamsize num);
istream& get(char* buf, streamsize num, char delim);
int get();
```

Перша форма функції get зчитує символи з масиву, на який посилається покажчик buf поки не будуть зчитані num-1 символів, виявлений символ переходу на наступний рядок або досягнеться кінець файлу. Функція **get()** записує нульовий символ у кінець масиву, на який посилається покажчик **buf** Символ переходу на новий рядок не зчитується. Він залишається в потоці, поки не буде виконана наступна операція введення.

Друга форма функції **get** зчитує символи в масив, на який посилається покажчик **buf**, поки не будуть зчитані num-1 символів, виявлений символ delim або досягнутий кінець файлу. Функція **get()** записує нульовий символ у кінець масиву, на який посилається покажчик **buf**. Символ **delim** з потоку не зчитується. Він залишається в потоці, поки не буде виконана наступна операція введення.

Третя форма функції **get** витягає з потоку наступний символ. Якщо виявлений кінець файлу, вона

повертає константу **EOF**. Ця форма функції **get()** схожа на функцію getc () у мові С.

Функція getline()

Крім функції **get()** у мові С++ існує функція **getline()**. Вона є членом кожного потокового класу. Її прототипи виглядають у такий спосіб.

```
istream& getline(char* buf, streamsize num);
istream& getline(char* buf, streamsize num, char delim);
```

Перша форма функції **getline** зчитує символи з масиву, на який посилається покажчик **buf** поки не будуть зчитані num-1 символів, виявлений символ переходу на наступний рядок або досягнутий кінець файлу. Функція getline() записує нульовий символ у кінець масиву, на який посилається покажчик **buf** Символ переходу на новий рядок видаляється з потоку, але не записується в буфер.
Друга форма функції **getline** зчитує символи в масив, на який посилається покажчик **buf**, поки не будуть зчитані num-1 символів, виявлений символ **delim** або досягнутий кінець файлу. Функція **getline()** записує нульовий символ у кінець масиву, на який посилається покажчик **buf.** Символ **delim** видаляється з потоку, але не записується в буфер.
Як бачимо, дві версії функції **getline()** дуже нагадують версії **get (buf, num)** і **get(buf, num, delim)**. Різниця полягає в тому, що на відміну від функції **get()** функція **getline()** видаляє роздільник з потоку.
Розглянемо програму, що демонструє роботу функції **getline()**. Вона зчитує вміст текстового файлу й виводить його на екран.

```
// Програма зчитує й виводить на екран рядки, зчитані з текстового файлу.
#include <iostream>
#include <fstream>
using namespace std;
int main() {

	ifstream in("myfile"); // Уведення
	if(!in) {
		cout << "Неможливо відкрити файл\n";
			return 1;
	}
	char str[255];
	while(in) {
		in.getline(str, 255);  // За замовчуванням delim == '\n'.
		if(in) cout << str << endl;
	}
	in.close();
	return 0;
}
```

Розпізнавання кінця файлу

Розпізнати кінець файлу можна за допомогою функції **eof()**, що має прототип

```
bool eof();
```

Якщо досягнуто кінець файлу, вона повертає значення true, а якщо ні, то вона повертає значення false.

Функція ignore()

За допомогою функції Ignore () можна вважати й проігнорувати символ із вхідного потоку.
| istream &ignore(streamsize num=l, int_type delim=EOF);
Вона зчитує й відкидає символи, поки не будуть пропущені пит символів (за замовчуванням параметр пит рівний 1) або не зустрінеться символ delim, який за замовчуванням дорівнює константі EOF. Виявлений роздільник не віддаляється з потоку введення. Тип int_type визначений як різновид типу int.
Наступна програма зчитує файл TEST. Вона ігнорує символи, поки не зустрінеться пробіл або не будуть лічено 10 символів. Потім вона виводить на екран іншу частину файлу.

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
	ifstream in("test");
```

```
        if(!in) {
        cout << "Неможливо відкрити файл.\n";
              return 1;
}
/* Ігноруються 10 символів, поки не зустрінеться пробіл. */
        in.ignore(10, ' ');
        char c;
        while(in) {
        in.get(c);
        if(in) cout << c;
        }
              in.close();
              return 0;
}
```

Функції peekq і putback()

Можна отримати наступний символ з потоку, не витягаючи його звідти. Для цього призначена функція **peekq** (), що має прототип, наведений нижче.

```
int_type peekq();
```

Ця функція повертає наступний символ з потоку введення або ознака кінця файлу. Тип int_type визначений як різновид типу int
Символ, зчитаний з потоку останнім, можна повернути назад за допомогою функції putback(). Її прототип має такий вигляд.

```
istream& putback(char c);
```

Тут параметр с означає символ, зчитаний останнім.

Функція flush()

При виведенні дані не відразу передаються фізичному обладнанню, пов'язаному з потоком. Замість цього вони накопичуються у внутрішньому буфері, поки він не заповниться. Однак існує спосіб примусово записати інформацію з буфера на диск, не чекаючи його заповнення. Для цього призначена функція flush(). Її прототип має такий вигляд.

```
ostream& flush();
```

Функцію flush слід викликати, коли програма виконується в несприятливих умовах (наприклад, якщо часто відбуваються збої харчування).
Закриття файлу або припинення роботи програми також очищає всі буфери.

Довільний доступ

Довільний доступ до файлу забезпечується функціями seekg() і seekp(). Їхні прототипи мають такий вигляд.

```
istream& seekg(off_type offset, seekdir origin);
ostream& seekp(off_type offset, seekdir origin);
```

Тип off_type є різновидом цілого типу. Він визначений у класі ios і дозволяє зберігати максимальні значення, які може ухвалювати параметр offset. Тип seekdir являє собою перерахування, певне в класі ios. У ньому втримуються різновиди пошуку, виконуваного функціями seekg і seekp.
Система введення- виведення мови С++ управляє двома покажчиками, пов'язаними з файлами. Перший покажчик, що визначає позицію, у якій виконується читання файлу, називається курсором читання (get pointer). Інший покажчик, що визначає позицію, у якій виконується запис у файл, називається курсором запису (put pointer). Щораз при виконанні введення й виведення відповідний курсор файлу переміщається на одну позицію вперед. Однак функції seekg() і seekpo дозволяють виконувати довільні переміщення по файлу.
Функція seekg() переміщає пов'язаний з нею курсор запису на offset символів, відраховуючи від позиції origin. Позиція origin задається трьома можливими значеннями.

```
ios::beg       // Початок файлу
ios::cur     //   Поточне положення
ios::end     //   Кінець файлу
```

Функція seekp() переміщає пов'язаний з нею курсор запису на offset символів, відраховуючи від позиції origin. Позиція origin задається трьома можливими значеннями зазначеними вище.
Як правило, довільний доступ при введенні- виведення здійснюється тільки до файлу, відкритого в бінарному режимі. Перетворення символів при зчитуванні текстових файлів може порушити правильний порядок проходження байтів у файлі.
Наступна програма демонструє роботу функції seekp(). Вона дозволяє змінювати зазначений символ у файлі.. Зверніть увагу на те, що файл відкритий для операцій вводу-виведення.

```
#include  <iostream>
#include  <fstream>
#include  <cstdlib>
using namespace  std;
int  main() {

            cout << "Застосування: CHANGE < ім'я файлу> <старий символ> <новий символ>\n";

                  fstream out("myfile.in", ios::in | ios::out | ios::binary);
                  if(!out) {
                  cout  <<   "Неможливо  відкрити  файл.";
                        return  1;
            }
                  int numb_char; char new_char;
                  cin >> numb_char >> new_char;

            out.seekp(numb_char, ios::beg);
            out.put(new_char); out.close();
            return 0;
      }
```

Наступна програма використовує функцію seekg(). Вона виводить на екран уміст файлу, починаючи з позиції, зазначеної в командному рядку.

```
#include <iostream>
#include <fstream>
#include <cstdlib>
using namespace std;
      int main(int argc, char* argv[]) {
      char ch;
      if(argc != 3) {
            cout << "Застосування: SHOW < ім'я файлу> <початкова позиція>\n";
                  return 1;
      }
      ifstream in(argv[1], ios::in | ios::binary);
      if(!in) {
            cout << "Неможливо відкрити файл.";
                  return 1;
      }
      in.seekg(atoii(argv[2]), ios::beg);
      while(in.get(ch)) cout << ch;
            return 0;
}
```

Нижче показане, як за допомогою функцій seekp() і seekg() переставити у зворотному порядку перші пит символів у файлі.

```
#include <iostream>
#include <fstream>
#include <cstdlib>
using namespace std;
int main(int argc, char* argv[]) {
      if(argc != 3) {
```

```
		cout << "Застосування: Reverse < ім'я файлу> <num>\n"; return 1; }
			fstream inout(argv[1], ios::in | ios::out | ios::binary);
		if(!inout) {
			cout << "Неможливо відкрити файл.\n";
			return 1;
		}
		long e, i, j;
		char c1, c2;
		e = atol(argv[2]);
		for (i = 0, j = e; i < j; i++, j--) {
			inout.seekg(i, ios::beg); inout.get(c1); inout.seekg(j, ios::beg);
inout.get(c2);
			inout.seekp(i, ios::beg); inout.put(c2); inout.seekp(j, ios::beg);
inout.put(c1);
		}
		inout.close(); return 0;
	}
```

Визначення поточної позиції

Поточну позицію курсорів читання й записи можна визначити за допомогою функцій tellg() і tellp(). Їхні прототипи мають такий вигляд

```
pos_type   tellg();
pos_type  tellp();
```

Тип pos_type визначений у класі ios і дозволяє зберігати максимальне значення, яке може повернути функція. Значення, що вертаються функціями tellp() і tellg(), можна використовувати в якості аргументів функції seekg() і seekp() відповідно.

```
istream& seekg(pos_type pos);
ostream& seekg(pos_type pos);
```

Ці функції дозволяють зберегти поточне положення файлового курсору, виконати певні файлові операції, а потім відновити колишнє положення курсору.

Статус уведення- виведення

Система введення- виведення мови C++ зберігає інформацію про результат каждбй операції вводу-виведення. Поточний стан системи введення- виведення зберігається в об'єкті класу iostate, який є перерахуванням, певним у класі ios. Крім цього, клас ios містить наступні члени.

```
/Ім'я                          Значення
ios::goodbit           //  Набір байтів, що описують нормальний стан.
ios::eofbit           //    1  якщо виявлений кінець файлу, 0 а якщо ні.
ios::failbit          //   1, якщо виявлена(можливо) поправна помилка, 0 а якщо ні.
ios::badbit         //   1, якщо виявлена непоправна помилка, 0 а якщо ні.
```

Існують два способи одержати інформацію про статус вводу-виведення. По-перше, можна викликати функцію rdstate(). Вона має наступний прототип.

```
iostate  rdstate();
```

Ця функція повертає поточний стан прапорів помилок. Як випливає из вищесказаного, якщо ніяких помилок не виявлене, функція rdstate () повертає значення goodbit. А якщо ні, то встановлюється прапор помилки.

Застосування функції rdstate() ілюструється наступною програмою.

```
#include  <iostream>
#include  <fstream>
using namespace  std;
void checkstatus(ifstream& in);
int main(int argc, char* argv[]) {
	if(argc != 2) {
		cout << "Застосування: Display < ім'я файлу>\n"; return 1; }
	ifstream in(argv[1]);
			if(!in) {
```

```
            cout << "Неможливо відкрити файл.\n";
            return 1;
      }
      char c;
      while(in.get(c)) {
      if(in) cout << c;
      checkstatus(in); }
      checkstatus(in);  // Перевірка заключного стану
      in.close();
      return 0; }
      void checkstatus(ifstream & in) {
            ios::iostate i;
            i = in.rdstate();
            if(i & ios::eofbit)
                  cout << "Виявлений  кінець файлу\n";
            else   if(i & ios::failbit)
                  cout << "Виявлена поправна  помилка\n";
            else  if(i & ios::badbit)
                  cout   <<   "Виявлена  непоправна  ошибка\n"; }
```

Ця програма завжди виявляє одну "помилку". Після завершення циклу while останній виклик функції checkstatus (), як і очікувалося, виявляє кінець файлу. Ця функція може виявитися корисної в будь-якій програмі.
Другий спосіб виявлення помилки заснований на застосуванні наступних функцій.

```
      bool  bad();
      bool  eof();
      bool   fail();
      bool  good();
```

Функція bad() повертає значення true, якщо встановлений прапор badbit. Функція eof () повертає значення true, якщо встановлений прапор failbit. Функція good() повертає значення true, якщо ніяких помилок не виявлене. А якщо ні, то функція повертає значення false.
Прапори, відповідні до виявлених помилок, можна скинути. Для цього слід викликати clear (), прототип якої має такий вигляд.

```
void clear(iostate flags = ios::goodbit);
```

Якщо параметр flags є об'єктом goodbit (за замовчуванням), усі прапори помилок скидаються. А якщо ні, то параметр flags слід задати довільно.

ПІДСУМКИ

Відкриття файлу

```
#include <fstream>
#include <iostream>

int main() {
    std::ifstream inputFile("input.txt"); // для читання
    std::ofstream outputFile("output.txt"); // для запису

    if (!inputFile.is_open() || !outputFile.is_open()) {
        std::cerr << "Unable to open file." << std::endl;
        return 1; // повертаємо код помилки
    }

    // ваш код для роботи з файлами

    inputFile.close(); // закриваємо файл після використання
    outputFile.close();

    return 0; // програма завершилася успішно
}
```

Читання з файлу:

```
#include <fstream>
#include <iostream>
```

```
#include <string>

int main() {
    std::ifstream inputFile("input.txt");

    if (!inputFile.is_open()) {
        std::cerr << "Unable to open file." << std::endl;
        return 1;
    }

    std::string line;
    while (std::getline(inputFile, line)) {
        std::cout << line << std::endl;
    }

    inputFile.close();

    return 0;
}
```

Запис у файл:

```
#include <fstream>
#include <iostream>

int main() {
    std::ofstream outputFile("output.txt");

    if (!outputFile.is_open()) {
        std::cerr << "Unable to open file." << std::endl;
        return 1;
    }

    outputFile << "Hello, World!" << std::endl;

    outputFile.close();

    return 0;
}
```

Позиціювання покажчика файлу:

```
#include <fstream>
#include <iostream>

int main() {
    std::fstream file("example.txt", std::ios::in | std::ios::out);

    if (!file.is_open()) {
        std::cerr << "Unable to open file." << std::endl;
        return 1;
    }

    file.seekg(5, std::ios::beg); // Переміщення покажчика для читання на 5 байт від початку
файлу
    file.seekp(10, std::ios::beg); // Переміщення покажчика для запису на 10 байт від початку
файлу

    // ваш код

    file.close();

    return 0;
}
```

Перевірка кінця файлу:

```
#include <fstream>
#include <iostream>
#include <string>

int main() {
    std::ifstream file("example.txt");
```

```
    if (!file.is_open()) {
        std::cerr << "Unable to open file." << std::endl;
        return 1;
    }

    std::string content;
    while (!file.eof()) {
        std::getline(file, content);
        std::cout << content << std::endl;
    }

    file.close();

    return 0;
}
```

Перейменування файлу:

```
#include <iostream>

int main() {
    const char* oldName = "old_name.txt";
    const char* newName = "new_name.txt";

    if (std::rename(oldName, newName) != 0) {
        std::cerr << "Error renaming file." << std::endl;
        return 1;
    }

    return 0;
}
```

Перевірка існування файлу:

```
#include <iostream>
#include <fstream>

int main() {
    const char* fileName = "example.txt";

    std::ifstream file(fileName);
    if (file.good()) {
        std::cout << "File exists." << std::endl;
    }
    else {
        std::cout << "File does not exist." << std::endl;
    }

    return 0;
}
```

Видалення файлу:

```
#include <iostream>

int main() {
    const char* fileName = "example.txt";

    if (std::remove(fileName) != 0) {
        std::cerr << "Error deleting file." << std::endl;
        return 1;
    }
    else {
        std::cout << "File successfully deleted." << std::endl;
    }

    return 0;
}
```

Отримання розміру файлу:

```
#include <iostream>
#include <fstream>

int main() {
    const char* fileName = "example.txt";

    std::ifstream file(fileName, std::ios::binary | std::ios::ate);
```

```
    if (!file.is_open()) {
        std::cerr << "Unable to open file." << std::endl;
        return 1;
    }

    std::streamsize fileSize = file.tellg();
    std::cout << "File size is: " << fileSize << " bytes." << std::endl;

    file.close();

    return 0;
}
```

Перенаправлення потоків:

```
#include <iostream>
#include <fstream>

int main() {
    std::ofstream outFile("output.txt");
    std::streambuf* coutbuf = std::cout.rdbuf(); // зберігаємо стандартний буфер cout

    std::cout.rdbuf(outFile.rdbuf()); // перенаправляємо вивід у файл

    std::cout << "This goes to the file." << std::endl;

    std::cout.rdbuf(coutbuf); // повертаємо стандартний буфер

    std::cout << "This goes to the console." << std::endl;

    return 0;
}
```

Зчитування та запис у бінарному режимі:

```
#include <iostream>
#include <fstream>

struct Data {
    int value;
    double price;
};

int main() {
    const char* fileName = "binary_data.bin";

    // Запис у бінарний файл
    std::ofstream binaryOutFile(fileName, std::ios::binary);
    if (!binaryOutFile.is_open()) {
        std::cerr << "Error opening binary file for writing." << std::endl;
        return 1;
    }

    Data data1 = { 42, 3.14 };
    binaryOutFile.write(reinterpret_cast<char*>(&data1), sizeof(Data));
    binaryOutFile.close();

    // Читання з бінарного файлу
    std::ifstream binaryInFile(fileName, std::ios::binary);
    if (!binaryInFile.is_open()) {
        std::cerr << "Error opening binary file for reading." << std::endl;
        return 1;
    }

    Data data2;
    binaryInFile.read(reinterpret_cast<char*>(&data2), sizeof(Data));
    binaryInFile.close();

    std::cout << "Read data: " << data2.value << ", " << data2.price << std::endl;

    return 0;
}
```

Використання буферу для зчитування та запису:

```cpp
#include <iostream>
#include <fstream>
#include <vector>

int main() {
    const char* fileName = "buffered_data.txt";

    // Запис у файл з використанням буфера
    std::ofstream outFile(fileName);
    if (!outFile.is_open()) {
        std::cerr << "Error opening file for writing." << std::endl;
        return 1;
    }

    std::vector<std::string> lines = { "Line 1", "Line 2", "Line 3" };

    for (const auto& line : lines) {
        outFile << line << std::endl;
    }

    outFile.close();

    // Читання з файлу з використанням буфера
    std::ifstream inFile(fileName);
    if (!inFile.is_open()) {
        std::cerr << "Error opening file for reading." << std::endl;
        return 1;
    }

    std::vector<std::string> readLines;
    std::string buffer;
    while (std::getline(inFile, buffer)) {
        readLines.push_back(buffer);
    }

    inFile.close();

    for (const auto& line : readLines) {
        std::cout << line << std::endl;
    }

    return 0;
}
```